MŰEGYETEM 1782

Design and implementation of critical infrastructure protection system

Dániel István Buza, Ferenc Juhász, György Miru
BSc. 3rd year

Supervisors:
Dr. Tamás Holczer, Dr. Márk Félegyházi
Department of Networked Systems and Services

2013

Összefoglaló

Dolgozatunk célja ipari irányítási és szabályozó rendszerek területén alkalmazott PLC eszközök ellen irányuló támadások detektálása és elemzése. Ennek érdekében alaposan megvizsgáltunk egy konkrét eszközt, valamint több már létező honeypotot. Az így megszerzett tapasztalatok felhasználásával tudtunk egy a publikusan elérhetőknél jobb honeypotot tervezni és megvalósítani. Megvizsgáltuk, hogy a létrehozott honeypot és a valós eszköz mennyire megkülönböztethetetlen egy külső támadó szemszögéből, és kielégítő eredményeket kaptunk. Továbbá célunk volt a konkrét eszközök biztonságos-ságának növelése azáltal, hogy feltérképeztük a gyenge pontjaikat és ezen ismeretek birtokában tűzfalszabályokat állítottunk fel a veszélyes támadások kivédésére.

Abstract

The aim of our paper is to improve the security of PLCs which are fundamental building blocks of critical infrastructures by detecting and analysing attacks. Therefore we analysed a physical device and several already existing honeypots. Based on our observations we were able to design and implement a new honeypot which is better in several aspects compared to publicly existing solutions. We analysed the indistinguishability of our honeypot and the real device from the perspective of an outsider attacker, and we received satisfying results. Furthermore we improved the security of the real device by finding its vulnerabilities and defining firewall rules to protect it from malicious attacks.

Contents

1 Introduction **4**

2 State of the art **6**

2.1 Virtual Honeypots From Botnet Tracking to Intrusion Detection . . . 6
2.2 Anti-honeypot technology . . . 7
2.3 Honeyd: A Virtual Honeypot Daemon . . . 8
2.4 HoneyC - The Low-Interaction Client Honeypot . . . 9
2.5 Honeytokens: The Other Honeypot . . . 10
2.6 Trend Micro articles about attacks against ICS/SCADA systems . . . 10
2.7 SCADA HoneyNet Project by Cisco . . . 12
2.8 SCADA Honeynet by Digital Bond . . . 13
2.9 Conpot by The Honeynet Project . . . 14

3 Implementation **16**

3.1 HyperText Transfer Protocol . . . 18
3.1.1 Observation of the PLC . . . 19
3.1.2 Simulating the HTTP service . . . 22
3.1.3 Comparing the simulated and the real behavior . . . 23
3.2 SNMP . . . 24
3.2.1 Introduction to SNMP . . . 24
3.2.2 Observation of PLC . . . 27
3.2.3 Simulating the SNMP service . . . 30
3.2.4 SNMP evaluation . . . 33
3.3 Communication with Siemens SIMATIC STEP 7 . . . 35
3.3.1 Siemens SIMATIC STEP 7 . . . 35
3.3.2 Observation of the PLC . . . 36
3.3.3 Implementation . . . 38

4 Testing **40**

5 Summary **41**

A

PLC snmpwalk output **47**

B

Simulated snmpwalk output **53**

1 Introduction

In recent decades, the development of computers and the range of problems they solve gradually increased. Supporting industrial processes with computers make the process cheaper, faster and improve the quality of the product.

Computers can be harnessed to perform various tasks, from design to production to the sale (including transport, advertising management too). The rest of our paper will focus on the electronic devices which are part of industrial process control systems. For example computers control pumps, boilers, turbines and many kind of industrial equipment.

The structure of industrial process control equipment (PLC) is very different from the traditional design of computers, since their function is completely different. The most obvious difference is that the PLCs have guaranteed response time. This property makes them usable in safety-critical systems. PLCs have different network interfaces to communicate with each other, controlled or regulated devices and other network devices. If a system has more than one PLC device then they can directly be linked to each other (similar to other networks the topology of these networks can be various).

The management and maintenance of the PLCs are mostly done over the network. Manufacturers strongly suggest that users should connect these devices only to trusted networks. For example, only a network that is not connected to the Internet. In the beginning the number of attacks against PLCs was very low. Unfortunately applying these PLC devices directly to unprotected network become a common practice. This can be useful during the maintenance (it can be done from anywhere even from other countries or continents). The same logic led some users to disable the PLC's built-in basic security mechanisms (for example, use a well-known default password).

Because the PLCs are unprotected they become popular targets for attackers. As explained sometimes it is not even a challenge to take over the control of a PLC[27]. As these devices control industrial processes the failure of these can mean that human lives are in danger. It can cause significant environmental and financial damage as well.

To protect against possible attacks we need to know that which techniques are useful against the PLCs. (Remember, often time-critical systems are concerned, even a short pause of control can make serious consequences.)

Honeypot is a system which seems to be a PLC based on network behavior, but actually it is a trap for attackers. One of the honeypot's aim is maintaining the attacker's interest and thus observe the methods against real PLCs. Thus, previously unknown attack methods can be revealed and it will allow the actual PLC's safety improvement.

From the point of interaction there are three kind of honeypots. Low-interaction honeypots simulate only basic network services (or only a base part of the basic network services). High-interaction honeypots simulate different complex network services. Hybrid honeypots usually simulate different services on different interaction levels. The advantage of the low-interaction ones is that they are easier to design and maintain. They can be more stable but they are easier to discover. The high-interaction honeypots are able to keep the attacker's attention longer. But they are much harder to implement because they have to implement the already known bugs and incorrect activities as well. Hybrid honeypots try to combine low and high interaction parts to get the advantages of both.

Of course it is impossible to emulate incorrect activities which are not yet known (they are also called 0-day vulnerabilities). But it is possible to connect real PLC devices which do not control any real actuators (but they seem to be) to the Internet and observe the generated traffic. In this way it is possible to learn real techniques from a real attacker and there is almost no chance to be discovered (because the attacker communicates with a real PLC and the controlled processes are emulated). Unfortunately this solution is very expensive, that is why we looked for alternative solutions.

In the present study we examine the honeypots available today with a special focus on industrial control systems. In addition, a specific PLC device also will be tested in order to detect possible weak points and implement a high-interaction honeypot. We are going to handle the different services completely separated in order to find the best way to implement them. And then we are going to integrate the service simulators to one honeypot system. We will develop a system which is complex but easy to configurate so it will be able to simulate different (but similar) PLC types.

In chapter 2 we examine the related work. In chapter 3 we discuss the complex method of developing our honeypot service-by-service. In chapter 4 we show the results of the tests which are comparing our honeypot to the specific device. Finally in chapter 5 we conclude our paper, and give a brief overview of some possible future work.

2 State of the art

In this chapter, we want to present the current state of the art of honeypots. Honeypots are widely used in many cases, where standard network intrusion detection systems could not provide the proper information to ensure the security of a device connected to the network. There are several books and papers discussing various honeypots, their advantages and disadvantages. We would like to give a short description about their results.

2.1 Virtual Honeypots From Botnet Tracking to Intrusion Detection

The book Virtual Honeypots: From Botnet Tracking to Intrusion Detection[34] by Niels Provos and Thorsten Holz provides a comprehensive description about the basics of the honeypots, as well as case studies where real honeypots were compromised by attackers. The book starts with the basic definition of honeypots: "an information system resource whose value lies in unauthorized or illicit use of that resource". Network intrusion detection systems are less efficient than honeypots, because the new encryption systems are providing more protection to the attackers. They are also suffering from high false positive rates, so a new system is required to detect and examine the methods of the adversaries. As stated in the book, honeypots are able to provide these services. For example, we can log keystrokes (which we could not have done by monitoring encrypted network traffic), or even detect new vulnerabilities, new methods of the attackers.

The book contains a detailed description of the two main types of honeypots: low- and high interaction. High interaction honeypots provide full access to a complete system. Thus, an adversary can take total control over the honeypot, and we can examine its tools and techniques. So the main advantages of these honeypots are that we can follow the attackers steps, how he takes control over the honeypot, and what is he doing after that. A considerable disadvantage of these honeypots is that the attacker can initiate malicious activity from our honeypot to other devices, but this is the price for detecting the adversarys behavior. High interaction honeypots can be real physical systems, like a computer or a router, but there are virtual honeypots. Virtual honeypots are easy to set up, and many of them can be used on one host machine, but they also have disadvantages, for example the attacker can detect that it is not a real machine, and try to mislead the observer. Virutal honeypots can be set up based on VMWare or User-Mode Linux. We can read also about Argos: this high interaction honeypot was developed by researchers from Vrije Universiteit Amsterdam, and is able to detect zero-day attacks (attacks for which no patch yet exists). The detection process is called dynamic taint analysis, where the malicious input from the attacker gets marked as tainted. As we saw earlier, high-interaction honeypots bear the risk that the attacker takes full control over the device, so we must ensure that they are not used for further attacks by the adversaries. One of the easisest ways of safeguarding our honeypots is the Honeywall by the Honeynet Project. The book also presents this software, how it is installed and configured in order to manage and protect our honeypots.

We can read also about low interaction honeypots in the book. As mentioned also in the introduction, these honeypots do not provide complete access to the operating system. This is an advantage and a disadvantage at the same time. It is good, because the attacker will never have complete control over the honeypot, but this makes the system less desirable to attack. The book also contains instructions about how to install and setup various low-interaction honeypots, for example Tiny Honeypot, Google Hack Honeypot or PHP.HoP. Safeguarding low-interaction honeypots is also important despite the fact that there are less threats to worry about in this situation, since its harder to abuse these systems. But if there is a vulnerability in the honeypot itself, it could turn into a high-interaction honeypot, and attackers can use it to malicious activities. The authors also provide solutions for enhancing the protection of the low-interaction honeypots.

We can find information in this book about a framework called Honeyd, which helps the setup of networking preferences of the honeypots. More information about this service will be provided later in this chapter, based on another paper[33]. The authors present their results about detecting honeypots, this topic is also related to our poject. Here the authors provide methods to detect low- and high-interaction honeypots using simple tools on Linux. There are also several instructive case studies in the book, where high-interaction honeypots were compromised, such as a Red Hat 8.0 or a Windows 2000 honeypot. In conclusion, this book gives an extensive description of the basics of the honeypots, and it is a good starting place to learn the subject of honeypots.

2.2 Anti-honeypot technology

The paper Anti-honeypot technology[29] by Neal Krawetz is discussing the topic from the adversarys side. As the title suggests, he writes about the detection of honeypots using the software called Honeypot Hunter by Send Safe. Since spammers continually scan the Internet for proxy relays, honeypots are a direct threat to them, because they can reveal the spammers true identity. As stated in this paper, spam developers are generally reactive, not proactive: they only change their methods when it is really needed. The development of this project indicates, that spammers are aware of the threat that honeypots mean to them. Honeypot Hunter uses simple methods to detect these honeypots. It tests an open proxy connection, and based on the response, classifies it as safe (good), bad (failed), or trap (honeypot). The Hunter opens a fail mail server, then it attempts to proxy back to itself through the server. If the server responds that it connected successfully, but the false mail server does not receive a connection, then it is likely a honeypot. As the author states, honeypots can be prepared for these systems, and they can be supported with anti-detection methods. This project is a good demonstration that it is important to create better and better honeypots, because anti-honeypot systems are evolving and they can suppress the positive effects of the honeypots.

2.3 Honeyd: A Virtual Honeypot Daemon

As previously mentioned, there is a paper by Niels Provos about a framework called Honeyd, which helps the setup of virtual honeypots[33]. Creating physical honeypots can be often time-consuming and expensive, but virtual honeypots are generally easy to setup and maintain, however, they also have disadvantages as we saw it earlier. Honeyd is a tool for creating proper network environment for honeypots. It is basically a daemon, which simulates the TCP/IP stacks of operating systems, thus fools fingerprinting tools like Nmap. Honeyd supports TCP, UDP and ICMP, and is capable of creating complex virtual network topologies. The paper explains how the incoming and outgoing packets are processed. Incoming requests are processed by a central packet dispatcher. The dispatcher calls the protocol specific handler of the corresponding honeypot configuration (or the default one if there is not any). The daemon can also establish connections to arbitrary services over TCP and UDP, or it can forward the connection request to a service running on a honeypot, to a web server for example. For outgoing requests, there is a personality engine which contains the networking behaviour of the simulated honeypots. This information is based on the Nmap fingerprint scan. Before sending a packet, it passes through the personality engine. The author explains the configuration of a real system using templates: how to setup a network and assign configurations to virtual network devices. In conclusion, Honeyd is an effective tool to setup and operate virtual networks for honeypots, it successfully mimics the network stack behaviour of several operating systems, thus fools fingerprinting devices.

2.4 HoneyC - The Low-Interaction Client Honeypot

This paper by Christian Seifert, Ian Welch and Peter Komisarczuk is about client honeypots[46]. So far we referred to honeypots as server honeypots, which attracts the attackers by providing different services, so we can gather information about their activities. Client honeypots are designed to interact with servers, in order to detect malicious behaviour. For example, if a client connects to a web server, the response of the server might contain code that is targeted at exploiting a vulnerability of the browser on the client. Like the server honeypots, there are also low- and high-interaction client honeypots. High-interaction client honeypots are traditionally run a web browser, so they focus on malicious web servers. They usually monitor the changes made in the system after a request has been made, and detect the hostile activity by this way. Honeyclient or Honeymonkey are such high-interaction client honeypots. As the authors stated, these high-interaction honeypots can be expensive and complex, since they are running a real operating system and monitor the changes in the whole system. Thus, they often suffer from performance issues, which make the low-interaction honeypots more desirable. These client honeypots are usually virtual honeypots instead of using a real system. This makes them cheaper and they are less vulnerable to the attacks. But they have also disadvantages, for example they might not be able to detect some unknown exploits that would be identified by a high-interaction honeypot. So the relationship between the low- and high-interaction honeypots is similar to the interaction level of server honeypots. As the authors stated, a low-interaction client honeypot should have three main tasks: create a queue of server requests, create these requests with several algorithms (like crawling or search engine integration), and analyze the system and the server response after the interaction with the server. HoneyC is a platform independent framework and completes these requirements. Its three main parts are the Queuer, the Visitor and the Analysis Engine. HoneyC focuses on the HTTP 1.1 protocol, so it creates HTTP requests, and analyzes the corresponding responses. The article contains also the results of querying 2000 websites, and analyzing the responses. It is not surprising, that the low-interaction honeypots have a better performance than the high-interaction ones, and the authors were able to detect malicious web servers as well. In conclusion, like server honeypots, client honeypots are really important to protect our systems. They provide an effective and safe method to detect hostile services, thus we can expect the advancement of these systems.

2.5 Honeytokens: The Other Honeypot

So far we mainly discussed honeypots that are some sort of computers, but this paper by Lance Spitzner changes this concept[47]. Since honeypots by definition are information system resources, they do not have to be a computer. A honeytoken is exactly this: a honeypot, which is not a computer, instead it is some type of digital entity. A honeytoken can be a credit card or an Excel spreadsheet, they can come in many forms, but their concept is the same as the honeypots: their value lies in the unauthorized use of the resource. Because of this, as the author states, they have the same power and advantages as traditional honeypots, since no one should be using or accessing them. For example, a hospital could use these honeytokens for preventing legal issues. Only authorized people should have access to medical records, and if a hospital wants to know if one of their workers is looking into files where they should not, honeytokens are a good way to find this out. Another good example is a database with credit card entries. Mixing honeytokens into valid credit card numbers can help to detect malicious activity in the system, since no one should access the false credit card records. As we can see, the honeytokens are really flexible and provide an easy way to protect systems with sensitive data. Compared to honeypots, they are much simpler than even the low-interaction virtual honeypots, even a short string or file can be a honeytoken, and if they are accessed, we would know that immediately. The cost of the honeytokens is also minimal: there is no technology to deploy, no vendors to contact, no licenses to update. As the author states, combining honeytokens with other solutions can make them even more efficient. For example, a honeytoken can inform us about a potential unauthorized behaviour, but with other devices, we can make sure that it is really an attacker with malicious intent, or just a worker looking where he or she should not. In conclusion, honeytokens are really cost effective, simple and efficient way of creating honeypots, we expect to see a great progression in the future.

2.6 Trend Micro articles about attacks against ICS/SCADA systems

Who's really attacking your ICS equipment?[51] is an article wrote by Kyle Wilhoit, member of the Trend Micro Forward-Looking Threat Research Team. This paper discusses the security of ICS systems, and presents results of attacks against a honeypot system. As the author states, when these systems were first brought into service, security was not a concern. SCADA systems were not even able to connect to the Internet, the physical separation addressed the need for security. However, nowadays many of these systems are exposed to the Internet (one can easily find embedded systems on the web with performing Google-dorks searches), which proves the insecurity of these systems. The research team developed a honeypot architecture that emulates several ICS/SCADA devices. The objective was to find out who and for what purpose is attacking these ICS systems. The honeypot architecture contains a high-interaction honeypot (a PLC system running on a virtual machine), a pure-production honeypot (a server which mimicks the human machine interface of a PLC), and also a real PLC, which can be considered as a pure-production honeypot. In addition to these honeypots, low-interaction honeypots were also set up. After setting up the honeypot system, the devices were seeded on Google, named for SCADA-1, SCADA-2 etc. so they would draw the adversaries attention. In 28 days, 39 attacks were reported from 14 different countries. It is important to note that port scans or automated attack attempts (like SQL injection) are not reported, only the attacks which would be a real threat against ICS/SCADA systems. The author recommends several safety instructions to protect these SCADA systems.

The SCADA That Didn't Cry Wolf[50] is the continuation of the previous paper. As the author states in the beginning of this article, the security of SCADA systems is still a concern everywhere in the world. While technological advancements have been made in the deployment of the these systems, their security does not improved. One of the biggest advancements is the cloud-based deployment. Several security concerns related to the cloud-based SCADA deployment have been reported worldwide. Since SCADA systems are widely used, especially in the USA, in China and in Japan, many countries started to implement standards to secure these systems. But, as the author states, risks and threats are becoming common to SCADA systems. For example, human machine interfaces which provide access to SCADA devices are a huge threat to these systems. An attacker can use traditional web application vulnerabilities (like SQL injection), or even unpatched operating system exploits. In addition to these risk, data historian informations can be also a threat to SCADA systems. A data historian is basically a centralized database for logging process information, and it is important to protect this information from unauthorized access.

As we saw in the previous article, the research team has set up a honeypot architecture to examine attacks against SCADA systems. While that honeypot deployment was successful, the team wanted a bigger data sample to better represent global perspective. They improved the architecture with several already-existing and also newly-created tools and scripts. The main goal was to make the system a believable, fully mimicked version of an ICS/SCADA system. The team also improved the identification of the attacks with special tools. In order to create global attack detection, the team created a honeynet from multiple honeypots. A total of 12 honeypots were deployed, and they worked separately in different countries like the USA, Russia or China. The attackers used the website ShodanHQ for reconnaissance and performed several port scans on the honeypots. Some of the attackers even expanded their port scanning selection, and they did not even use slow scanning to reduce getting noticed. Over three months, the research team observed 74 attacks from 16 different countries. Out of these 74 attacks, 11 were considered critical, which means it can cause a catastrophic failure of an ICS device's operation. Like in the first honeypot architecture, only targeted attack were considered real attacks. The author also presents the attack origin and type breakdowns in the paper.

In conclusion, these two papers are a good presentation that ICS systems are constantly threatened. ICS systems possess a great responsibility, they are used in many countries in automation or manufacturing systems. It is important to examine the methods of the attackers and develop security solutions based on the examination.

2.7 SCADA HoneyNet Project by Cisco

The Scada HoneyNet Project[49] was started in 2004 by Cisco Critical Infrastructure Assurance Group (CIAG) and was discontinued in 2005. It consists of a set of python scripts, each of them implementing a service of the simulated PLC. The project heavily utilises Honeyd[31], which is a small daemon that creates virtual hosts on a network. The hosts can be configured to run arbitrary services, and their personality can be adapted so that they appear to be running certain operating systems. The Honeyd daemon can be set to simulate a computer that has the OS fingerprint of a PLC and runs the given Cisco scripts on the appropriate ports. With the help of these scripts the Honeyd PLC realises a Telnet, FTP, HTTP and Modbus services.

The Telnet service responds only to ls and cd commands, but their implementation is rather bugous, their effect is not the same as the original commands'. The user initially has the 'Hostname#' prompt, and using the 'cd *param*' command just concatenates a white space and the *param* to this console prompt. Even with the '.' and '..' parameters the effect is the same. Meanwhile ls returns a constant string saying the user is in the root directory, whether he "changed" the directory with cd or not. There's also a 'help' command implemented that lists all the usual Telnet commands, but none of them actually works.

The FTP service has more commands realised than the Telnet but in most cases these commands lack the correct functionality and their effect is not the desired one. The help command lists roughly 30 commands, but only user, pass, cwd, list, quit, port and syst are implemented. The 'user' prints the "331 Guest login ok, send your complete e-mail address as a password." predefined string, while the 'pass' command prints the "230 User Logged in" string, typing any parameter after the user and pass commands has no effect, the returned value is always the same. The cwd command changes the directory almost the same way that the Telnet script's cd command does, the 'cwd *param*' returns the "250 Changed directory to *param*" message. The list, port and syst commands also print a static string, and there is a quit command, which closes the session. Both the Telnet and FTP scripts have a logging function that logs the entered command (without the parameters) and the current time in a file.

The Modbus script can accept Modbus packets, than returns them to the sender without any modification or interpretation. It has the same logging functionality as the Telnet and FTP services. Last, there is an HTTP script which returns a static HTML page which contains three dead links.

In summary these scripts seem unfinished, the services are only partially implemented and the realised functionality is nor realistic neither interactive. Also, it is worth to mention that bugs and mistakes are present in the code, for example, if the log file does not exist, the script doesn't create it, instead it throws an unhandled exception. Even an inexperienced attacker would notice in seconds that the simulated PLC is not real, and the information provided by the logs can not be used to uncover the identity or the methods of the attacker, therefore the SCADA HoneyNet Project clearly fails to reach its goals.

2.8 SCADA Honeynet by Digital Bond

The SCADA Honeynet[9] is maintained by Digital Bond and is freely available from their website. It utilises two virtual machines one of which is a Generation III honeywall (by The HoneyNet Project[14]) extended with Digital Bonds Quickdraw IDS signatures[9]. The purpose of this unit is to monitor all network activity to identify and log every malicious attack that may occur against the simulated PLC. The other virtual machine simulates a popular PLC that runs five services (FTP, Telnet, HTTP, SNMP, Modbus TCP), the FTP, HTTP and Modbus services are implemented by different Java applications while the Telnet and SNMP services are realised by python scripts. The core of the VM is Honeyd that routes the created virtual host's network traffic (the data streams and datagrams) from the appropriate ports to these applications and scripts.

The telnet script presents a VxWorks[13] telnet banner which is typical for the simulated PLC, and also prompts the user for login information, however no username-password combination is considered valid, so the script always returns an authentication error and logs the login attempt in a text file. The SNMP service is implemented by the SNMP script that is shipped with Honeyd and configured to not accept any community string.

The Java application (called iFTPd.jar) that implements the FTP server is a tinkered version of Independent FTP Daemon[4]. This is an open source FTP server, according to the manifest file the simulator is based on the iftpd 1.5 version. Just like the telnet script the FTP service is configured to present a VxWorks ftp banner. It also does not authenticate any username-password combination, but logs them in a text file. The modbus service is implemented by ModSim.jar which is just a frame for jamod.jar which contains the net.winpi.modbus package. The jamod[52] (Java Modbus Library) is a free and open-source JAVA Modbus project, that is available from sourceforge.net. Jamod.jar is a Java modbus implementation that creates XML based logs. The HTTP service is provided by FizmezWebServer[22], which is a simple JAVA based web server. It was a GNU licensed project by David Bond. The web service presents a simple html page identical to a Schneider Electric PLC web-page.

The Digital Bond's SCADA Honeynet is a huge improvement over the Cisco Honeynet Project. With the returned service banners and OS fingerprint it can make scanning and information gathering tools (such as nmap or nessus) believe that it is a real PLC, thus it can be effective against automated attacks and tools. However, the simulated services provide very little interaction, and they might not be able to keep an attacker attracted for long enough to uncover new targeted PLC attacks.

2.9 Conpot by The Honeynet Project

The Conpot project[1] by The Honeynet Project[5] was released in May 2013, and it is available for everyone from their website. Conpot is an ICS honeypot with the goal to collect intelligence about the motives and methods of adversaries targeting industrial control systems. The honeypot realizes two major ICS protocols: Modbus and SNMP. There is also an HTTP service, and a logging system in the honeypot. These services are implemented with Python scripts. The honeypot emulates a Siemens S7-200 CPU type PLC by default, but it can be easily reconfigured through various profiles.

The SNMP module of the default profile contains common variables such as system uptime or system description. When using SNMPv1, these variables can be accessed with a predefined community string, and SNMPv3 is also supported, where the user can access these with the proper authentication (user name and authentication key). There are less variables in the honeypot than in a real, working PLC, but the SNMP module can be configured easily: the management information base (MIB) can be modified, so even more variables can be accessed through SNMP. The community strings for SNMPv1 and the authentication for SNMPv3 can also be customized.

The Modbus service of the honeypot can be configured with various slave devices. In the definition of the slaves, there are input and output blocks. Both type have starting address and size. The input can be analog input (measurement for example), or binary input (power on/off for example), and these values can be randomized. The default profile contains two slaves, but it is easily customizable to have more or less slave devices.

There is also a web server in the honeypot, but when trying to access the website, it gets into a redirect chain. The website contains the system description and the system uptime, and these are queried by SNMP requests. After these requests, an HTTP 303 (See Other) error occurs, but the location in the error description stays the current site, so the process starts over again. Beside the main page, there are also html files for various HTTP errors like 403 (Forbidden) or 404 (Not Found), but they don't seem to work either.

Conpot has also a logging service. This service logs the events of the HTTP, SNMP and Modbus services. Every logged event begins with the system time with millisecond accuracy. HTTP logging is pretty simple: when a request occurs, the source address, the request type, and the requested resource will be saved in the log. According to the source files, the same occurs when a response is sent (except for the type), but there is no response yet because of the redirect chain explained before. Logging SNMP requests is quite similar, the version, the source address, the source port, and the requested object is logged. When the honeypot sends an SNMP response, then the destination address and port, the requested variable and its value will be saved in the log. Logging Modbus activity is even more detailed. When a Modbus client connects to the Conpot, the client's address, the port, and the session ID is stored. After that, when Modbus traffic occurs, the logger saves the source address, the session ID, the slave device ID, the request and the response to this request. There's also a log entry with the session ID when a Modbus client disconnects from the server.

The base concept of the Conpot by The Honeynet Project is well, but it still have some defects which really need fixing. The honeypot is easy to install and use, but to reach it's full potential, it needs a lot of customization. The HTTP server is not working yet, but it's an important part of a honeypot, because it is a major attack surface where we can examine the behaviour of an attacker. Because of the little interaction the honeypot provides, it's possible that the attacker moves on before it's methods and behavior can be discovered.

3 Implementation

In this chapter we discuss the method of developing our honeypot. The real PLC device is a Siemens Simatic 300(1) type and have an IM151-8 PN/DP CPU. It uses firmware version 3.2.6. Our goal is to develop a high-interaction honeypot which appears identical to the real device from an attacker's point of view. We will develop a system which is complex but easy to configure so it will be able to simulate different (but similar) PLC types.

Before, the implementation phase began a lab environment was set up. The PLC was installed in the same subnet as a virtual machine that run Backtrack R5[15]. This machine was used to further discover and analyse the PLC. The initial nessus[16] and nmap[17] scans showed that the PLC runs four services these are the http, https, isotsap and snmp. The Siemens' port forwarding guide[19] for the PLC, also confirmed that the PLC is capable of running these services. The latter sections will discuss the exploration and implementation of these protocols in details.

When the PLC does not have an IP address configured, it looks for a master on the LAN. This is done bye broadcasting different Ethernet frames. The protocol of these is part of the PROFINET[18] standard, which is a closed commercial standard. It defines protocols for real time and isochronous real time communication over TCP/IP and Ethernet. In order to simulate this behaviour, large amount of broadcast traffic was captured and analysed. It was found out, that the capture only contains three different type of frames, in which the only changing field is a sequence number and a CRC checksum. After this, a python script was written that creates and broadcasts these packets. It increments the sequence fields and uses the CRC values from the original packets. The length of the cycle is nine. Later it was discovered that the PLC stops sending these frames after it receives an IP address, so this script is not part of the assembled honeypot. Still, it can be used to simulate a PLC on LAN that is looking for a host.

After each of the services, mentioned above, were implemented, they were integrated onto a virtual machine to create the honeypot (the abstract model of the honeypot is presented on 1). The VM runs a minimal version of Ubuntu Linux, that only has the necessary services and libraries installed. There is also a bash script written, that can start, restart or stop the honeypot. It is run by *(initd)* on every start up. Every simulator has its own way to fine tune it, however there is a main configuration file of the honeypot. In this file global settings can be set, such as the IP and network interface that the honeypot is being run on. The startup-script reads these values and configures each service simulator before it launches them. The script also sets iptables rules to block all incoming connection that are not destined to one of the simulators. The TCP connections are refused by a TCP reset, just like on the real PLC. The UDP datagrams that are not sent to the

SNMP service are simply dropped. The script starts tcpdump to capture the network traffic on the honeypots interface. The final step is to change the behaviour of the TCP/IP stack. The files in /proc/sys/ipv4/ provide an interface for this on Linux. These files contain values, that can be read or written and control the operation of TCP and IP. The IP ttl is set to 30, and the MTU is 1518, also the PLC has a fixed TCP window size, which unfortunately can not be exactly set, because of the limitations of the /proc file system. Some of the TCP related values are compiled into the kernel, so they cannot be changed through these files. Currently this is a limitation of the honeypot. In the future this issue needs to be addressed.

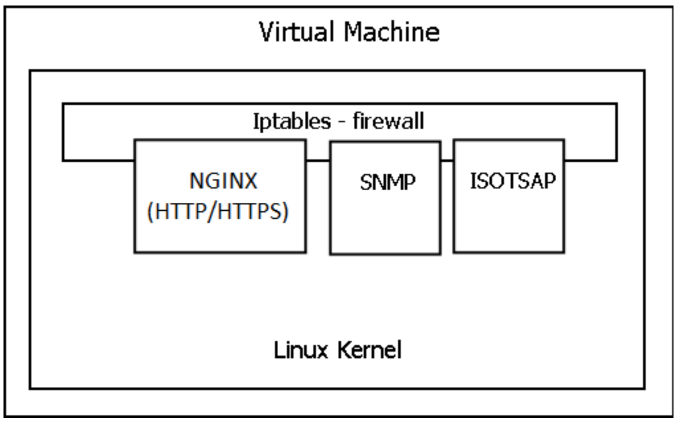


Figure 1: The model of the integrated honeypot

3.1 HyperText Transfer Protocol

HTTP is one of the most common Internet protocols. As we can read in the introduction of RFC 2616[26] HTTP is application-level protocol for hypermedia information systems. The first version of HTTP, referred as HTTP/0.9 was a simple protocol for raw data transfer. HTTP/1.0 is defined in RFC 1945[21] improved the capabilities of the protocol by allowing messages to be in MIME-like format containing meta-information about the data transferred and modifiers on the request/response semantics. HTTP/1.0 does not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, or virtual hosts. HTTP/1.1 is designed to improve the capabilities of the protocol by including more specific information about this form of communication.

Usually HTTP communication runs over TCP/IP connection. By default it uses TCP port 80 but it can use other ports as well. HTTP interaction is always initiated by the client. It creates a new connection to the server and posts a request message. It is not necessary to request data; the request can query the methods that the server supports. The client can send meta-information about itself (for example client program name, version, etc.).

After getting a request the server posts a respond with the requested information. It also can send meta information about itself (server program name, version, location of server, etc.). And then the connection closes. If there is no respond to a request for a while then the connection closes itself and it posts the client a request timeout (status code 408) message. HTTP/1.1 allows to keep alive the connection so multiple requests and responses can be sent over one connection.

The communication can be listened and played back by a third party. Of course in many cases the captured information is useless (for example visiting public websites). But in many situations it is not acceptable to allow any third party to record the communication (for example while making bank transfers or handling kind of secure information). Therefore it is necessary to improve the security of the protocol.

HTTP Secured (HTTPS) is an extension of HTTP service. It is not an independent protocol it is layering HTTP over SSL/TLS protocol. It is a simple way of improving the security of HTTP protocol. Mostly it uses port TCP 443 and usually the HTTP clients and servers also able to communicate over HTTPS. Using SSL protocol only guarantee that the communication between the client and *a* server cannot be observed by a third party. But it does not necessarily guarantee that the client reached the server it *wanted* to reach.

Every server have to generate a certificate which can be self-signed (which is only use for testing a server application during development) or signed by a public certificate authorities (for example Thawte). Only these authorities can create trusted certifications. With a trusted certification the server can guarantee the client that it is *the server it wanted* to reach.

As it was mentioned before in many cases there is no benefit by using HTTPS instead of HTTP. Maybe this is the reason why only the quarter of the most popular websites implemented HTTPS in the first half of 2013 according to SSL Pulse of Trustworthy Internet Movement[11]. The research included over 160,000 web sites and it clearly shows the increasing percent of the secure web sites. Over 11 months from 2012-10-04 to 2013-09-02 the percent of secures web sites increased from 13.6% to 24.6%. It is a very surprising fact that over one month (to 2013-10-02) it increased to 49.4%. Summary the increscent over one year the number of secured web sites become nearly four times greater.

There are a lot of new standards and features of using HTTP which is not in the official protocol definition. The different implementations of actors (both servers and clients) usually implement different parts of the protocol and its features. Therefore to make the honeypot's HTTP service similar to the PLC's HTTP service first we have to observe the behavior of the PLC device, mapping its capabilities and examine the exact way of communication it uses.

3.1.1 Observation of the PLC

We found that TCP port 80 is open for HTTP service. First we used a simple HTTP client (Firefox 14.0.1) to examine the served pages. The first page redirects the client to a welcome page. On the welcome page there are a few links to the manufacturer's website, we can select the preferred language and we can navigate to the rest of the pages. From the welcome page we can get to the start-page where there are several information about the PLC. We can check it's module name, module type and it's status. Also there is a graphical representation of the PLC. We can see the graphics of the front panel and we can check the status of the LEDs and the mode switch.

This page also informs us about the time set on the PLC and we can print the contents. There is a login field where we can log in with the user name and password combination which was set by the configuration of the PLC. After login we can reach some other pages which provide some specific information about the device (firmware version, serial number, etc.) and we can also find some statistics about network activity.

A very conspicuous fact about the site is the request which can be read in the navigation bar of the browser. Loading any page (except the welcome page) request the same file with different parameters. For example the request for the start page is the following:

```
10.105.0.47/Portal/Portal.mwsl?PriNav=Start
```

Because the responded page depends on the parameters of the requested URL we can say that there is a server-side processing. Thus we can say the web server of the PLC generates the content dynamically. Another noticeable fact is the extension of the requested file, because `.mwsl` is not a common extension.

After some searching we found that `.mwsl` probably stands for Mini Web Server Language. It shows that the server uses the MiniWeb project[6]. This project is owned by Stanley Huang. His aim was to develop a small HTTP server with high efficiency and high portability written in C language. It is free and open source. It serves static pages so we can say that the PLC probably use an improved application which is based on this project.

Because HTTP is a text based protocol it is easy to observe the headers of the communication. To examine the header content we send requests manually using telnet. First we should find out the capabilities of the server with sending it an options command:

```
OPTIONS * HTTP/1.1
```

The response contains status code 405 (method not allowed) but it also tells us that the server will response to only GET and POST methods:

```
HTTP/1.1 405 Method Not Allowed
Content-Length: 0
Connection: close
Allow: GET, POST
Content-Type: text/html
```

It is not necessary to implement the HEAD method because the only difference of GET is that the response must not contain the message-body just the header and the meta-information. Despite it is not listed in the allowed method list we found that it is working correctly. Our request was:

```
HEAD * HTTP/1.1
```

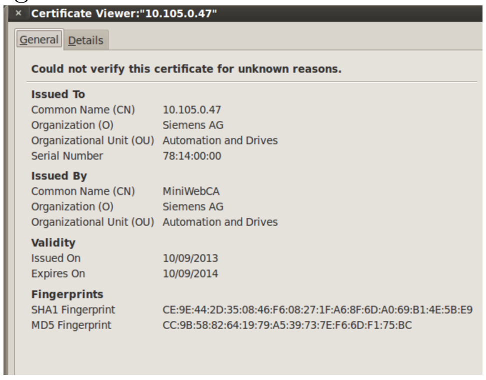
And the response was:

```
HTTP/1.1 200 OK (65519)
Content-Type:text/html
Content-Length: 15
Connection: close

200 OK (65519)
```

It is important to notice that there is no information in the headers about the server or the device. It contains just the necessary fields but no optional.

Figure 2: Details of the PLC's certificate



Could not verify this certificate for unknown reasons.

Issued To

Common Name (CN)	10.105.0.47
Organization (O)	Siemens AG
Organizational Unit (OU)	Automation and Drives
Serial Number	78:14:00:00

Issued By

Common Name (CN)	MiniWebCA
Organization (O)	Siemens AG
Organizational Unit (OU)	Automation and Drives

Validity

Issued On	10/09/2013
Expires On	10/09/2014

Fingerprints

SHA1 Fingerprint	CE:9E:44:2D:35:08:46:F6:08:27:1F:A6:8F:6D:A0:69:B1:4E:5B:E9
MD5 Fingerprint	CC:9B:58:82:64:19:79:A5:39:73:7E:F6:6D:F1:75:BC

We found that port TCP 443 was also opened. So the device implements HTTPS service as well. There is no difference of the served data and pages. The device uses a self-signed (not trusted) certificate as it is shown on Figure 2. The certificate is issued to Siemens AG organization issued by the same organization with MiniWebCA common name. The signature algorithm is SHA-1 with RSA Encryption and it uses an 1024 bits long public key. Observing different PLC devices we found that each device use a unique certificate (instead of using the same certificate on every device).

During the test we found that sometimes the PLC is not responding. The connection between HTTP traffic and not responding become clear when we found that requesting `Portal.mwsl` without parameters causes a complete crash of the system. After receiving the specificated request the PLC device hangs: it does not react to any network activity and it has to be manually reseted.

We guessed that this phenomenon is caused by a memory leak. We observed the miniweb server (*the version which is available on internet*, not *the* version which is running on the real device) with valgrind[12] to find if there is any memory leak. According the valgrind's result there is no memory-leak during normal running mode (with receiving any right or wrong requests). We have to note that the real device uses a modified version of miniweb. It contains the methods which are necessary to generate the pages dynamically. These methods have to process the parameters of the requests so we can say that probably the extension methods cause some memory-leak or null pointer dereference which leads to the crash.

To improve the safety of a PLC it is strongly suggested to use a firewall to filter HTTP requests and drop the ones which contain `Portal.mwsl` without parameters. This function can be implemented by `iptables` as well using `-m string` option.

We found this phenomenon by ourselves and later we found that others also noticed the problem a few months earlier.[2]

3.1.2 Simulating the HTTP service

As it was mentioned before the device uses a server based on the MiniWeb Server Project. Therefore we also use a server based on MiniWeb. The first change we had to implement is the form of the HTTP responses. The default header of the response mentions several information that the real device does not (e.g. the name of the server, the default cache-control rule) in the following way:

```
HTTP/1.1 200 OK
Server: MiniWeb
Cache-control: no-cache
Pragma: no-cache
Connection: keep-alive
Accept-Ranges: bytes
Last-Modified: Tue, 23 Jul 2013 08:45:49 GMT
Content-Type: text/html
Content-Length: 268
```

We changed the source to get the expected header containment. The welcome page is contained in `Intro.mwsl` and the start-page is generated from `Portal.mwsl`. Without logging in only the welcome page and the start-page can be requested. So we had to copy the two reachable pages. We used the `wget`[3] tool to copy the page content.

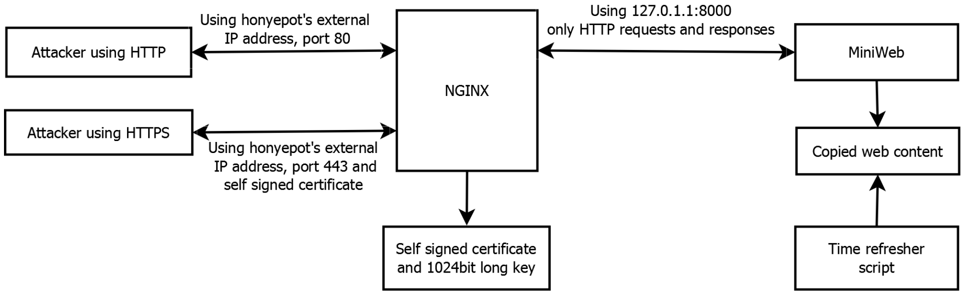
We had to manage some changes in the copied files to remove all links to the real device. And we rewrote the login mechanism (there is no name and password check) to simulate that there is a password but the attacker failed to guess it. After the changes we were able to simulate the HTTP service with static pages.

The pages also contain timestamps. We wrote a bash script to replace time and date data in the used file. It copies the file to a template file and during the copy it changes the time-stamp. Then it replaces the original file with the copied file and during it changes the date-stamp. It only works once per second (to reduce the used CPU time):

```
#!/bin/bash
while:
do
   time="$(date +"%r" | tr '[A-Z]' '[a-z]')"
   date="$(date + "%m/%d/%Y")"
   sed "s/[0-9][0-9]:[0-9][0-9]:[0-9][0-9] [a-p]m/$time/"
      <Portal.mwsl >temp
   sed "s:[0-9][0-9]/[0-9][0-9]/[0-9][0-9][0-9][0-9]:$date:"
      <temp >Portal.mwsl
   sleep 1
```

We also simulate HTTPS service. We used OpenSSL[8] and generated an 1024 bit long key and a self signed certificate. We added the same meta-information (e.g. location, common name, organization) to the certificate that the device adds to it's self signed certificate. To simulate HTTPS we used nginx[7]. With suitable configuration it tunnels HTTPS traffic to the miniweb server over HTTP.

Figure 3: The HTTP/HTTPS service environment


Attacker using HTTP
Using honyepot's external IP address, port 80
Attacker using HTTPS
Using honyepot's external IP address, port 443 and self signed certificate
NGINX
Self signed certificate and 1024bit long key
Using 127.0.1.1:8000 only HTTP requests and responses
MiniWeb
Copied web content
Time refresher script

An another advantage of using nginx is that we can separate the HTTP traffic into two parts: traffic between the honeypot and the outside network and the traffic inside the honeypot (between nginx and miniweb). Thus the system becomes more configurable. Figure 3 illustrates the complete developed structure.

3.1.3 Comparing the simulated and the real behavior

We have no intention to hide the already known differences between the real device and our honeypot. In the following section we will mention the differences which are able to vanish by future work. Visually there is no difference between the websites. But on the honeypot's site it is impossible to log in because we pretend that there is a valid username and password combination but the attacker's guess was wrong.

On the honeypot's page there is no effect of selecting other languages (it only uses English language). We have to note that on the real device's site it is also impossible to reach any other offered languages because it does not have enough memory to contain the language files.

We do not simulate any changes of the visual presentation of the PLC on the site (e.g. the LEDs state never change). We can say that we pretend that the environment of the simulated device never changes. It can be believable that a device's environment does not have changes which change the PLC's running state. We do not simulate the already known bug by requesting `Portal.mwsl` without any parameters however a script can be easily added to stop all communication for a while after visiting the page.

3.2 SNMP

3.2.1 Introduction to SNMP

The Simple Network Management Protocol (SNMP) is an internet protocol which was created to manage and monitor devices on an IP network. SNMP is part of the Internet Protocol Suite, its different versions are defined in RFCs, and it is maintained by the Internet Engineering Task Force (IETF). SNMP is used to obtain administratively important information from network devices, including, but not limited to, routers, servers, switches, hosts and printers. In a typical SNMP conversation there are two participants, a manager, that queries the requests and a managed device, that replies to these. The managed device runs an SNMP software that is called the Agent. The Agent interprets the queries and returns the requested data to the manager.

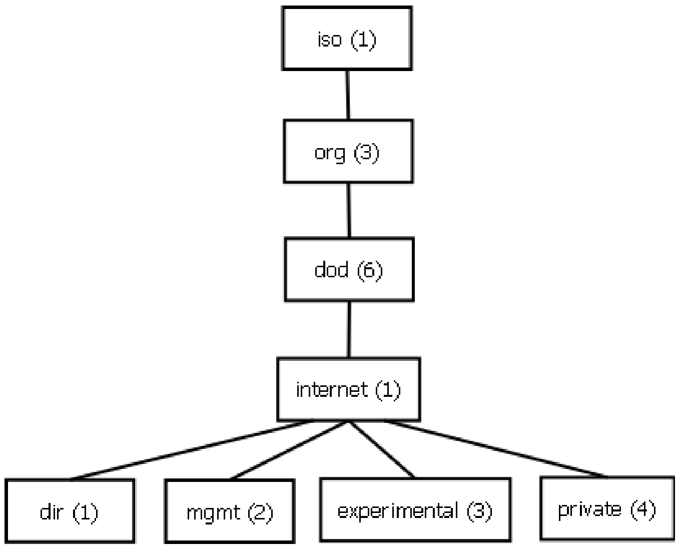


Figure 4: The structure of the OIDs

The SNMP standard doesn't define which information an Agent should offer, instead a hierarchic, highly customisable data structure is used. The data is divided into variables, and these variables are organised into a tree structure. Each variable is referenced by a globally unique identifier, called object identifier (OID). Just like the data, the OID itself is hierarchical, it is represented by a series of numbers divided by dots (e.g. 1.3.6.1.2.1.1.1.0) in which each segment represent a node on the appropriate level of the tree structure, as seen on Figure 4. The first segment selects the root of the data structure, the value of this can be 0, 1 and 2. OIDs beginning with 0 represent records defined by ITU-T, ISO defined OIDs begin with 1 and there is a so called joint-iso-itu-t database, these OIDs begin with

2. The whole hierarchy and the metadata (variable names, permissions and types) are described in Management Information Databases (MIBs) which use ASN.1 notation. During an SNMP request the manager queries an OID and if the Agent's MIB contains it, the Agent replies with the type and value of the requested variable.

The protocol also allows the modification of data through set requests. Each variable in a MIB has a permission flag, that can be set to read-only or read and write access. If an Agent receives a set request for read-only variable it replies with an error message. The SNMP protocol also permits the Agent to send information asynchronously to the manager, without the manager initiating the conversation. This is done with special messages called SNMP traps, the Agent can be configured to trigger these traps on special events such as reboot, interface down, etc.

Currently there are three major versions of SNMP respectively SNMPv1, SNMPv2 and SNMPv3. In SNMPv1 and v2 the messages are encoded by ASN.1 BER notation, defined by the International Telecommunication Union (ITU) in the x.680-683[37][38][36][35] and x.690[39] recommendations. This notation requires each field of the message to be preceded with bytes that reveal the type and length of the field. These versions use a community string for authentication, and it is included in the request and reply message in plain text. While SNMPv2 is backward compatible with v1 the v3 standard proposes a new message format and no longer compatible with the previous versions. Also the SNMPv3 defines additional security features including reliable authentication and encryption of messages, yet the previous versions are still more common because of their simplicity and compatibility. All of the SNMP versions operate over UDP and use the port 161 for communication and the port 162 for traps.

The SNMP standard defines five core message types GetRequest, GetNextRequest, SetRequest, GetResponse and Trap. These are called Protocol Data Units (PDUs). The GetRequests are sent by the manager to the Agent and they query a specific OID, the Agent sends a GetResponse that contains the requested variable if it is present in the Agent's MIB, else the GetResponse's error field is set to NoSuchName error. The SetRequest contains the OID and the new value of the variable, if the chosen variable is read-only the SNMP Agent may send a ReadOnly error, otherwise no reply is sent back. The GetNextRequest operates the same way as the GetRequest but the response contains the next OID and value from the hierarchy if there is one, else the Agent replies with a NoSuchName error.

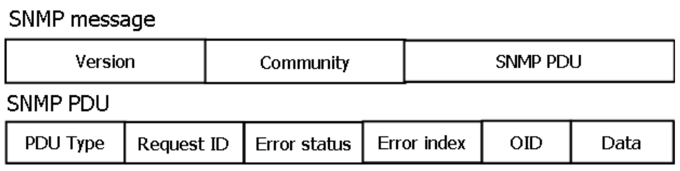


Figure 5: SNMP message structure

All these messages share five common fields, the SNMP version field, the community string field, the sequence field, which is used to link the GetResponse to the appropriate Get or GetNext request, an OID field and an error field. The error field should only be set other than zero in GetResponses. The most common error messages are the value/response is too big (tooBig), there is no such object (NoSuchName), bad value for the SNMP object (BadValue), the object is read only (ReadOnly) and general SNMP error (GenErr). There is also an error index field that can be used to further specify the error that occurred. The SNMPv2 and v3 introduce new message and error types, the discussion of these is not in the scope of this paper. The Figure 5 show the typical format of an SNMP message.

For those who are interested, William Stallings' book, the "SNMP, SNMPV2, Snmpv3, and RMON 1 and 2" [48] provides further information on the subject. The following RFCs are used to define the SNMP standard and the data structures that it uses:

- SNMPv1 RFC 1157[24]: defines the SNMPv1 protocol.
- SMIv1 RFCs:
  - RFC 1155[42]: Structure and Identification of Management Information for TCP/IP-based internets
  - RFC 1212[41]: Concise MIB Definitions
  - RFC 1215[40]: Convention for Defining Traps for use with the SNMP
- MIB-II RFCs:
  - RFC 1213[30]: Management Information Base for Network Management of TCP/IP-based internets: MIB-II
  - RFC 2863[28]: The Interfaces Group MIB (IF)
  - RFC 3418[32]: Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)
  - RFC 4001[25]: Textual Conventions for Internet Network Addresses
  - RFC 4022[44]: Textual Conventions for Internet Network Addresses (TCP)

- RFC 4113[20]: Management Information Base for the User Datagram Protocol (UDP)
- RFC 4293[45]: Management Information Base for the Internet Protocol (IP)

3.2.2 Observation of PLC

The SNMP service is commonly installed on intermediary and end network devices, because it provides a simple and easy way to monitor and manage the device. SNMP is often implemented on different PLCs as well, for the same reason. The SIEMENS S7 support page[19] provides a list of ports that may be open on the S7 family PLCs. The port 161 being listed there suggests that the observed device might run an SNMP Agent. An initial Nessus[16] scan of the PLC proved that it has SNMP service running. It also provided two community string that allows access to the stored records. These strings were *public* and *private* which are the two most common default community strings for SNMP.

The next step in uncovering the PLC's SNMP Agent was to query all the records that the Agent offers and identify which MIBs are installed on it. To achieve this the snmpwalk program was used which is part of the Net-Snmp[23] suit. The Net-Snmp suit contains multiple programs that can generate Set, Get or GetNext requests, snmpd is also part of the bundle which is the most commonly used SNMP Agent. The snmpwalk keeps sending GetNextRequests, always with the OID that was returned by the previous GetResponse, until a NoSuchName error is encountered, which means that the end of the MIB is reached. The snmpwalk takes an OID or a partial OID as an argument which identifies a node in hierarchy and returns all the variables that are under the chosen node in the tree structure. If the snmpwalk's argument is the root of the hierarchy than all the records are returned that the Agent holds. The PLC's Agent was parsed this way both with *public* and *private* community strings and the results were identical. All the three possible roots of the hierarchy were scanned, but only the 1.3.6.1.2 node held data, which contains ISO defined internet related management information.

The output of the snmpwalk was further analysed and the MIBs used by the PLC were determined. The first seven records that the Agent provides contains general information about the PLC such as system description and uptime.

```
SNMPv2-MIB::sysDescr.0 = STRING: Siemens, SIMATIC S7, IM151-8,
 6ES7 151-8AB01-0AB0 , HW: 3, FW: V3.2.6, S C-C1TR95142012
SNMPv2-MIB::sysObjectID.0 = OID: SNMPv2-SMI::zeroDotZero
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks:
(285968480) 33 days, 2:21:24.80
```

```
SNMPv2-MIB::sysServices.0 = INTEGER: 78
```

The next 88 variables are from the IF-MIB. The PLC has four network interfaces, the first one is a serial port that uses PROFIBUS protocol to control the devices attached to it in a real production environment, the second interface is a Fast Ethernet port that is used to manage the PLC, it should be connected to the management network. The third and fourth interface can be used to form a ring topology of the PLCs. The IF-MIB provides information about the connections state and speed, the interfaces' state and physical address, the amount of different types of data received and sent, the number of discarded packets and the length of the outgoing packet queue.

```
IF-MIB::ifDescr.1 = STRING: Siemens SIMATIC S7, internal, Rack 0,
Slot 2
IF-MIB::ifDescr.2 = STRING: Siemens SIMATIC S7, Ethernet Port 1,
link, 100 Mbit, full duplex, autonegotiation
IF-MIB::ifDescr.3 = STRING: Siemens SIMATIC S7, Ethernet Port 2,-
IF-MIB::ifDescr.4 = STRING: Siemens SIMATIC S7, Ethernet Port 3,-
```

The following 68 records are from the IP-MIB extended with the RFC1213-MIB. The IP-MIB consists of information about and obtained by the IP stack, such as the number of received and sent IP packets, fragments, header errors, discards, etc., the IP address of the interface, the next hop and the default gateway. It also provides the number of different types of ICMP messages sent and received. The RFC1213-MIB keeps records about routing (e.g the routing protocol, route type, the age of the route being used to communicate with the manager, routing mask).

```
IP-MIB::ipForwarding.0 = INTEGER: notForwarding(2)
IP-MIB::ipDefaultTTL.0 = INTEGER: 30
IP-MIB::ipInReceives.0 = Counter32: 329598
IP-MIB::ipInHdrErrors.0 = Counter32: 2
IP-MIB::ipInAddrErrors.0 = Counter32: 125821
RFC1213-MIB::ipRouteNextHop.10.105.0.0 = IpAddress:
255.255.255.255
RFC1213-MIB::ipRouteType.10.105.0.0 = INTEGER: direct(3)
RFC1213-MIB::ipRouteProto.10.105.0.0 = INTEGER: local(2)
```

The IP-MIB is followed by the TCP-MIB and UDP-MIB which contains 29 and 10 records respectively. Both of these hold information about the open sockets, their port numbers, the sent and received packets and the TCP-MIB keeps track of the active and passive TCP connections.

```
TCP-MIB::tcpCurrEstab.0 = Gauge32: 0
TCP-MIB::tcpInSegs.0 = Counter32: 1636
```

```
TCP-MIB::tcpOutSegs.0 = Counter32: 1828
TCP-MIB::tcpRetransSegs.0 = Counter32: 0
TCP-MIB::tcpConnState.0.0.0.0.443 = INTEGER: listen(2)
TCP-MIB::tcpConnState.0.0.0.0.80 = INTEGER: listen(2)
TCP-MIB::tcpConnState.0.0.0.0.102 = INTEGER: listen(2)
```

The last 28 variables come from the SNMP-MIB. These records provide information about the SNMP protocol itself, including the number of received request and errors, the number of parse errors, and the amount of different type of error messages sent out.

```
SNMPv2-MIB::snmpInPkts.0 = Counter32: 793
SNMPv2-MIB::snmpOutPkts.0 = Counter32: 791
SNMPv2-MIB::snmpInBadVersions.0 = Counter32: 0
SNMPv2-MIB::snmpInBadCommunityNames.0 = Counter32: 0
SNMPv2-MIB::snmpInBadCommunityUses.0 = Counter32: 0
SNMPv2-MIB::snmpInASNParseErrs.0 = Counter32: 0
SNMPv2-MIB::snmpInTooBigs.0 = Counter32: 0
SNMPv2-MIB::snmpInNoSuchNames.0 = Counter32: 0
SNMPv2-MIB::snmpInBadValues.0 = Counter32: 0
SNMPv2-MIB::snmpInReadOnlys.0 = Counter32: 0
SNMPv2-MIB::snmpInGenErrs.0 = Counter32: 0
SNMPv2-MIB::snmpInTotalReqVars.0 = Counter32: 802
SNMPv2-MIB::snmpInTotalSetVars.0 = Counter32: 0
SNMPv2-MIB::snmpInGetRequests.0 = Counter32: 1
SNMPv2-MIB::snmpInGetNexts.0 = Counter32: 806
SNMPv2-MIB::snmpInSetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpInGetResponses.0 = Counter32: 0
SNMPv2-MIB::snmpInTraps.0 = Counter32: 0
SNMPv2-MIB::snmpOutTooBigs.0 = Counter32: 0
SNMPv2-MIB::snmpOutNoSuchNames.0 = Counter32: 2
SNMPv2-MIB::snmpOutBadValues.0 = Counter32: 0
SNMPv2-MIB::snmpOutGenErrs.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetNexts.0 = Counter32: 0
SNMPv2-MIB::snmpOutSetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetResponses.0 = Counter32: 815
SNMPv2-MIB::snmpOutTraps.0 = Counter32: 0
SNMPv2-MIB::snmpEnableAuthenTraps.0 = INTEGER: disabled(2)
End of MIB
```

These records were proven invaluable in the process of further uncovering the behaviour of the SNMP Agent of the PLC. This step was necessary because the SNMP standard doesn't define these behaviours and different vendors' application may operate differently. To further analyse the PLC, a python script was created, that could forge valid SNMP packets with adjustable version, community string, SNMP packet type, error status, error index, OID and data fields. The packet capture of the replied messages and the PLC's SNMP-MIB provided feedback about the effects of the crafted messages. This way, the following observations were made; the Agent stops parsing the message if the SNMP version field is not set to SNMPv1, or the community name is not public or private, or a parsing error occurres such as wrong message length or wrong ASN.1 variable type, than the appropriate records are incremented. The SNMP Agent silently ignores any type of message other than Set, Get or GetNext request. This means that the Agent doesn't count the incoming forged GetResponses or Traps. Although, the inbound SetRequests increment the snmpInSetRequests variable, they neither modify the selected record as expected nor cause a ReadOnly error message. They have no effect on any variable. It was also discovered, that the Get and GetNext requests error fields are not parsed, which means the different inbound error records always remain zero. Finally, the PLC was bombarded with various malformed packets, that had wrong length, or invalid variable types, or extremely long OID, data, community name fields, or they used wrong encoding. All these packets triggered parse error on the PLC, however no error message was sent by the Agent. After all these tests, it was assumed that the Agent on the PLC only replies to Get and GetNext requests and it is restricted to generate GetResponses and NoSuchName messages only.

After all the records held by the PLC were carefully mapped (the full-length output of the snmpwalk can be found in Appendix A) and the exact behaviour of the Agent was noted, the next task was to create an Agent that appears to be identical to the one running on the PLC.

3.2.3 Simulating the SNMP service

After the thorough exploration of the PLC came the implementation phase, before the actual realization multiple approaches were considered. The first idea was to utilize an already existing open source SNMP Agent and modify it to our needs. The Net-Snmp suit's snmpd application was chosen because it is the most commonly used open source Agent and most importantly it provides an interface to modify the way it accesses the requested data through so called sub-agents. These sub-agents can be used to redefine how the information held by a record is queried, which was mandatory for us, since some of the data a real Agent would provide needed to be altered in order to simulate the PLC. After some experimenting this approach was dropped, because in order to make snmpd behave the same way as the PLC's Agent we would have to make significant changes in it, and the creation of sub agent for each MIB required extensive work. Also the logging functionality of snmpd was not sufficient for our needs.

The experiment with snmpd showed that the amount of work needed to patch up an existing Agent is close to be the same as writing a new Agent requires, meanwhile creating a new Agent provides obvious benefits, such as being able to customise it and implement extended logging functionality. The conclusion was to create our own Agent. The first step was to chose a programming language. The python became our choice, because it has an excessive amount of libraries available, it offers an easy way to control and use sockets and data processing and manipulation is very simple and efficient in it, which was a main concern for us. Before writing an Agent from scratch, the already existing python SNMP library, PySNMP was examined. The PySNMP is a cross-platform, pure python SNMP engine that capable to act as manager, agent or proxy, it is a complex library that consists more than 15000 lines of code. Because of the complexity and size of the PySNMP it was ruled out. This decision was reinforced by the fact that the only feature of the library that we could utilise was the message handling and processing. Thus, the decision was made to write our own python SNMP Agent implementation. The rest of this section describes the functions and operation of the created python script in details.

The realised Agent, just like the original, listens on the UDP port 161, and accepts SNMP requests and replies to them. Instead of using real MIBs it parses an XML file that contains the list of records that are present on the real PLC. All these records have an OID and a type attribute. They either contain the static data (e.g. the system descriptor, or interface description) that they represent or they contain a special mark and string that tells the interpreter how the dynamic data (e.g. the system uptime, or the number of received IP packets) should be created or retrieved.

```
<oid id="1.3.6.1.2.1.1.1.0" type="string">Siemens, SIMATIC S7,
IM151-8, 6ES7 151-8AB01-0AB0 , HW: 3, FW: V3.2.6,
S C-C1TR95142012</oid>
<oid id="1.3.6.1.2.1.2.2.1.2.1" type="string">Siemens SIMATIC S7,
internal, Rack 0, Slot 2</oid><!--ifdescr-->
<oid id="1.3.6.1.2.1.1.3.0" type="timetick">!sysUpTime</oid>
<oid id="1.3.6.1.2.1.4.2.0" type="integer">30</oid><!--ipDefaultTTL-->
<oid id="1.3.6.1.2.1.4.3.0" type="counter">!ipInReceives</oid>
```

The parser reads these variables and organises them into a tree structure, where each segment of the OID represents a node on the appropriate level of the tree and the leafs hold the data (or the special string) and the meta data of the record. This structure is implemented in python with nested lists. When a Get or GetNext request arrives, the script searches this hierarchy with the queried OID, if there is a match a GetResponse is sent back with the retrieved data, else a NoSuchName error is generated. The following code snippet is the recursive method that is used when a GetRequest arrives.

```
#finds an oid in the database
def find_match(root, oid, index):
for elements in root:
    if elements['indexVal']==oid[index]:
        if elements['isLeaf']==True:
            return OID_OK, elements
        elif index<len(oid)-1:
            return find_match(elements['childs'],
oid, int(index+1))
return OID_NOT_PRESENT, None
```

The program accepts incoming datagrams in an infinite loop and tries to parse them as SNMP packets. If an error occurres during this process the snmpASNParseErr record is incremented, and the error and the current time is logged. Otherwise, the parser extracts the version, community name, request type, request ID and OID values and with the current time and the sender's IP address these values are logged as well. Also, the script keeps track of the SNMP related records in the database, so after each request it increments the ones that need to be changed.

After, a Get or GetNext request is successfully parsed and the version and community string are checked and if the requested OID is present in the hierarchy, the next step is to acquire the data that the OID represents. If the data is static it is contained in the database so no further steps are needed, otherwise it needs to be read or generated. One of the dynamically generated records is system uptime. When it is queried the time in seconds since the script has been started is returned. Other example of generated data is the traffic of the serial interface. The virtual machine that simulates the PLC has no real serial bus so these values must be created. It is safe to do so, because a possible attacker has no direct access to this interface on a real PLC, hence the attacker has no way to verify the actual number. The returned records' values (e.g. serial in/out packets, in/out unicast packets, etc.) are different pseudo randomly incremented counters.

There are variables that an attacker could directly or indirectly alter. Such variables are for example the number of received packets on the connected interface or the number of received ICMP echo requests. The value of these records can not be simply generated because an attacker can try to modify these and check if the returned numbers vary accordingly. To avoid the detection of the honeypot, these values are read from the /proc file system which contains information gathered by the OS on Unix systems. The script receives the name of the interface and this name is used when reading device related information from the /proc/net/dev file. The IP, TCP, UDP and ICMP related data is acquired form the /proc/net/snmp. The purpose of this file is to provide information about these protocols for different SNMP Agents. The TCP current establishments value is read from the

/proc/net/sockstat file. As it was mentioned before, the script keeps track of the SNMP related events. It uses the data gathered this way to serve requests for SNMP records. The code snippet shows how the ipInReceives record's value is accessed.

```
#IP Data
elif value[0:2]=="ip":
    data=open("/proc/net/snmp", "r")
    lines=data.readlines()
    words=[]
    i=0
    for i,ln in enumerate(lines):
        temp=ln.strip().split(" ")
        if temp[0]=="Ip:":
            words=lines[i+1].strip().split(" ")
            words= filter(lambda a: a != '', words)
            break
    if value=="ipInReceives":
        ret=words[3]
```

After, the data is obtained, the script creates a valid SNMP GetResponse and replies to the manager. The message format of the GetResponse is the same as the GetRequest's, but instead of a Null field at the end, the response has a valid data field, and the different length fields are also modified to conform to the ASN.1 standard. When GetRequest arrives with an invalid OID, a NoSuchName error is sent back. This message is generated the same way as the GetResponse, the only difference is that it contains no data and has its error field set. After the appropriate answer is sent back, the SNMP communication is over the Agent has no further tasks to do.

3.2.4 SNMP evaluation

The implemented SNMP Agent was tested with snmpwalk. The entire output of the command can be found in Appendix B. The result of the test corresponded to the expectations. Later, the Agent was tested against different Get and GetNext requests, the response format was always identical to the original PLC's responses. Finally, a variety of malformed SNMP packets were sent to the Agent and to the PLC, and it was observed that the changed records in the SNMP-MIBs were the same and the value of these records were equal.

It is important to note, that there is a known limitation of the SNMP Agent. If a specific group of records are queried from the local network, the returned data is not valid. The following OIDs are affected:

```
IP-MIB::ipNetToMediaIfIndex.1.10.105.1.216 = INTEGER: 1
IP-MIB::ipNetToMediaPhysAddress.1.10.105.1.216 = STRING:
```

```
0:c:29:76:44:ce
IP-MIB::ipNetToMediaNetAddress.1.10.105.1.216 = IpAddress:
10.105.1.216
IP-MIB::ipNetToMediaType.1.10.105.1.216 = INTEGER: dynamic(3)
```

These records hold information about the address of the next hop toward the manager that requested the OID. If the manager is located on a remote network, the default gateway's address is returned, however if the manager and the Agent are on the same LAN, the response contains the manager's address. The main problem is that the OID used to query this record contains the IP address. In a real Agent these records are dynamically created (so called read-create type), but the underlying data structure of the simulator does not allow this. In our SNMP Agent the data hierarchy is initialised from an XML file, when the script is started, later the dynamic modification of this structure is not possible. In the future, this hierarchy needs to be redesigned to support the read-create functionality. As of now, if the honeypot is configured with a public IP address, it is safe to assume that the attacker is not on the local network, thus the honeypot can successfully fulfil its task.

3.3 Communication with Siemens SIMATIC STEP 7

3.3.1 Siemens SIMATIC STEP 7

Siemens SIMATIC STEP7[10] is an engineering software for configuring and programming Siemens type controllers. We can setup and program several automation systems, for example SIMATIC HMI panels, or Siemens PLCs. STEP 7 contains various components which extend the functions of the basic software, for example S7 Graph for describing procedures in a quick and easily understandable way, or the S7 PLCSIM, for simulating programmable logic controllers. However, we are going to discuss only the basic software, because it is enough for our project.

When we want to configure or program our PLC, we have to "build" it in STEP 7 first. STEP 7 contains a hardware configuration module, where we can do this easily. STEP 7 supports many Siemens PLC modules, for example there are several CPUs and I/O modules, with different version numbers, so we can create our PLC with ease based on the real one. After that, we can start to configure the device. We can setup general parameters and basic functions like name of the PLC, password protection for access, or enabling the web server. There is also a configuration panel for setting up the network, so we can set the IP address of the modules (if it is connected through Industrial Ethernet), and we can define the connections between the modules too. STEP 7 supports three programming languages for creating the software of our automation system: Ladder, Function Block Diagram and Instruction List. This means that we can also develop the proper software for our PLC with the help of STEP 7.

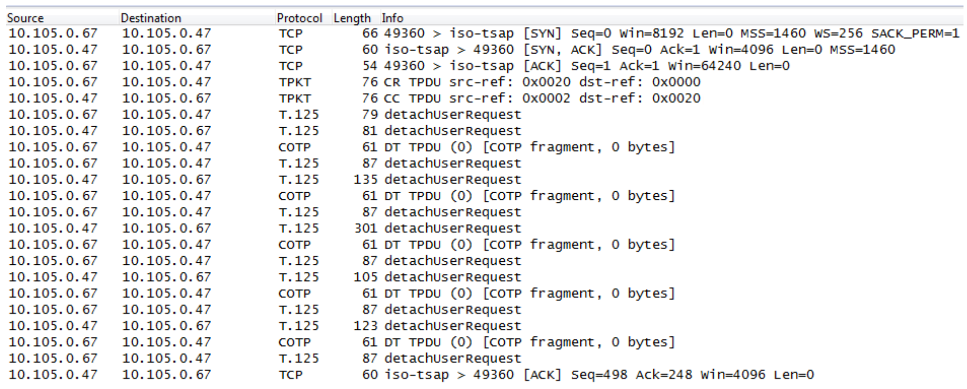
After configuring the system, we can download the software to the PLC. This is the most important part of our project. The programming can be done through Industrial Ethernet, and STEP 7 is capable to create a remote connection to the device via Internet. Through this connection we can program the PLC from an external network. Thus, if the PLC is accessible from the Internet, a potential attacker could see that it is accepting these types of connections. It is important to simulate this behaviour of the PLC, so our honeypot could be even more realistic.

The communication is going through TCP port 102 of the PLC. The traits of connections on this port are defined in the RFC 1006[43]. As we can read in the introduction of the RFC, this service helps the "porting" of ISO-standardized applications to TCP/IP environment. This is done by the Transport Service Access Point which takes advantage of the layered structure of both TCP/IP and the ISO protocol. This Access Point appears identical to the services and interfaces offered by the ISO-TSAP, but it is implemented on top of the TCP/IP protocol. This way, higher level ISO layers can operate fully without knowing that they are running over a TCP/IP network. The ISO protocol exchanges data between peers in transport protocol data units (TPDUs). These units are encapsulated into discrete TCP packets, TPKTs. TCP manages a continuous stream of data, but the ISO protocol requires single discrete data objects. This conversion is done by the TPKTs: they contain both the information required by TCP to manage the stream as well as the TPDU data segments. There are several service primitives which are required by the ISO protocols in order to work properly. These are primitives for establishing, maintaining and closing connections or sending data. As we saw, these primitives travel in specific TPKTs. In our project, the initiation of the connection is always one sided: the STEP 7 requests the connection from the PLC. Because of this behaviour, only three primitive is important for our project, these are the connection request (CR), connection confirm (CC), and data transfer (DT). We are going to take a closer look on these objects in the following subsections.

3.3.2 Observation of the PLC

After scanning the PLC for open ports, we found that TCP port 102 is open. This means that we can perform the configuration and programming of the PLC with the help of STEP 7. Since the port is open, and it is common that these systems can be accessed through a remote connection from the Internet via STEP 7, it was obvious that we should simulate this service in some form. We started to examine STEP 7 and this connection, and we found out, that there is a setting in the PLC which protects it from unauthorized access. This means that we can not download the program to the PLC until we entered the correct password. We decided that we are going to simulate this behaviour on the PLC, but without a correct password. So whenever someone tries to access the PLC, the response always will be that the entered password is incorrect. On one hand, this is a fairly simple solution, but it is also believable: a system which has such a great responsibility in controlling facilities or power plants should not be accessible to anyone. On the other hand, if we would like to simulate the programmability, the response to various inputs could be problematic.

We examined the concrete communication between the PLC and STEP 7 with the help of Wireshark. We can see a part of this communication on Figure 6. As specified in the ISO standards, the connection between the communicating halves should be established with a two-way handshake. Because of this, at first STEP 7 sends a connection request TPDU to the PLC. This packet contains a source identifier, so the other half can reference to the source through this. The PLC responds with a connection confirmation,



```
Source        Destination      Protocol  Length  Info
10.105.0.67   10.105.0.47      TCP          66 49360 > iso-tsap [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
10.105.0.47   10.105.0.67      TCP          60 iso-tsap > 49360 [SYN, ACK] Seq=0 Ack=1 Win=4096 Len=0 MSS=1460
10.105.0.67   10.105.0.47      TCP          54 49360 > iso-tsap [ACK] Seq=1 Ack=1 Win=64240 Len=0
10.105.0.67   10.105.0.47      TPKT         76 CR TPDU src-ref: 0x0020 dst-ref: 0x0000
10.105.0.47   10.105.0.67      TPKT         76 CC TPDU src-ref: 0x0002 dst-ref: 0x0020
10.105.0.67   10.105.0.47      T.125        79 detachUserRequest
10.105.0.47   10.105.0.67      T.125        81 detachUserRequest
10.105.0.67   10.105.0.47      COTP         61 DT TPDU (0) [COTP fragment, 0 bytes]
10.105.0.67   10.105.0.47      T.125        87 detachUserRequest
10.105.0.47   10.105.0.67      T.125       135 detachUserRequest
10.105.0.67   10.105.0.47      COTP         61 DT TPDU (0) [COTP fragment, 0 bytes]
10.105.0.67   10.105.0.47      T.125        87 detachUserRequest
10.105.0.47   10.105.0.67      T.125       301 detachUserRequest
10.105.0.67   10.105.0.47      COTP         61 DT TPDU (0) [COTP fragment, 0 bytes]
10.105.0.67   10.105.0.47      T.125        87 detachUserRequest
10.105.0.47   10.105.0.67      T.125       105 detachUserRequest
10.105.0.67   10.105.0.47      COTP         61 DT TPDU (0) [COTP fragment, 0 bytes]
10.105.0.67   10.105.0.47      T.125        87 detachUserRequest
10.105.0.47   10.105.0.67      T.125       123 detachUserRequest
10.105.0.67   10.105.0.47      COTP         61 DT TPDU (0) [COTP fragment, 0 bytes]
10.105.0.67   10.105.0.47      T.125        87 detachUserRequest
10.105.0.47   10.105.0.67      TCP          60 iso-tsap > 49360 [ACK] Seq=498 Ack=248 Win=4096 Len=0
```

Figure 6: Wireshark capture of the communication with the real PLC

it also sends back the STEP 7's source reference for verification. After this, the communication is done purely by DT TPDUs, so only data transfer is done between the PLC and the STEP 7. The process of programming the PLC is done in two phases. In the first phase, STEP 7 queries the PLC for its parameters. These are the following: station name, module name, module type, order number and serial number. We can see by examining the contents of the TPDUs that all of these attributes are queried by the STEP 7, but actually only three of them are used. STEP 7 prints the module type, module name and station name for the user, so if there is more than one module on the same IP address, the user can select to which module the STEP 7 should upload the program. It is important to note that even if there is only one module on the specified address, this phase is always executed. After selecting the module, the second phase begins. Since we are building a new connection, connection request and connection cofnirmation TPDUs are sent again. If the PLC is not protected with a password, the STEP 7 performs the programming in this phase. Otherwise, the PLC prompts for a password, and if the user enters a wrong password, the PLC prompts again until a correct password is entered or the connection is closed by the user.

As we can see, the communication is very simple. Both phases are relatively short, and there are few changing parameters from the PLC's side. These variables are the source identifier from the connection request TPDU, and a sequence number for every entered incorrect password. We have to provide also the parameters of the PLC for the first phase, but those are constant values, so it should be fairly easy to insert them into the data units. In the tests, we used our PLC's parameters, but by changing these parameters, it is easy to mimic a different PLC. Thanks to the Wireshark capture, the structure and order of the packets are given, and we are able to realize the communication based on this information.

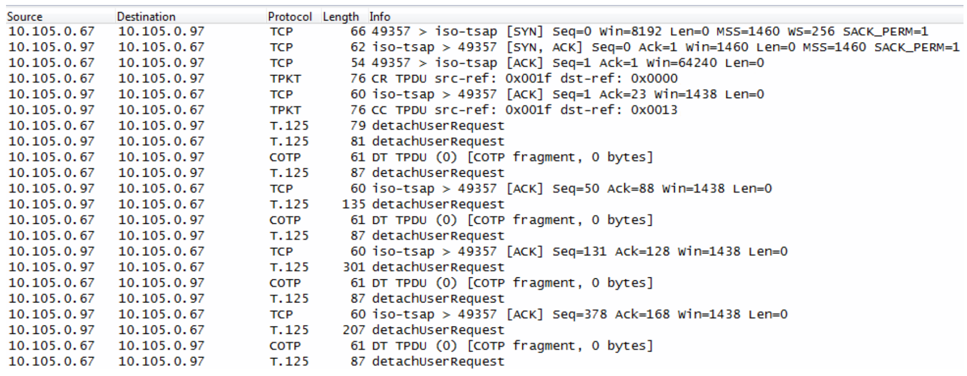
3.3.3 Implementation

After examining the communication between the STEP 7 and a real PLC, we started to implement the simulation of this service. We decided that the service should be a simple script written in Python, because every tool is available in Python to simulate such a communication. Basically the only thing we need is the socket module in Python, since we can easily create a TCP stream with this module, and we only have to read and write the correct bytes in order to realize the communication.

At the beginning, the script starts to listen on TCP port 102. As we saw in the observation phase, the communication begins with a two-way handshake, a connection request TPDU followed by a connection confirmation TPDU. We have to receive the connection request, which contains the source reference, which identifies the STEP 7 as the initiator of this communication. We have to include this information in the connection confirmation packet as the destination reference. Reading the source identifier from the first packet is easy, since these TPDUs have a specified format, which of course the TCP still keeps for the compatibility with ISO systems.

As we saw in the observation phase, the first phase is about gathering information. STEP 7 queries the PLC for specific parameters. We have to include these parameters in our packets, so the attacker could believe that this is a real PLC. As we stated, STEP 7 queries several parameters from the PLC. These parameters can be found out from the data sheet of the device, or usually they are also shown on the website of the PLC. As we saw, only three of the parameters are presented to the user of STEP 7, the module name, the module type and station name, so we could think that the other parameters does not matter, they could be any dummy values, but this is not exactly true. It is important to note that the module type is determined by the serial number, not the module type parameter sent in the packets. Thus, the serial number must be valid so the STEP 7 can identify the module type of the device. By analyzing the packets, we can locate the proper position of the parameters in the TPDUs. The first phase contains only these variable attributes, so we can easily create and send the packets based on the Wireshark capture.

When the first phase is completed, the user can select the module he wants to program, and the second phase begins. Since this is a new connection, we have to establish it with connection request and connection confirm messages. This is done the same way as in the first phase. However, there is a slight difference between the connection request TPDUs. Namely, these



```
Source        Destination     Protocol  Length Info
10.105.0.67   10.105.0.97     TCP          66 49357 > iso-tsap [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
10.105.0.97   10.105.0.67     TCP          62 iso-tsap > 49357 [SYN, ACK] Seq=0 Ack=1 Win=1460 Len=0 MSS=1460 SACK_PERM=1
10.105.0.67   10.105.0.97     TCP          54 49357 > iso-tsap [ACK] Seq=1 Ack=1 Win=64240 Len=0
10.105.0.67   10.105.0.97     TPKT         76 CR TPDU src-ref: 0x001f dst-ref: 0x0000
10.105.0.97   10.105.0.67     TCP          60 iso-tsap > 49357 [ACK] Seq=1 Ack=23 Win=1438 Len=0
10.105.0.97   10.105.0.67     TPKT         76 CC TPDU src-ref: 0x001f dst-ref: 0x0013
10.105.0.67   10.105.0.97     T.125        79 detachUserRequest
10.105.0.97   10.105.0.67     T.125        81 detachUserRequest
10.105.0.67   10.105.0.97     COTP         61 DT TPDU (0) [COTP fragment, 0 bytes]
10.105.0.67   10.105.0.97     T.125        87 detachUserRequest
10.105.0.97   10.105.0.67     TCP          60 iso-tsap > 49357 [ACK] Seq=50 Ack=88 Win=1438 Len=0
10.105.0.97   10.105.0.67     T.125       135 detachUserRequest
10.105.0.67   10.105.0.97     COTP         61 DT TPDU (0) [COTP fragment, 0 bytes]
10.105.0.67   10.105.0.97     T.125        87 detachUserRequest
10.105.0.97   10.105.0.67     TCP          60 iso-tsap > 49357 [ACK] Seq=131 Ack=128 Win=1438 Len=0
10.105.0.97   10.105.0.67     T.125       301 detachUserRequest
10.105.0.67   10.105.0.97     COTP         61 DT TPDU (0) [COTP fragment, 0 bytes]
10.105.0.67   10.105.0.97     T.125        87 detachUserRequest
10.105.0.97   10.105.0.67     TCP          60 iso-tsap > 49357 [ACK] Seq=378 Ack=168 Win=1438 Len=0
10.105.0.97   10.105.0.67     T.125       207 detachUserRequest
10.105.0.67   10.105.0.97     COTP         61 DT TPDU (0) [COTP fragment, 0 bytes]
10.105.0.67   10.105.0.97     T.125        87 detachUserRequest
```

Figure 7: Wireshark capture of the communication with the honeypot

requests contain that it is a connection initiation for the first or the second phase. This way, we can identify which phase should begin after the connection is established, so we can avoid code duplication, and more importantly, if the user cancels the module selection at the end of the first phase and begins a new connection, we can detect that it is not a download request (which it should be after the first phase), and we can send the attributes of the PLC again. After the connection is established, we query for the password of the PLC. But it does not matter what the user sends, we always send back the incorrect password message. This packet contains a variable number which increases by one on every attempt, so we have to form the message this way. Again, using the packet contents from the Wireshark capture, this phase is not hard to implement.

We tested the simulation with STEP 7. We can see a part of this communication on Figure 7. When we are trying to connect to the script, the first phase begins, and STEP 7 prints the correct attributes. After selecting the module, the script prompts for a password. There is no correct password, so when the user enters one, the script prompts for a password again and again, until the user closes the connection by clicking Cancel. If the user decides to not select a module, then tries to connect again, the script sends the attributes in the first phase properly. Analyzing the network traffic between the script and STEP 7 also proves that the simulation is working properly. We also added a small logging function to the script, which logs every incoming connection with the exact time and IP address into a log file.

4 Testing

We performed the needed tests for the single services, and presented the results in the related sections. After the integration, it was important to test the system as a whole. We tested first the operation of the services in the integrated environment. After making sure that the communication with the modules of the honeypot is proper, we checked the difference between the honeypot and the real PLC.

We asked an independent team which made success in several "capture the flag" competitions to examine the devices for this purpose. They were able to successfully distinguish the honeypot from the real PLC. Although we tried to mimic the PLCs networking stack, due to operating system limitations, we were not able to emulate the full TCP/IP stack. Thus, based on the `nmap O` operating system scan, one can distinguish between the two devices. In the nmaps operating system guess report there is an entry for Linux when scanning the honeypot (with several other guesses), but Linux is not guessed when the PLC is scanned. There is also a difference in the MAC addresses. Nmap is able to guess the origin of the ethernet card based on the prefix of the MAC address. So in the PLCs scan this value is "Siemens", but in the honeypots scan it is "VMware" (we are using a VMware ESXi based virtual machine).

We fixed the problem with the MAC address, since on Linux it is easy to change the MAC address to one originated from Siemens. However, fooling nmap with the proper TCP/IP parameters is harder. One solution can be writing a TCP proxy, which re-frames the outgoing packets, so it would seem it is coming from a real PLC. This is one of our future goals. Otherwise the team could not spot any differences between the two systems.

5 Summary

The aim of this study was to create a high interaction honeypot, that appears to be a Siemens PLC form an attackers point of view. It needs to be able to log all the action an attacker takes, while trying to exploit the PLC. So that later by analysing these log files new targeted attacks can be uncovered, possibly before they reach the real equipments. In order to achieve this all the existing PLC honeypots were examined, and a real PLC was thoroughly audited.

After the exploration of the device, all the discovered services (HTTP, HTTPS, ISOTSAP and SNMP) were further inspected. Than the simulators of these were implemented and integrated into a Linux based virtual machine. The resulting VM is the honeypot. Although, currently the honeypot has a few limitations which needs to be addressed in the future, still it preforms its task better than any of its predecessors.

The most important current issue is the incorrect TCP window size, which cannot be set to the exact required value because of the Linux kernels limitations. In the future a kernel patch or TCP proxy (that re-frames all the outgoing TCP PDUs and sends them with raw sockets) needs to be written to overcome this problem. The other known issue is related to the SNMP routing records (it is expressed in details in the 3.2.4 SNMP evaluation section), this is a less significant problem, because it only exists if the attacker is on the same LAN as the honeypot. The honeypot was tested by independent professionals and no other issue was discovered.

Our plans for the close future is to configure the honeypot with a public IP address and gather information. Later, by analysing the logs we will try to identify new threats that target industrial control systems.

Acknowledgement

We would like to acknowledge the help and the provided Siemens PLC device to Óbuda University and evopro corporation.

References

[1] The conpot project. http://www.conpot.org. Last accessed: 2013-08-04.

[2] Crash per webinterface. http://www.sps-forum.de/simatic/52478-s7-1200-crash-per-webinterface.html. Last accessed: 2013-10-16.

[3] Gnu wget. http://www.gnu.org/software/wget/. Last accessed: 2013-10-16.

[4] Independent ftp daemon. http://sourceforge.net/projects/iftpd/. Last accessed: 2013-06-17.

[5] Introducing conpot. http://honeynet.org/node/1047. Last accessed: 2013-08-04.

[6] Miniweb project webpage. http://miniweb.sourceforge.net/. Last accessed: 2013-10-07.

[7] Nginx site. http://wiki.nginx.org. Last accessed: 2013-10-16.

[8] Openssl: The open source toolkit for ssl/tls. http://www.openssl.org. Last accessed: 2013-10-16.

[9] Scada honeynet. http://www.digitalbond.com/tools/scada-honeynet/. Last accessed: 2013-06-17.

[10] Simatic step 7 engineering software - software for simatic controllers - siemens. http://www.automation.siemens.com/mcms/simatic-controller-software/en/step7/Pages/Default.aspx. Last accessed: 2013-10-18.

[11] Trustworthy internet movement - ssl pulse. https://www.trustworthyinternet.org/ssl-pulse/. Last accessed: 2013-10-10.

[12] Valgrind home page. http://valgrind.org. Last accessed: 2013-10-16.

[13] Wind river vxworks rtos. http://www.windriver.com/products/vxworks/. Last accessed: 2013-06-17.

[14] Honeywall project site. http://www.honeyd.org/honeywall/, 2009. Last accessed: 2013-06-17.

[15] Backtrack linux - penetration testing distribution. http://www.backtrack-linux.org/, 2013. Last accessed: 2013-10-23.

[16] Nessus vulnerability scanner. http://www.tenable.com/products/nessus, 2013. Last accessed: 2013-10-10.

[17] Nmap - free security scanner for network exploration & security audits. http://nmap.org/, 2013. Last accessed: 2013-10-23.

[18] Profinet. http://www.profibus.com/technology/profinet/, 2013. Last accessed: 2013-10-11.

[19] Siemens product support. http://support.automation.siemens.com /WW/llisapi.dll?func=cslib.csinfo&lang=en&objid=8970169&caller=view, 2013". Last accessed: 2013-10-13.

[20] J. Flick B. Fenner. Rfc 4113: Management information base for the user datagram protocol (udp), 2005.

[21] Tim Berners-Lee, Roy Fielding, and H Frystyk. Rfc 1945: Hypertext transfer protocol http/1.0, 1996.

[22] D. Bond. Fizmez web server. http://sourceforge.net/projects/fizmezwebserver/. Last accessed: 2013-06-17.

[23] A. Burger. Net-snmp. http://net-snmp.sourceforge.net/, 2013".

[24] JD Case, MS Fedor, ML Schoffstall, and C Davin. Rfc 1157: Simple network management protocol, 1990.

[25] M. Daniele, B. Haberman, S. Routhier, and J. Schoenwaelder. Rfc 4001: Textual conventions for internet network addresses, 2005.

[26] R Fielding, J Gettys, J Mogul, H Frystyk, L Masinter, P Leach, and T Berners-Lee. Rfc 2616: Hypertext transfer protocol http/1.1, 1999.

[27] K Gorzelak, T Grudziecki, P Jacewicz, P Jaroszewski, Ł Juszczyk, P Kijewski, and A Belasovs. Proactive detection of network security incidents. 2012.

[28] F. Kastenholz K. McCloghrie. Rfc 2863: The interfaces group mib, 2000.

[29] Neal Krawetz. Anti-honeypot technology. *Security & Privacy, IEEE*, 2(1):76–79, 2004.

[30] Keith McCloghrie and M Rose. Rfc 1213: Management information base for network management of tcp/ip-based internets: Mib-ii, 1991.

[31] Provos N. Developments of the honeyd virtual honeypot. http://www.honeyd.org/, 2007. Last accessed: 2013-06-16.

[32] R Presuhn. Rfc 3418: Management information base (mib) for the simple network management protocol (snmp), 2002.

[33] Niels Provos. Honeyd-a virtual honeypot daemon. In *10th DFN-CERT Workshop, Hamburg, Germany*, volume 2, 2003.

[34] Niels Provos and Thorsten Holz. *Virtual honeypots: from botnet tracking to intrusion detection.* Pearson Education, 2007.

[35] ITUT Rec. X. 683 (2002) or iso/iec 8824-4: 2002: Information technologyabstract syntax notation one (asn. 1): Parameterization of asn. 1 specifications (2002).

[36] ITUT Rec. Information technology-open systems interconnection-abstract syntax notation one (asn. 1): Information object specification itu-t rec. x. 682 (1994)— iso/iec 8824-3: 1995, 1994.

[37] ITUT Rec. X. 680 abstract syntax notation one (asn. 1)-specification of basic notation, 1994.

[38] ITUT Rec. X. 681 (2002) or iso/iec 8824-2: 2002: Information technologyabstract syntax notation one (asn. 1): Information object specification (2002), 2002.

[39] ITUT Recommendation. Itu-t recommendation x. 690 asn. 1 encoding rules: Specification of basic encoding rules (ber), canonical encoding rules (cer) and distinguished encoding rules (der), 2008.

[40] M ROSE. Rfc 1215: Convention for defining traps for use with the snmp, 1991.

[41] M Rose and K McCloghrie. Rfc 1212: Concise mib definitions, 1991.

[42] Marshall Rose and Keith McCloghrie. Rfc 1155: Structure and identification of management information for tcp, 1990.

[43] Marshall T Rose and Dwight E Cass. Rfc 1006: Iso transport service on top of the tcp version: 3, 1987.

[44] Ed. S. Routhier. Rfc 4022: Management information base for the transmission control protocol (tcp), 2005.

[45] Ed. S. Routhier. Rfc 4293: Management information base for the internet protocol (ip), 2006.

[46] Christian Seifert, Ian Welch, Peter Komisarczuk, et al. Honeyc-the low-interaction client honeypot. *Proceedings of the 2007 NZCSRCS, Waikato University, Hamilton, New Zealand*, 2007.

[47] Lance Spitzner. Honeytokens: The other honeypot, 2003.

[48] William Stallings. *SNMP,SNMPV2,Snmpv3,and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1998.

[49] Pothamsetty V. and Franz M. Scada honeynet project: Building honeypots for industrial networks. 2005.

[50] Kyle Wilhoit. The SCADA That Didnt Cry Wolf, 2013.

[51] Kyle Wilhoit. Whos Really Attacking Your ICS Equipment?, 2013.

[52] D. Wimberger and J. Charlton. Java modbus library (jamod). http://jamod.sourceforge.net/index.html. Last accessed: 2013-06-17.

A PLC snmpwalk output

```
SNMPv2-MIB::sysDescr.0 = STRING: Siemens, SIMATIC S7, IM151-8,
6ES7 151-8AB01-0AB0 , HW: 3, FW: V3.2.6, S C-C1TR95142012
SNMPv2-MIB::sysObjectID.0 = OID: SNMPv2-SMI::zeroDotZero
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (285968480) 33
days, 2:21:24.80
SNMPv2-MIB::sysContact.0 = STRING:
SNMPv2-MIB::sysName.0 = STRING:
SNMPv2-MIB::sysLocation.0 = STRING:
SNMPv2-MIB::sysServices.0 = INTEGER: 78
IF-MIB::ifNumber.0 = INTEGER: 4
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifIndex.3 = INTEGER: 3
IF-MIB::ifIndex.4 = INTEGER: 4
IF-MIB::ifDescr.1 = STRING: Siemens SIMATIC S7, internal, Rack 0, Slot
2
IF-MIB::ifDescr.2 = STRING: Siemens SIMATIC S7, Ethernet Port 1, link,
100 Mbit, full duplex, autonegotiation
IF-MIB::ifDescr.3 = STRING: Siemens SIMATIC S7, Ethernet Port 2,-
IF-MIB::ifDescr.4 = STRING: Siemens SIMATIC S7, Ethernet Port 3,-
IF-MIB::ifType.1 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.2 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.3 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.4 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifMtu.1 = INTEGER: 1518
IF-MIB::ifMtu.2 = INTEGER: 1518
IF-MIB::ifMtu.3 = INTEGER: 1518
IF-MIB::ifMtu.4 = INTEGER: 1518
IF-MIB::ifSpeed.1 = Gauge32: 100000000
IF-MIB::ifSpeed.2 = Gauge32: 100000000
IF-MIB::ifSpeed.3 = Gauge32: 100000000
IF-MIB::ifSpeed.4 = Gauge32: 100000000
IF-MIB::ifPhysAddress.1 = STRING: 0:1b:1b:1a:d6:e0
IF-MIB::ifPhysAddress.2 = STRING: 0:1b:1b:1a:d6:e1
IF-MIB::ifPhysAddress.3 = STRING: 0:1b:1b:1a:d6:e2
IF-MIB::ifPhysAddress.4 = STRING: 0:1b:1b:1a:d6:e3
IF-MIB::ifAdminStatus.1 = INTEGER: up(1)
IF-MIB::ifAdminStatus.2 = INTEGER: up(1)
IF-MIB::ifAdminStatus.3 = INTEGER: down(2)
IF-MIB::ifAdminStatus.4 = INTEGER: down(2)
```

```
IF-MIB::ifOperStatus.1 = INTEGER: up(1)
IF-MIB::ifOperStatus.2 = INTEGER: up(1)
IF-MIB::ifOperStatus.3 = INTEGER: down(2)
IF-MIB::ifOperStatus.4 = INTEGER: down(2)
IF-MIB::ifLastChange.1 = Timeticks: (650) 0:00:06.50
IF-MIB::ifLastChange.2 = Timeticks: (650) 0:00:06.50
IF-MIB::ifLastChange.3 = Timeticks: (650) 0:00:06.50
IF-MIB::ifLastChange.4 = Timeticks: (650) 0:00:06.50
IF-MIB::ifInOctets.1 = Counter32: 0
IF-MIB::ifInOctets.2 = Counter32: 991100834
IF-MIB::ifInOctets.3 = Counter32: 0
IF-MIB::ifInOctets.4 = Counter32: 0
IF-MIB::ifInUcastPkts.1 = Counter32: 330090
IF-MIB::ifInUcastPkts.2 = Counter32: 5361
IF-MIB::ifInUcastPkts.3 = Counter32: 0
IF-MIB::ifInUcastPkts.4 = Counter32: 0
IF-MIB::ifInNUcastPkts.1 = Counter32: 0
IF-MIB::ifInNUcastPkts.2 = Counter32: 10279825
IF-MIB::ifInNUcastPkts.3 = Counter32: 0
IF-MIB::ifInNUcastPkts.4 = Counter32: 0
IF-MIB::ifInDiscards.1 = Counter32: 0
IF-MIB::ifInDiscards.2 = Counter32: 0
IF-MIB::ifInDiscards.3 = Counter32: 0
IF-MIB::ifInDiscards.4 = Counter32: 0
IF-MIB::ifInErrors.1 = Counter32: 0
IF-MIB::ifInErrors.2 = Counter32: 0
IF-MIB::ifInErrors.3 = Counter32: 0
IF-MIB::ifInErrors.4 = Counter32: 0
IF-MIB::ifInUnknownProtos.1 = Counter32: 0
IF-MIB::ifInUnknownProtos.2 = Counter32: 2170273
IF-MIB::ifInUnknownProtos.3 = Counter32: 0
IF-MIB::ifInUnknownProtos.4 = Counter32: 0
IF-MIB::ifOutOctets.1 = Counter32: 0
IF-MIB::ifOutOctets.2 = Counter32: 225384205
IF-MIB::ifOutOctets.3 = Counter32: 0
IF-MIB::ifOutOctets.4 = Counter32: 0
IF-MIB::ifOutUcastPkts.1 = Counter32: 2957552
IF-MIB::ifOutUcastPkts.2 = Counter32: 2527
IF-MIB::ifOutUcastPkts.3 = Counter32: 0
IF-MIB::ifOutUcastPkts.4 = Counter32: 0
IF-MIB::ifOutNUcastPkts.1 = Counter32: 0
IF-MIB::ifOutNUcastPkts.2 = Counter32: 2955026
IF-MIB::ifOutNUcastPkts.3 = Counter32: 0
IF-MIB::ifOutNUcastPkts.4 = Counter32: 0
```

```
IF-MIB::ifOutDiscards.1 = Counter32: 0
IF-MIB::ifOutDiscards.2 = Counter32: 0
IF-MIB::ifOutDiscards.3 = Counter32: 0
IF-MIB::ifOutDiscards.4 = Counter32: 0
IF-MIB::ifOutErrors.1 = Counter32: 0
IF-MIB::ifOutErrors.2 = Counter32: 0
IF-MIB::ifOutErrors.3 = Counter32: 0
IF-MIB::ifOutErrors.4 = Counter32: 0
IF-MIB::ifOutQLen.1 = Gauge32: 0
IF-MIB::ifOutQLen.2 = Gauge32: 0
IF-MIB::ifOutQLen.3 = Gauge32: 0
IF-MIB::ifOutQLen.4 = Gauge32: 0
IF-MIB::ifSpecific.1 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.2 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.3 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.4 = OID: SNMPv2-SMI::zeroDotZero
IP-MIB::ipForwarding.0 = INTEGER: notForwarding(2)
IP-MIB::ipDefaultTTL.0 = INTEGER: 30
IP-MIB::ipInReceives.0 = Counter32: 329598
IP-MIB::ipInHdrErrors.0 = Counter32: 2
IP-MIB::ipInAddrErrors.0 = Counter32: 125821
IP-MIB::ipForwDatagrams.0 = Counter32: 0
IP-MIB::ipInUnknownProtos.0 = Counter32: 0
IP-MIB::ipInDiscards.0 = Counter32: 0
IP-MIB::ipInDelivers.0 = Counter32: 203781
IP-MIB::ipOutRequests.0 = Counter32: 2535
IP-MIB::ipOutDiscards.0 = Counter32: 0
IP-MIB::ipOutNoRoutes.0 = Counter32: 0
IP-MIB::ipReasmTimeout.0 = INTEGER: 120 seconds
IP-MIB::ipReasmReqds.0 = Counter32: 0
IP-MIB::ipReasmOKs.0 = Counter32: 0
IP-MIB::ipReasmFails.0 = Counter32: 0
IP-MIB::ipFragOKs.0 = Counter32: 0
IP-MIB::ipFragFails.0 = Counter32: 0
IP-MIB::ipFragCreates.0 = Counter32: 0
IP-MIB::ipAdEntAddr.10.105.0.47 = IpAddress: 10.105.0.47
IP-MIB::ipAdEntIfIndex.10.105.0.47 = INTEGER: 1
IP-MIB::ipAdEntNetMask.10.105.0.47 = IpAddress: 255.255.254.0
IP-MIB::ipAdEntBcastAddr.10.105.0.47 = INTEGER: 1
IP-MIB::ipAdEntReasmMaxSize.10.105.0.47 = INTEGER: 1534
RFC1213-MIB::ipRouteDest.10.105.0.0 = IpAddress: 10.105.0.0
RFC1213-MIB::ipRouteIfIndex.10.105.0.0 = INTEGER: 1
RFC1213-MIB::ipRouteMetric1.10.105.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteMetric2.10.105.0.0 = INTEGER: -1
```

```
RFC1213-MIB::ipRouteMetric3.10.105.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteMetric4.10.105.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteNextHop.10.105.0.0 = IpAddress: 255.255.255.255
RFC1213-MIB::ipRouteType.10.105.0.0 = INTEGER: direct(3)
RFC1213-MIB::ipRouteProto.10.105.0.0 = INTEGER: local(2)
RFC1213-MIB::ipRouteAge.10.105.0.0 = INTEGER: 2859685
RFC1213-MIB::ipRouteMask.10.105.0.0 = IpAddress: 255.255.254.0
RFC1213-MIB::ipRouteMetric5.10.105.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteInfo.10.105.0.0 = OID: SNMPv2-SMI::zeroDotZero
IP-MIB::ipNetToMediaIfIndex.1.10.105.1.216 = INTEGER: 1
IP-MIB::ipNetToMediaPhysAddress.1.10.105.1.216 = STRING: 0:c:29:76:44:ce
IP-MIB::ipNetToMediaNetAddress.1.10.105.1.216 = IpAddress: 10.105.1.216
IP-MIB::ipNetToMediaType.1.10.105.1.216 = INTEGER: dynamic(3)
IP-MIB::ipRoutingDiscards.0 = Counter32: 0
IP-MIB::icmpInMsgs.0 = Counter32: 11
IP-MIB::icmpInErrors.0 = Counter32: 0
IP-MIB::icmpInDestUnreachs.0 = Counter32: 0
IP-MIB::icmpInTimeExcds.0 = Counter32: 0
IP-MIB::icmpInParmProbs.0 = Counter32: 0
IP-MIB::icmpInSrcQuenchs.0 = Counter32: 0
IP-MIB::icmpInRedirects.0 = Counter32: 0
IP-MIB::icmpInEchos.0 = Counter32: 11
IP-MIB::icmpInEchoReps.0 = Counter32: 0
IP-MIB::icmpInTimestamps.0 = Counter32: 0
IP-MIB::icmpInTimestampReps.0 = Counter32: 0
IP-MIB::icmpInAddrMasks.0 = Counter32: 0
IP-MIB::icmpInAddrMaskReps.0 = Counter32: 0
IP-MIB::icmpOutMsgs.0 = Counter32: 11
IP-MIB::icmpOutErrors.0 = Counter32: 0
IP-MIB::icmpOutDestUnreachs.0 = Counter32: 0
IP-MIB::icmpOutTimeExcds.0 = Counter32: 0
IP-MIB::icmpOutParmProbs.0 = Counter32: 0
IP-MIB::icmpOutSrcQuenchs.0 = Counter32: 0
IP-MIB::icmpOutRedirects.0 = Counter32: 0
IP-MIB::icmpOutEchos.0 = Counter32: 0
IP-MIB::icmpOutEchoReps.0 = Counter32: 11
IP-MIB::icmpOutTimestamps.0 = Counter32: 0
IP-MIB::icmpOutTimestampReps.0 = Counter32: 0
IP-MIB::icmpOutAddrMasks.0 = Counter32: 0
IP-MIB::icmpOutAddrMaskReps.0 = Counter32: 0
TCP-MIB::tcpRtoAlgorithm.0 = INTEGER: constant(2)
TCP-MIB::tcpRtoMin.0 = INTEGER: 2000 milliseconds
TCP-MIB::tcpRtoMax.0 = INTEGER: 128000 milliseconds
TCP-MIB::tcpMaxConn.0 = INTEGER: 0
```

```
TCP-MIB::tcpActiveOpens.0 = Counter32: 0
TCP-MIB::tcpPassiveOpens.0 = Counter32: 44
TCP-MIB::tcpAttemptFails.0 = Counter32: 0
TCP-MIB::tcpEstabResets.0 = Counter32: 0
TCP-MIB::tcpCurrEstab.0 = Gauge32: 0
TCP-MIB::tcpInSegs.0 = Counter32: 1636
TCP-MIB::tcpOutSegs.0 = Counter32: 1828
TCP-MIB::tcpRetransSegs.0 = Counter32: 0
TCP-MIB::tcpConnState.0.0.0.0.443 = INTEGER: listen(2)
TCP-MIB::tcpConnState.0.0.0.0.80 = INTEGER: listen(2)
TCP-MIB::tcpConnState.0.0.0.0.102 = INTEGER: listen(2)
TCP-MIB::tcpConnLocalAddress.0.0.0.0.443 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnLocalAddress.0.0.0.0.80 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnLocalAddress.0.0.0.0.102 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnLocalPort.0.0.0.0.443 = INTEGER: 443
TCP-MIB::tcpConnLocalPort.0.0.0.0.80 = INTEGER: 80
TCP-MIB::tcpConnLocalPort.0.0.0.0.102 = INTEGER: 102
TCP-MIB::tcpConnRemAddress.0.0.0.0.443 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnRemAddress.0.0.0.0.80 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnRemAddress.0.0.0.0.102 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnRemPort.0.0.0.0.443 = INTEGER: 0
TCP-MIB::tcpConnRemPort.0.0.0.0.80 = INTEGER: 0
TCP-MIB::tcpConnRemPort.0.0.0.0.102 = INTEGER: 0
TCP-MIB::tcpInErrs.0 = Counter32: 0
TCP-MIB::tcpOutRsts.0 = Counter32: 2
UDP-MIB::udpInDatagrams.0 = Counter32: 783
UDP-MIB::udpNoPorts.0 = Counter32: 0
UDP-MIB::udpInErrors.0 = Counter32: 201442
UDP-MIB::udpOutDatagrams.0 = Counter32: 785
UDP-MIB::udpLocalAddress.0.0.0.0.49484 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalAddress.0.0.0.0.34964 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalAddress.0.0.0.0.161 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalPort.0.0.0.0.49484 = INTEGER: 49484
UDP-MIB::udpLocalPort.0.0.0.0.34964 = INTEGER: 34964
UDP-MIB::udpLocalPort.0.0.0.0.161 = INTEGER: 161
SNMPv2-MIB::snmpInPkts.0 = Counter32: 793
SNMPv2-MIB::snmpOutPkts.0 = Counter32: 791
SNMPv2-MIB::snmpInBadVersions.0 = Counter32: 0
SNMPv2-MIB::snmpInBadCommunityNames.0 = Counter32: 0
SNMPv2-MIB::snmpInBadCommunityUses.0 = Counter32: 0
SNMPv2-MIB::snmpInASNParseErrs.0 = Counter32: 0
SNMPv2-MIB::snmpInTooBigs.0 = Counter32: 0
SNMPv2-MIB::snmpInNoSuchNames.0 = Counter32: 0
SNMPv2-MIB::snmpInBadValues.0 = Counter32: 0
```

```
SNMPv2-MIB::snmpInReadOnlys.0 = Counter32: 0
SNMPv2-MIB::snmpInGenErrs.0 = Counter32: 0
SNMPv2-MIB::snmpInTotalReqVars.0 = Counter32: 802
SNMPv2-MIB::snmpInTotalSetVars.0 = Counter32: 0
SNMPv2-MIB::snmpInGetRequests.0 = Counter32: 1
SNMPv2-MIB::snmpInGetNexts.0 = Counter32: 806
SNMPv2-MIB::snmpInSetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpInGetResponses.0 = Counter32: 0
SNMPv2-MIB::snmpInTraps.0 = Counter32: 0
SNMPv2-MIB::snmpOutTooBigs.0 = Counter32: 0
SNMPv2-MIB::snmpOutNoSuchNames.0 = Counter32: 2
SNMPv2-MIB::snmpOutBadValues.0 = Counter32: 0
SNMPv2-MIB::snmpOutGenErrs.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetNexts.0 = Counter32: 0
SNMPv2-MIB::snmpOutSetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetResponses.0 = Counter32: 815
SNMPv2-MIB::snmpOutTraps.0 = Counter32: 0
SNMPv2-MIB::snmpEnableAuthenTraps.0 = INTEGER: disabled(2)
End of MIB
```

B
Simulated snmpwalk output

```
SNMPv2-MIB::sysDescr.0 = STRING: Siemens, SIMATIC S7, IM151-8,
6ES7 151-8AB01-0AB0 , HW: 3, FW: V3.2.6, S C-C1TR95142012
SNMPv2-MIB::sysObjectID.0 = OID: SNMPv2-SMI::zeroDotZero
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (6937) 0:01:09.37
SNMPv2-MIB::sysContact.0 = STRING:
SNMPv2-MIB::sysName.0 = STRING:
SNMPv2-MIB::sysLocation.0 = STRING:
SNMPv2-MIB::sysServices.0 = INTEGER: 78
IF-MIB::ifNumber.0 = INTEGER: 4
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifIndex.3 = INTEGER: 3
IF-MIB::ifIndex.4 = INTEGER: 4
IF-MIB::ifDescr.1 = STRING: Siemens SIMATIC S7, internal, Rack 0, Slot
2
IF-MIB::ifDescr.2 = STRING: Siemens SIMATIC S7, Ethernet Port 1, link,
100 Mbit, full duplex, autonegotiation
IF-MIB::ifDescr.3 = STRING: Siemens SIMATIC S7, Ethernet Port 2,-
IF-MIB::ifDescr.4 = STRING: Siemens SIMATIC S7, Ethernet Port 3,-
IF-MIB::ifType.1 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.2 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.3 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.4 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifMtu.1 = INTEGER: 1518
IF-MIB::ifMtu.2 = INTEGER: 1518
IF-MIB::ifMtu.3 = INTEGER: 1518
IF-MIB::ifMtu.4 = INTEGER: 1518
IF-MIB::ifSpeed.1 = Gauge32: 100000000
IF-MIB::ifSpeed.2 = Gauge32: 100000000
IF-MIB::ifSpeed.3 = Gauge32: 100000000
IF-MIB::ifSpeed.4 = Gauge32: 100000000
IF-MIB::ifPhysAddress.1 = STRING: 0:1b:1b:1a:d6:e0
IF-MIB::ifPhysAddress.2 = STRING: 0:1b:1b:1a:d6:e1
IF-MIB::ifPhysAddress.3 = STRING: 0:1b:1b:1a:d6:e2
IF-MIB::ifPhysAddress.4 = STRING: 0:1b:1b:1a:d6:e3
IF-MIB::ifAdminStatus.1 = INTEGER: up(1)
IF-MIB::ifAdminStatus.2 = INTEGER: up(1)
IF-MIB::ifAdminStatus.3 = INTEGER: down(2)
IF-MIB::ifAdminStatus.4 = INTEGER: down(2)
IF-MIB::ifOperStatus.1 = INTEGER: up(1)
```

```
IF-MIB::ifOperStatus.2 = INTEGER: up(1)
IF-MIB::ifOperStatus.3 = INTEGER: down(2)
IF-MIB::ifOperStatus.4 = INTEGER: down(2)
IF-MIB::ifLastChange.1 = Timeticks: (650) 0:00:06.50
IF-MIB::ifLastChange.2 = Timeticks: (650) 0:00:06.50
IF-MIB::ifLastChange.3 = Timeticks: (650) 0:00:06.50
IF-MIB::ifLastChange.4 = Timeticks: (650) 0:00:06.50
IF-MIB::ifInOctets.1 = Counter32: 0
IF-MIB::ifInOctets.2 = Counter32: 120
IF-MIB::ifInOctets.3 = Counter32: 0
IF-MIB::ifInOctets.4 = Counter32: 0
IF-MIB::ifInUcastPkts.1 = Counter32: 7
IF-MIB::ifInUcastPkts.2 = Counter32: 2
IF-MIB::ifInUcastPkts.3 = Counter32: 0
IF-MIB::ifInUcastPkts.4 = Counter32: 0
IF-MIB::ifInNUcastPkts.1 = Counter32: 0
IF-MIB::ifInNUcastPkts.2 = Counter32: 0
IF-MIB::ifInNUcastPkts.3 = Counter32: 0
IF-MIB::ifInNUcastPkts.4 = Counter32: 0
IF-MIB::ifInDiscards.1 = Counter32: 0
IF-MIB::ifInDiscards.2 = Counter32: 0
IF-MIB::ifInDiscards.3 = Counter32: 0
IF-MIB::ifInDiscards.4 = Counter32: 0
IF-MIB::ifInErrors.1 = Counter32: 0
IF-MIB::ifInErrors.2 = Counter32: 0
IF-MIB::ifInErrors.3 = Counter32: 0
IF-MIB::ifInErrors.4 = Counter32: 0
IF-MIB::ifInUnknownProtos.1 = Counter32: 0
IF-MIB::ifInUnknownProtos.2 = Counter32: 0
IF-MIB::ifInUnknownProtos.3 = Counter32: 0
IF-MIB::ifInUnknownProtos.4 = Counter32: 0
IF-MIB::ifOutOctets.1 = Counter32: 0
IF-MIB::ifOutOctets.2 = Counter32: 8817
IF-MIB::ifOutOctets.3 = Counter32: 0
IF-MIB::ifOutOctets.4 = Counter32: 0
IF-MIB::ifOutUcastPkts.1 = Counter32: 100
IF-MIB::ifOutUcastPkts.2 = Counter32: 61
IF-MIB::ifOutUcastPkts.3 = Counter32: 0
IF-MIB::ifOutUcastPkts.4 = Counter32: 0
IF-MIB::ifOutNUcastPkts.1 = Counter32: 0
IF-MIB::ifOutNUcastPkts.2 = Counter32: 0
IF-MIB::ifOutNUcastPkts.3 = Counter32: 0
IF-MIB::ifOutNUcastPkts.4 = Counter32: 0
IF-MIB::ifOutDiscards.1 = Counter32: 0
```

```
IF-MIB::ifOutDiscards.2 = Counter32: 0
IF-MIB::ifOutDiscards.3 = Counter32: 0
IF-MIB::ifOutDiscards.4 = Counter32: 0
IF-MIB::ifOutErrors.1 = Counter32: 0
IF-MIB::ifOutErrors.2 = Counter32: 0
IF-MIB::ifOutErrors.3 = Counter32: 0
IF-MIB::ifOutErrors.4 = Counter32: 0
IF-MIB::ifOutQLen.1 = Gauge32: 0
IF-MIB::ifOutQLen.2 = Gauge32: 0
IF-MIB::ifOutQLen.3 = Gauge32: 0
IF-MIB::ifOutQLen.4 = Gauge32: 0
IF-MIB::ifSpecific.1 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.2 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.3 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.4 = OID: SNMPv2-SMI::zeroDotZero
IP-MIB::ipForwarding.0 = INTEGER: notForwarding(2)
IP-MIB::ipDefaultTTL.0 = INTEGER: 30
IP-MIB::ipInReceives.0 = Counter32: 978
IP-MIB::ipInHdrErrors.0 = Counter32: 0
IP-MIB::ipInAddrErrors.0 = Counter32: 0
IP-MIB::ipForwDatagrams.0 = Counter32: 0
IP-MIB::ipInUnknownProtos.0 = Counter32: 0
IP-MIB::ipInDiscards.0 = Counter32: 0
IP-MIB::ipInDelivers.0 = Counter32: 988
IP-MIB::ipOutRequests.0 = Counter32: 994
IP-MIB::ipOutDiscards.0 = Counter32: 0
IP-MIB::ipOutNoRoutes.0 = Counter32: 10
IP-MIB::ipReasmTimeout.0 = INTEGER: 120 seconds
IP-MIB::ipReasmReqds.0 = Counter32: 0
IP-MIB::ipReasmOKs.0 = Counter32: 0
IP-MIB::ipReasmFails.0 = Counter32: 0
IP-MIB::ipFragOKs.0 = Counter32: 0
IP-MIB::ipFragFails.0 = Counter32: 0
IP-MIB::ipFragCreates.0 = Counter32: 0
IP-MIB::ipAdEntAddr.10.105.0.97 = IpAddress: 10.105.0.97
IP-MIB::ipAdEntIfIndex.10.105.0.97 = INTEGER: 1
IP-MIB::ipAdEntNetMask.10.105.0.97 = IpAddress: 255.255.254.0
IP-MIB::ipAdEntBcastAddr.10.105.0.97 = INTEGER: 1
IP-MIB::ipAdEntReasmMaxSize.10.105.0.97 = INTEGER: 1534
RFC1213-MIB::ipRouteDest.10.105.0.0 = IpAddress: 10.105.0.0
RFC1213-MIB::ipRouteIfIndex.10.105.0.0 = INTEGER: 1
RFC1213-MIB::ipRouteMetric1.10.105.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteMetric2.10.105.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteMetric3.10.105.0.0 = INTEGER: -1
```

```
RFC1213-MIB::ipRouteMetric4.10.105.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteNextHop.10.105.0.0 = IpAddress: 255.255.255.255
RFC1213-MIB::ipRouteType.10.105.0.0 = INTEGER: direct(3)
RFC1213-MIB::ipRouteProto.10.105.0.0 = INTEGER: local(2)
RFC1213-MIB::ipRouteAge.10.105.0.0 = INTEGER: 731
RFC1213-MIB::ipRouteMask.10.105.0.0 = IpAddress: 255.255.254.0
RFC1213-MIB::ipRouteMetric5.10.105.0.0 = INTEGER: -1
RFC1213-MIB::ipRouteInfo.10.105.0.0 = OID: SNMPv2-SMI::zeroDotZero
IP-MIB::ipNetToMediaIfIndex.1.10.105.0.1 = INTEGER: 1
IP-MIB::ipNetToMediaPhysAddress.1.10.105.0.1 = STRING: 0:b0:64:12:af:4
IP-MIB::ipNetToMediaNetAddress.1.10.105.0.1 = IpAddress: 10.105.0.1
IP-MIB::ipNetToMediaType.1.10.105.0.1 = INTEGER: dynamic(3)
IP-MIB::ipRoutingDiscards.0 = Counter32: 0
IP-MIB::icmpInMsgs.0 = Counter32: 22
IP-MIB::icmpInErrors.0 = Counter32: 0
IP-MIB::icmpInDestUnreachs.0 = Counter32: 8
IP-MIB::icmpInTimeExcds.0 = Counter32: 0
IP-MIB::icmpInParmProbs.0 = Counter32: 0
IP-MIB::icmpInSrcQuenchs.0 = Counter32: 0
IP-MIB::icmpInRedirects.0 = Counter32: 0
IP-MIB::icmpInEchos.0 = Counter32: 7
IP-MIB::icmpInEchoReps.0 = Counter32: 7
IP-MIB::icmpInTimestamps.0 = Counter32: 0
IP-MIB::icmpInTimestampReps.0 = Counter32: 0
IP-MIB::icmpInAddrMasks.0 = Counter32: 0
IP-MIB::icmpInAddrMaskReps.0 = Counter32: 0
IP-MIB::icmpOutMsgs.0 = Counter32: 22
IP-MIB::icmpOutErrors.0 = Counter32: 0
IP-MIB::icmpOutDestUnreachs.0 = Counter32: 8
IP-MIB::icmpOutTimeExcds.0 = Counter32: 0
IP-MIB::icmpOutParmProbs.0 = Counter32: 0
IP-MIB::icmpOutSrcQuenchs.0 = Counter32: 0
IP-MIB::icmpOutRedirects.0 = Counter32: 0
IP-MIB::icmpOutEchos.0 = Counter32: 7
IP-MIB::icmpOutEchoReps.0 = Counter32: 7
IP-MIB::icmpOutTimestamps.0 = Counter32: 0
IP-MIB::icmpOutTimestampReps.0 = Counter32: 0
IP-MIB::icmpOutAddrMasks.0 = Counter32: 0
IP-MIB::icmpOutAddrMaskReps.0 = Counter32: 0
TCP-MIB::tcpRtoAlgorithm.0 = INTEGER: constant(2)
TCP-MIB::tcpRtoMin.0 = INTEGER: 2000 milliseconds
TCP-MIB::tcpRtoMax.0 = INTEGER: 128000 milliseconds
TCP-MIB::tcpMaxConn.0 = INTEGER: 0
TCP-MIB::tcpActiveOpens.0 = Counter32: 0
```

```
TCP-MIB::tcpPassiveOpens.0 = Counter32: 0
TCP-MIB::tcpAttemptFails.0 = Counter32: 0
TCP-MIB::tcpEstabResets.0 = Counter32: 0
TCP-MIB::tcpCurrEstab.0 = Gauge32: 1
TCP-MIB::tcpInSegs.0 = Counter32: 0
TCP-MIB::tcpOutSegs.0 = Counter32: 0
TCP-MIB::tcpRetransSegs.0 = Counter32: 0
TCP-MIB::tcpConnState.0.0.0.0.443 = INTEGER: listen(2)
TCP-MIB::tcpConnState.0.0.0.0.80 = INTEGER: listen(2)
TCP-MIB::tcpConnState.0.0.0.0.102 = INTEGER: listen(2)
TCP-MIB::tcpConnLocalAddress.0.0.0.0.443 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnLocalAddress.0.0.0.0.80 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnLocalAddress.0.0.0.0.102 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnLocalPort.0.0.0.0.443 = INTEGER: 443
TCP-MIB::tcpConnLocalPort.0.0.0.0.80 = INTEGER: 80
TCP-MIB::tcpConnLocalPort.0.0.0.0.102 = INTEGER: 102
TCP-MIB::tcpConnRemAddress.0.0.0.0.443 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnRemAddress.0.0.0.0.80 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnRemAddress.0.0.0.0.102 = IpAddress: 0.0.0.0
TCP-MIB::tcpConnRemPort.0.0.0.0.443 = INTEGER: 0
TCP-MIB::tcpConnRemPort.0.0.0.0.80 = INTEGER: 0
TCP-MIB::tcpConnRemPort.0.0.0.0.102 = INTEGER: 0
TCP-MIB::tcpInErrs.0 = Counter32: 0
TCP-MIB::tcpOutRsts.0 = Counter32: 0
UDP-MIB::udpInDatagrams.0 = Counter32: 1
UDP-MIB::udpNoPorts.0 = Counter32: 4294
UDP-MIB::udpInErrors.0 = Counter32: 120
UDP-MIB::udpOutDatagrams.0 = Counter32: 0
UDP-MIB::udpLocalAddress.0.0.0.0.49484 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalAddress.0.0.0.0.34694 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalAddress.0.0.0.0.161 = IpAddress: 0.0.0.0
UDP-MIB::udpLocalPort.0.0.0.0.49484 = INTEGER: 0
UDP-MIB::udpLocalPort.0.0.0.0.34964 = INTEGER: 0
UDP-MIB::udpLocalPort.0.0.0.0.161 = INTEGER: 0
SNMPv2-MIB::snmpInPkts.0 = Counter32: 436
SNMPv2-MIB::snmpOutPkts.0 = Counter32: 437
SNMPv2-MIB::snmpInBadVersions.0 = Counter32: 0
SNMPv2-MIB::snmpInBadCommunityNames.0 = Counter32: 0
SNMPv2-MIB::snmpInBadCommunityUses.0 = Counter32: 0
SNMPv2-MIB::snmpInASNParseErrs.0 = Counter32: 0
SNMPv2-MIB::snmpInTooBigs.0 = Counter32: 0
SNMPv2-MIB::snmpInNoSuchNames.0 = Counter32: 0
SNMPv2-MIB::snmpInBadValues.0 = Counter32: 0
SNMPv2-MIB::snmpInReadOnlys.0 = Counter32: 0
```

```
SNMPv2-MIB::snmpInGenErrs.0 = Counter32: 0
SNMPv2-MIB::snmpInTotalReqVars.0 = Counter32: 435
SNMPv2-MIB::snmpInTotalSetVars.0 = Counter32: 0
SNMPv2-MIB::snmpInGetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpInGetNexts.0 = Counter32: 436
SNMPv2-MIB::snmpInSetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpInGetResponses.0 = Counter32: 0
SNMPv2-MIB::snmpInTraps.0 = Counter32: 0
SNMPv2-MIB::snmpOutTooBigs.0 = Counter32: 0
SNMPv2-MIB::snmpOutNoSuchNames.0 = Counter32: 1
SNMPv2-MIB::snmpOutBadValues.0 = Counter32: 0
SNMPv2-MIB::snmpOutGenErrs.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetNexts.0 = Counter32: 0
SNMPv2-MIB::snmpOutSetRequests.0 = Counter32: 0
SNMPv2-MIB::snmpOutGetResponses.0 = Counter32: 435
SNMPv2-MIB::snmpOutTraps.0 = Counter32: 0
SNMPv2-MIB::snmpEnableAuthenTraps.0 = INTEGER: disabled(2)
End of MIB
```