CESNET Technical Report 4/2012

Monitoring of Malicious Traffic in IP Telephony Infrastructure

Jakub Šafařík – Filip Řezáč – Miroslav Vozňák

CESNET z.s.p.o., Zikova 4, Praha, Czech Republic
safarik@cesnet.cz, filip@cesnet.cz, voznak@cesnet.cz

Received 14.12.2012

Abstract

The technical report aims at monitoring and analysis of malicious traffic in Voice over IP networks. The traffic is monitored via prepared set of honeypot applications, which can monitor various aspects of nowadays IP telephony infrastructure – from an end-point-device to PBX (Private branch exchange) functionality emulation. Using a honeypot brings valuable data about attacks with no threat to production systems. Grouping honeypots in different locations into one cloud solution provides more accurate data and independent results. An analysis of a honeypot data is crucial for further improvement of existing security mechanism in IP telephony infrastructure. The report contains information about each honeypot included in proposed cloud solution with an analysis of a detected malicious activity.

Keywords: Artemisa, Dionaea, honeypot, Kippo, VoIP attacks, VoIP honeypot

1 Introduction

The report describes the use of honeypots in a VoIP infrastructure. These systems become increasingly necessary as the number of IP-based telephony solutions rises. Nearly all large companies today rely on some kind of IP telephony in their internal communication. This situation only induces greater attacker interest in these services. Nowadays, many companies have experienced abuse such as social engineering or spit calls.

The way to protect this infrastructure is to keep up with hackers and constantly improve security mechanisms. But achieving this simple goal is not easy at all. The basic rule is to keep all systems and their versions up to date, with at least access policies properly set and encryption of all crucial data. But this is not always possible in VoIP systems. The question is how do we find the system's bottleneck? There is an option in security audit of whole VoIP network or using some penetration testing software [8]. Other way is in monitoring of malicious traffic with honeypots.

Honeypots lure hackers through exposed security holes or vulnerabilities while emulating target services. Using honeypots we obtain real data about hacker activities and map actual attacks in the network, which is otherwise not possible [6,7].

The main purpose of a honeypot is to simulate the real system and interact with anyone in the same way as the production system would. It watches the behavior of anyone who interacts with it [2]. This report provides a close look on individual honeypot applications and its integration in honeypot cloud solution.

2 Honeypot Network Concept

Running individual honeypot application on a single server brings valuable information. Exceeding numbers of running honeypots, especially if running in different networks on various

geographic locations, causes unwanted overhead in data analysis. Without an automatic aggregation mechanism, this situation leads to decreasing profit from honeypot's data. Figure 1 illustrates a honeypot cloud concept with centralized point for data analysis.

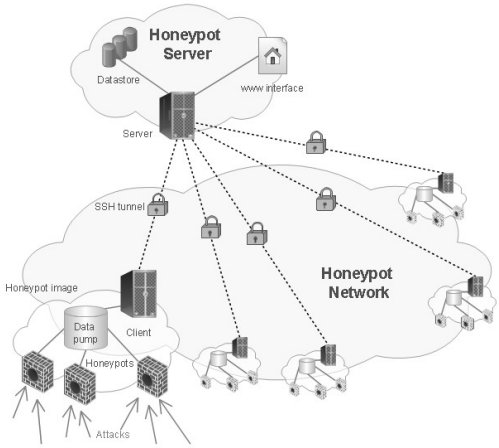

Fig. 1. Honeypot Network Concept.

Whole concept is based on prepared honeypot image. The image can run on virtual or physical machine, and there is no need in installing software or configuration of the server. This image contains all software needed for correct honeypot functionality, data pump and client application for communication with server. Deployment of this honeypot image should be as easy as possible.

Data from all honeypot images goes through data pump and cleaning function. Client sends prepared data to server application via encrypted SSH tunnel. Centralized server serves as a data store for all honeypots data and monitors all running honeypot images. Server side application transforms and integrates data from client to data store. Results are accessible via web interface for further analysis and security improvements.

This architecture provides unified honeypot images with easy scalability for other honeypot modules. We are oriented on VoIP infrastructure, but honeypot image can contain various honeypots or different honeypots in each image connected to one server. This solution gives us flexible platform for independent observation and evaluation of real threats.

3 Honeypot Features

Most important parts of honeypot image are honeypots application, which log and monitor malicious activity. Nowadays exists many honeypot solutions emulating both single and multiple services. Because of our focus on IP telephony we decide to deploy VoIP oriented honeypots. Each honeypot emulate different aspects of VoIP network, its features are described below.

3.1 Artemisa Features

An *artemisa* honeypot can be deployed in any VoIP infrastructure which uses a SIP protocol. In this infrastructure it plays a role of a regular SIP phone (see Fig. 2).

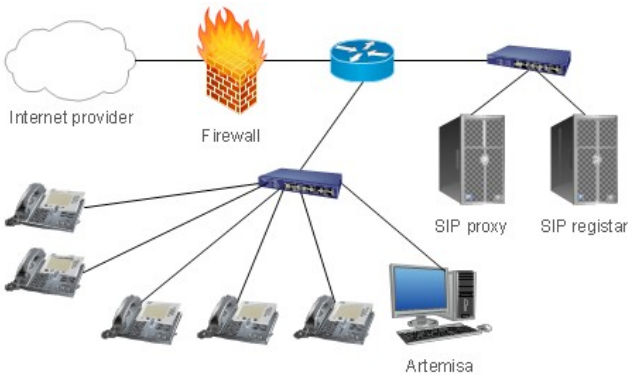

Fig. 2. A VoIP Topology with a Honeypot.

The programme connects to SIP proxy with the extensions defined in a configuration file. The extensions should be within the range which is typically used for real accounts. The main purpose is to establish a better masking against the potential attackers. *Artemisa* itself does not simulate PBX but rather an active end point device.

Once the call is established on one of *artemisa* extensions, the honeypot simply answers the call. At the same time, it starts to examine the incoming SIP message. *Artemisa* then classifies the call and saves the result for a further review by the security administrator (Fig. 3).

The message is classified in the following steps. First of all, *artemisa* looks for fingerprints of well-known attack tools. If the attacker uses some popular hacking tool, the fingerprint of this tool can easily reveal the malicious intentions. Then it checks domain names and SIP ports on the attacker side (provided they are really opened). There is also a similar check for media ports. Requested URI are also checked, as well as the ACK message received from the user. Finally, *artemisa* checks the received RTP stream – provided a RTP stream was established (the audio trail of the received call can be stored in a WAV format).

This sequence of procedures helps *Artemisa* to classify the call. The result is then shown in a console. The results can be saved into a pre-defined folder or they can be sent as a notification by e-mail.

```
... output ommitted ...

| | Category: Interactive attack

+ Checking if media port is opened...
|
| No RTP info delivered.
|
| Category: Spoofed message

... output ommitted ...

+ The message is classified as:
| Attack tool
| Spoofed message
| Interactive attack
| Dial plan fault
| Scanning
| Ringing

************************************ Correlation **************

Artemisa concludes that the arrived message is likely to be:

* The attack was created employing the tool SIPVicious.
* A flooding attack.

... output ommitted ...
```

Fig. 3. An Example of the Output File.

Once the call has been examined, a series of bash scripts is executed. These scripts are executed with pre-defined arguments. *Artemisa* can launch some countermeasures against the incoming attacks.

3.2 Kippo Features

The second used honeypot is based on different foundations. It is not VoIP oriented as *artemisa*. It simulates a SSH server. When someone tries to connect to a server with a honeypot running on it, the *iptables* application redirects this user to the honeypot. This happens where the user IP address is not included in the list of permitted IP address.

Once the connection with the honeypot is established, the attacker must enter correct username and password. These are set to the most used username root and password is the second most common combination of numbers 123456 (Table 1 lists Top 10 most frequently used passwords). Other combinations for the root access can be added to data/pass.db file.

Kippo logs every login attempt. Where the entered combination is valid, the intruder is granted access to a fake filesystem. Every command entered into the honeypot is logged and behaviour typical for a particular command is emulated (for the most common commands only). If the user tries to download something from the Internet, *kippo* saves this file into a secure folder for further examination.

All logs made by *kippo* are saved in a MySQL database which facilitates the subsequent analysis.

3.3 Dionaea Features

All previously mentioned honeypots were single service oriented ones. *Dionaea* belongs to a multi-service oriented honeypot which can simulate many services at a time. Typically are information from these multiple services only general but *dionaea* serves only small number of them like SMB (Microsoft's printers, files, serial ports sharing protocol), HTTP, FTP, TFTP, MSSQL (Microsoft SQL server), SIP protocols. Attackers abuse these protocols in most cases. *Dionaea* has also ability to save malicious content needed by hackers securely, but as a contrary to *Kippo* can also emulate code from these files.

Describing features of all this protocols is beyond the scope of this technical report and further features focus only on the SIP protocol. *Dionaea* works in a different way as *Artemisa*. There is no need for connecting to an external (or production) VoIP server. It simply waits for any SIP message and tries to answer it. It supports all SIP requests from RFC 3261 (REGISTER, INVITE, ACK, CANCEL, BYE, OPTIONS). *Dionaea* supports multiple SIP sessions and RTP audio streams (data from stream can be recorded). For better simulation of a real IP telephony system, it is possible to configure different user agent phone mimics with custom username, password combinations. There is functionality for a different pickup time on simulated phones via pickup delay feature. All traffic is monitored, and logs are saved in plain-text files and in sqlite database.

4 Honeypot Usability Tests

4.1 Artemisa's Usability Tests

As mentioned before, *artemisa* investigates all traffic which is routed to its extensions. That's not the whole truth. *Artemisa* can run in three different modes depending on the settings in the behavior.conf file. These modes are called passive, active and aggressive.

In the passive mode, *artemisa* only takes the incoming calls and answers them. Using the active mode, we can achieve the same functionality as in the passive mode. In addition, *artemisa* starts to examine the incoming SIP messages as described above. The last mode – the aggressive mode – attacks the intruder with its own bash script (the scripts are located under the /scripts directory). Typically, we run *artemisa* in the active mode so that all SIP messages routed to the honeypot are analysed.

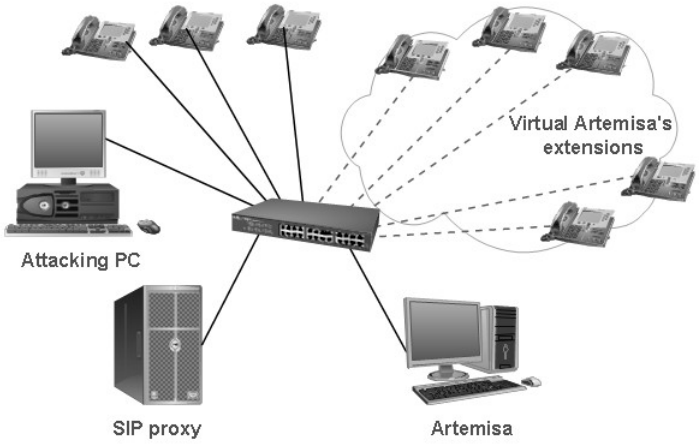

Fig. 4. Artemisa Testing Topology.

To test the usefulness of an *artemisa* honeypot, we prepared some experiments in our testing topology (fig. 4). We built a simple VoIP network with asterisk running as PBX and some end-point hardware and software phones. The honeypot was installed on a machine inside our network with five extensions. These extensions were running in the active mode. Since we are developing our own IPS (Intrusion prevention system), we have chosen to test the honeypot under test scenarios similar to that IPS system.

First of all, we should start scanning the whole network from the point of view of a typical intruder. Many applications can be used for this purpose. We used two such applications – *nmap* and *SIPVicious* (*svmap*).

Both these applications yielded useful information. Yet neither *nmap* nor *svmap* was detected by the honeypot. In case of *svmap* there was information about the incoming SIP message, but this message was not analysed. No results were created after the network was scanned. This behaviour was quite surprising as typically, each attack starts by scanning the network. *Artemisa* should take account of such situations.

With *svmap* we know about the running user-agent at honeypot's IP address (Fig. 5)

158.196.244.241:5060 | Twinkle/1.4.2 | T-Com Speedport W500V / Firmware v1.37 MxSF/v3.2.6.26

Fig. 5. Svmap Application Output.

Using this information we began looking for extensions running in the testing network. Direct scanning of the SIP proxy server was not detected by the honeypot, but when we use the *svwar* tool directly against the IP address on which a honeypot is running, we get information about all active extensions. This scan was recognized by the honeypot and an adequate result file has been created. *Artemisa* correctly concludes that messages received came from a *SIPVicious* scanner. On the other hand we know from the *SIPVicious* output that these extensions do not behave as normal clients. This can stir up more caution on the side of the intruder.

The aim of other attacks was to flood the client´s device with various types of SIP messages. Using some of these attacks, the intruder can achieve a DoS attack on a closed group of end-point devices [5]. For this kind of attack we used a number of tools including *udpflood*, *rtpflood*, *inviteflood* and *sipp*.

Each of these applications can launch a simple DoS attack. As *artemisa* is a mere VoIP honeypot, it only detects attacks using the SIP protocol. Accordingly, only flood attacks from *inviteflood* and *sipp* [8] were detected. In case of *inviteflood*, the application was successfully recognized thanks to its well-known fingerprint.

The sip application was not designed for hacking or penetration testing but this functionality can be achieved easily. We used specific call scenarios with a similar impact as the above mentioned flooding tools. Using *sipp* we can generate a high number of SIP messages which was immediately detected as a flood attack by the honeypot. In this situation, the whole honeypot stopped responding shortly and no

result was recorded for the attack at all. Mere 250 SIP messages per second caused this situation. If we use lower sending rates, the attack was recognized well and the output file was successfully created.

Identifying a spit call is one of the most important features of the honeypot. We used application called *Spitfile* for simulating these calls. *Spitfile* is an open-source SIP penetration tool [4]. Using this tool, we can easily generate arbitrary calls. All of these calls were successfully detected by a*rtemisa* and the appropriate output files were generated.

At last we tried to make a call to the honeypot extensions with hardware and software phone. Calls were marked as a scanning and a ringing attack in both cases. So it seems that *artemisa* evaluated almost every SIP message aiming at its extensions as some kind of attack.

The results from the detected attacks are stored in the results/directory inside the *artemisa* folder. The output is in a simple text format and in html format, both of them containing the same data.

All functionalities mentioned before concern honeypots running in the active mode. The aggressive mode looks more interesting with its ability to counterattack the intruder. *Artemisa* contains three bash scripts to stop malicious activity.

Let's start with the last script on_spit.sh. Inside this script, there is only one comment. This comment may activate a firewall rule, but the command is not included. Even the remaining scripts do not contain anything but comments inside. A simple condition (commented) is included in the on_scanning.sh script, which runs a python script to crash the scanning by the *SIPVicious* application (but only this particular application). The on_flood.sh script has a commented command inside to apply an *iptables* rule on the IP address and port. These are given by a parameter. This solution is not bad but if we want to block some traffic, there is a chance that a false positive attack will be blocked, so some automatic recovery mechanism should also be included. This can be easily solved by adding another script. This script will remove the rule after a certain interval. The main issue in blocking traffic using *iptables* is that the command applies the rule on a local machine. However, it only blocks the consequences on the honeypot, not on the main firewall which protects the whole infrastructure. This feature makes the aggressive mode useless against the attack of any intruder.

Using the honeypot in the passive mode is worthless because *artemisa* only answers the call with no further analysis and without results being saved to a file.

4.2 Kippo Data Analysis

We use a *kippo* honeypot to analyse SSH traffic in a real network with seven active monitoring sensors. The honeypot has been active and gathering data for a month. During this period, 873342 connection attempts were observed.

Only a small part of these connections was successful. Table 1 lists ten most frequently used password combinations enabling connections.

Table 1. 10 Most Fequently Used Password Combinations.

Password	Count
	28146
123456	17625
password	6325
1234	5663
12345	5501
123	5342
1qa2ws3ed	5278
a	5121
test	4743
qwerty	4601

As we can see from the table, password 123456 was used 17625 times. The correct username root was used only in 2551 cases. These 2551 cases led to a connection to a fake filesystem. An intruder then typically uploads some kind of a script which should be used for a DoS attack on an outside server. The in-depth analysis of the attacker's input is beyond the scope of this report.

Another interesting fact acquired from the honeypot is the approximate location of the origin of the attacker's connection (see Fig. 6).

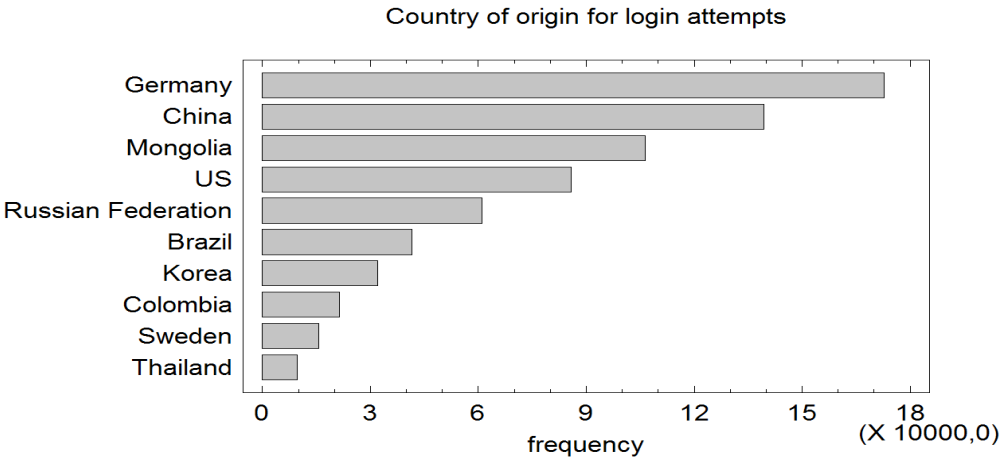

Fig. 6. Country of Origin for Logging Attempts.

The figure only shows first ten positions. Attacks from Germany account for 25.21% of the total connection attempts. China came second with 20.33% and Mongolia third with 15.52%. Almost 75% of connection attempts were made from one of the first four countries.

As mentioned above, we have seven active *kippo* sensors. Traffic to these sensors is redirected from whole C network range, but approximately only 30% of all attempts. With nearly a million attempts, each sensor was tested every 23 seconds.

4.3 Dionaea Data Analysis

Dionaea was monitoring malicious traffic for 18 days. Number of attack is not as high as in *kippo* case but still high enough. Distribution of an attacking IP addresses shows Fig.7.

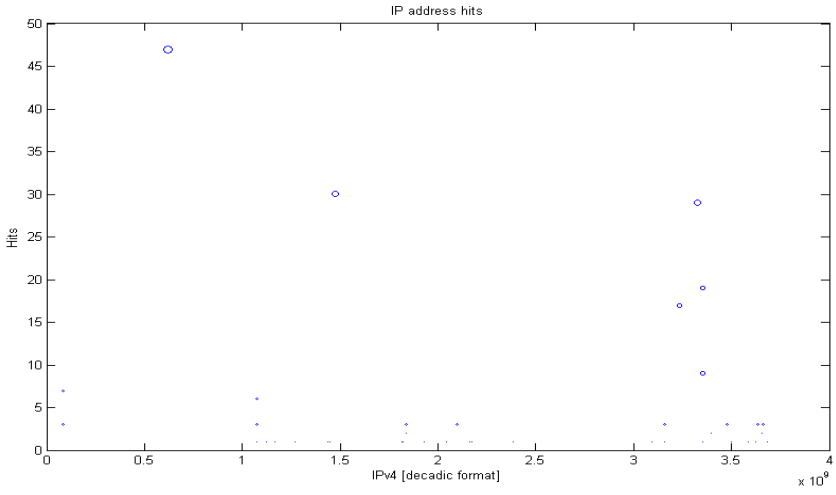

Fig. 7. Hits on Attacker IP Address.

Most attacking attempts came from Israel (first peak in Fig. 7). The second most used IP was originated in Germany and third in Russia.

Dionaea provides also additional information about attacks like used SIP messages, SIP header information, SDP and RTP statistics. Interesting is typical attack behaviour based on send SIP messages. All attacks occur in typical sequences. Table 2 shows gathered SIP message data. In column groups is the number of SIP message's grouped by different connections. One connection is a single session with emulated honeypot's SIP proxy. Ratio simply illustrates average number of messages in each connection group.

Table 2. SIP Messages Types Analysis.

SIP Message	Groups	Count	Ratio
ACK	40	303	7,575
BYE	4	4	1,000
CANCEL	1	11	11
INVITE	18	85	4,722
OPTIONS	76	76	1,000
REGISTER	28	1745	62,321

Most of can attacks can be divided into 2 groups. First represents various types of a PBX scanning & probing. Attacker send OPTION message and wait for an answer or simply try to place a call with immediate cancelation (it means INVITE message followed by CANCEL message).

Other group represents flood attacks. Our previous tests confirmed an extraordinary vulnerability of SIP proxy against OPTIONS floods, but attackers still use only REGISTER message flooding. These results are also confirmed by testing that has been done before and is described below.

4.4 DoS Effectiveness Against SIP Server

A testing topology for measure DoS effectiveness against SIP server was made of Asterisk, some end-point devices and an attacker's computer. Many tools were applied by tester at hacker's computer. Most important was *sipp* (in repository named as sip-tester) for simulating calls on SIP server. With our own scenario, *sipp* can be also used as attacking tool. We used flooding tools such as *inviteflood*, *udpflood*, *flood2* and *juno*. These are not accessible from the repository, but source code is available for downloading on the internet. Tested were application for network scanning – *hping*, *fping*, *nmap*, *SipVicious* [9, 10].

All of these applications mentioned before we adopted for purposes of SIP proxy testing. All attacks follow the same scenario. Malicious packets was launched after 10s, and lasted for 60s, then we observed recovering of server after attack for 30 seconds. Graph on figure 8 illustrates the impact of flooding SIP proxy with different SIP messages.

4.4.1 CPU Depletion Attacks

Sipp application is typically used to carry out stress tests of SIP proxy. Our scenario allows us to flood the target SIP server with different types of SIP messages. There is also *inviteflood* application, which sends only INVITE messages. With *inviteflood*, we can change only count of INVITE messages sent to the server and not the rate of packets per second. In order to compare efficiency of flood attacks based on different SIP methods, we adjusted each of them to the same sending rate. It was set to 250 messages per second. Efficiency of SIP messages shows figure 8.

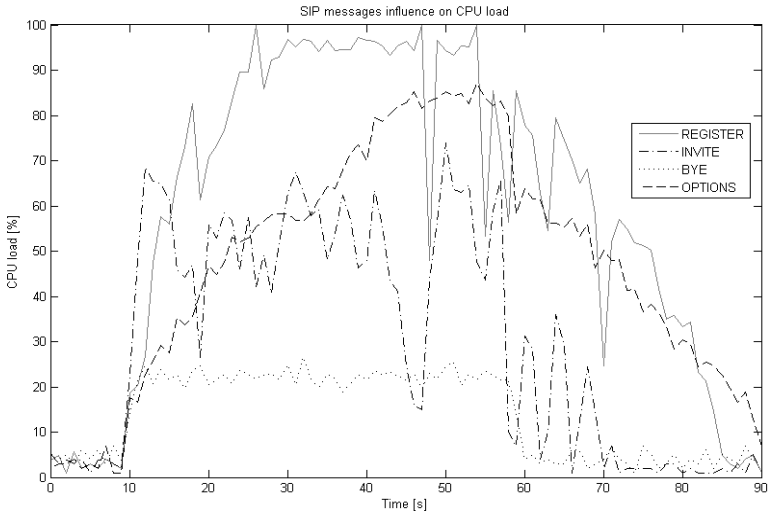

Figure 8: The Impact of Different SIP Messages on SIP Server's CPU Load.

In figure 8 we omitted attacks with CANCEL and ACK messages because its impact was very similar as an attack with BYE message. It is clear that the biggest threat is flooding the server with REGISTER and OPTIONS messages. INVITE message flooding is also critical but only with higher rates. Even time required for a recovery of server after the attack was longer in case of REGISTER and OPTION messages. For reaching equal CPU load on the server as with REGISTER message, we must use a ten times bigger load with INVITE messages. That is because SIP server does not have enough information in a register message, so it tries to find or guess missing data, which consumes more CPU resources. Flooding with OPTIONS messages (which have a structure similar to INVITE message) consumes the more resources the longer an attack last. Performing attacks, with BYE, CANCEL and ACK messages, have a small impact on server. With increasing rate was attack more and more like flooding the server with UDP packets.

Appropriate rules using SNORT and SnortSAM were created against all flooding attacks. The issue of creating effective rules in SNORT system is described here [10]. Then system successfully suppressed all attacks as is depicted in Fig. 9 on example with REGISTER flooding.

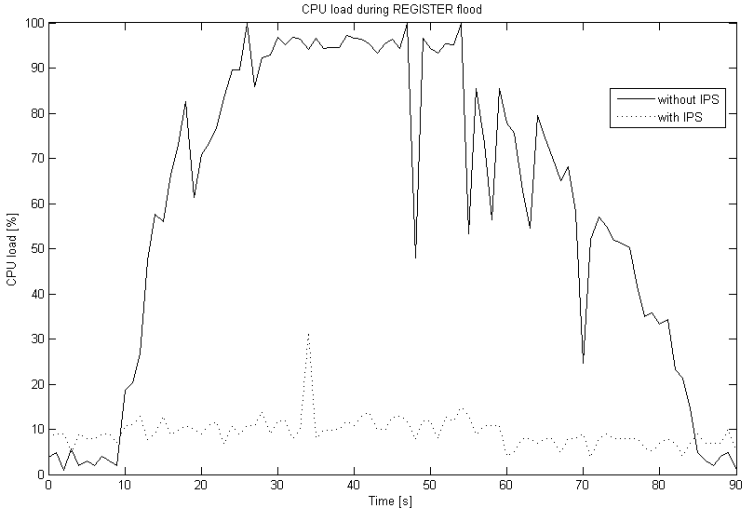

Figure 9: IPS defense influence on the server's CPU load

4.4.2 **Link Flooding Attacks**

Udpflood floods the target SIP server with useless UDP packets. It aim's on consuming the capacity of link to server more than to consume its CPU resources. With running attack, it is not possible to register on SIP server or to create call. Current calls routing through SIP server are also interrupted. With IPS is possible to detect this attack, but its blocking on the server side is effortless because the attack do not allow correct packets to reach SIP server. Elimination of this attack right on the server is not possible.

4.4.3 **TCP SYN Flood Attack**

The last kind of attack against SIP server was to flood it with TCP SYN flag set packets. Applications flood2 and juno were used. Both applications interrupt all communication with server. An active IPS made the situation even worst because it consumes almost all CPU resources for analyzing incoming traffic.

4.4.4 **Assessment of Results**

The performed tests clearly indicate SIP server's vulnerability against DoS attacks. Even with the server running on a limited physical machine, it is possible to implement effective security mechanism. But some of the attacks cannot be blocked right on the server as was shown in provided tests. The most dangerous attacks include flooding with REGISTER, INVITE and OPTIONS messages. ACK, CANCEL and BYE messages are nearly harmless on lower rates and on higher rates have the same impact as udpflood.

The defense against unblockable attacks lies in secure network topology like inclusion of a DMZ (demilitarized zone) located between trusted and untrusted zone (inner and outer). The purpose of the DMZ is to separate the safe inner part of network from the rather dangerous outer part. The potential attack from the inside network should be another source of threat. So security mechanisms like encryption, VoIP VLANs and methods as dynamic ARP inspection, DHCP snooping will provide an adequate response to security breaches. Using a honeypot in DMZ is an inspiration for further security precautions to be implemented. Although encryption is a good way to increasing security, we cannot deploy it, if we do not have its full support on all end-point devices.

5 Conclusion

All the tests which we carried out on a VoIP honeypots gave us a solid look on its features. The main goal of this technical report is to show the possibilities of honeypots and analyse gathered data. Another goal is to consider its deploying in image for a honeypot network concept. The implementation of honeypot network concept is a part of future plans.

The test revealed that *artemisa* is not the silver bullet solution for discovering all security threats. Its main disadvantage is that it does not recognize a scanning attempt into the infrastructure. It is freezing while analysing a flooding attack by the *sipp* application was also quite surprising. It was a result of the flooding at higher rates. Accordingly, *artemisa* cannot handle such a big mass of SIP messages.

Typical *artemisa* behaviour configuration is active mode for call inspection. *Artemisa* honeypot is not a simulation of PBX, but rather of an endpoint device. Despite the fact that it freezes at high flooding rates, its principal utility is to detect suspicious call activity and spit attacks. Similar functionality has also *dionaea* honeypot without typical *artemisa* flaws. Reasons for not including *artemisa* into final image are discussed in section 4.1.

SSH honeypot *kippo* gives us a good idea about connection attempts on a standard SSH port. The information acquired was used to gain a better understanding of the attacker's behaviour. It is also a good source of malicious IP addresses. The database containing username and password combinations used is also very interesting. After some time, the rate of connection attempts decreases. This happens once the honeypot has been discovered. It is necessary to change the honeypot configuration regulary.

Dionaea honeypot is great in simulating a SIP proxy. The data gathered show real attacks and gave us valuable feedback for improving existing security mechanisms. Quite surprising was flooding attacks

using only REGISTER messages. We found no connection between IP addresses used for attacks on both *kippo* and *dionaea*.

At this point a single honeypot image is deployed on the CESNET2, where using *IPtables* only ports 5060 and 22 are allowed. The SIP port 5060 traffic is monitored by *dionaea* tool and any incidents are recorded in the SQLite database. Similarly, on port 22 traffic is analyzed by *kippo* where any suspicious activity is saved in the MySQL database. The goal is to filter the data, edit and send them to a central registry of security incidents - Mentat. All Mentat participants can use aggregated data for own purposes. Implementation of this aggregator between honeypots and Mentat is scheduled for 2/2013. [11] Future possibilities of honeypot data analysis lies in involving a distributed system for communication with IPS systems and immediate reaction on attacks on one system to improving security against detected attacks on other connected systems. This automatic defence mechanism should detect real threats and raise security in systems without malicious activity proactively.

6 Acknowledgment

This work has been supported by the Ministry of Education of the Czech Republic within the project LM2010005.

7 References

[1] F. REZAC, M. VOZNAK, J. RUZICKA: *Security Risks in IP Telephony*, CESNET CONFERENCE 2008-SECURITY, pp.31-38, 2008.

[2] L. SPITZNER: *Honeypots: Tracking Hackers*, Addison-Wesley Proffesional, 2002.

[3] D. SISALEM, J. KUTHAN, T.S. ELHERT, F. FRAUNHOFER: *Denial of Service Attacks Targeting SIP VoIP Infrastructure: Attack Scenarios and Prevention Mechanisms*, IEEE Network, 2006.

[4] F. REZAC, M. VOZNAK: *SIP Penetration Test System*, Networking Studies 2011 Selected Technical Reports, p.167-182, CESNET, May 2011, ISBN 978-80-904689-1-7

[5] D. ENDLER, M. COLLIER: *Hacking Exposed VoIP*, McGraw-Hill Osborne Media, 2009.

[6] N. PROVOS, T. HOLZ: *Virtual honeypots*, Addison-Wesley Professional, 2007.

[7] R.C. JOSHI, A. SARDANA: *Honeypots: A New Paradigm to Information Security*, Science Publishers, 2011.

[8] M. VOZNAK, F. REZAC, K. TOMALA: *SIP Penetration Test System*, TSP 2010: 33rd International Conference on Telecommunications and Signal Processing, pp. 504-508, 2010.

[9] M. VOZNAK, F. REZAC: *SPAM over Internet telephony*, in Proceedings of 11th International Conference on Research in Telecommunication Technology (RTT 2009), Srby, pp. 119-121, 2009.

[10] M. VOZNAK, J. SAFARIK: *SIP Proxy Robustness against DoS Attacks*, in Proceedings of the Applied Computing Conference 2011 (ACC '11), Angers, France, November 17-19, 2011. ISBN 978-1-61804-051-0

[11] J. SAFARIK, M. VOZNAK, F. REZAC: *Security Evaluation of Multimedia Systems*, TERENA Networking Conference TNC 2012, Reykjavik, Iceland, May 2012 In Proc. Networking to services, TNC2012, The 28th Trans European Research and Education Networking Conference, 21 - 24 May, 2012