Voice2Web security testing

Technical report

Jan Staněk
Research & Development center(RDC) for Mobile Applications
February 2009

Abstract

This document describes the vulnerability tests made on one of the servers of Voice2Web project. All the information about the motivation, test design, test results and proposed defense are gathered in this document along with the information about the current defense of the server. Because there is an infinite amount of possible attacks I decided to aim at the, from my point of view, most common types that propose the biggest security risk for the V2W environment and tried to counter them using the security solution installed in the previous part of V2W security project.

Table of Contents

Introduction

Voice2web project is one of RDC projects and it aims on voice applications. It gives the user the oportunity to build and test his own voice applications in an easy way using VXMLIDE application. The security part of the Voice2Web project is to secure the servers used in the project, making them as invulnerable as possible. In the previous stage, documented in the first report [1], first steps of tightening the security were made and some protection software was installed and configured. The second part of security subproject of Voice2Web project is described in this report. This part consists of practical testing of the defense abilities of the most vulnerable server used in the project – the Asterisk server. The tests were divided into two groups – general tests of overall defense abilities of the server and SIP specific tests aimed on VoIP traffic and Asterisk application. The results of the tests were analyzed and some new security arrangements were suggested. During the tests was also discovered one bug in used open source Snortsam plug-in – more information are provided later in this document.

Motivation

VXMLIDE is still in the development phase and is used only by a few people from RDC and some students of ČVUT. That was good for the early phase but it is supposed to be used more widely now and there are plans to open it for public access soon. One of the prerequisities for this opening is of course establishing good security because every publicly known server naturally becomes a target of hackers. Although the server is not publicly known yet, there is about one ssh dictionary brute-force attack every two days and it is probably going to be much worse. And that is the main reason to try to test and improve the servers defense before its purpose and IP address are revealed to the public. Every successful attempt to attack the server now can help us develop appropriate defense and precede the problems later. And that is really important because the failure of the server now means very little compared to the problems it might produce when the application is publicly used.

Short introduction of Asterisk server

Asterisk server is a common PC that serves as

- server running the VXMLIDE application for voice applications development
- IP telephone switch that enables the users to call their developed voice applications
- Trac (project management and bug/issue tracking system) for the Voice2Web project
- testing machine for other branches of the V2W project (for example DNS testing nowadays)

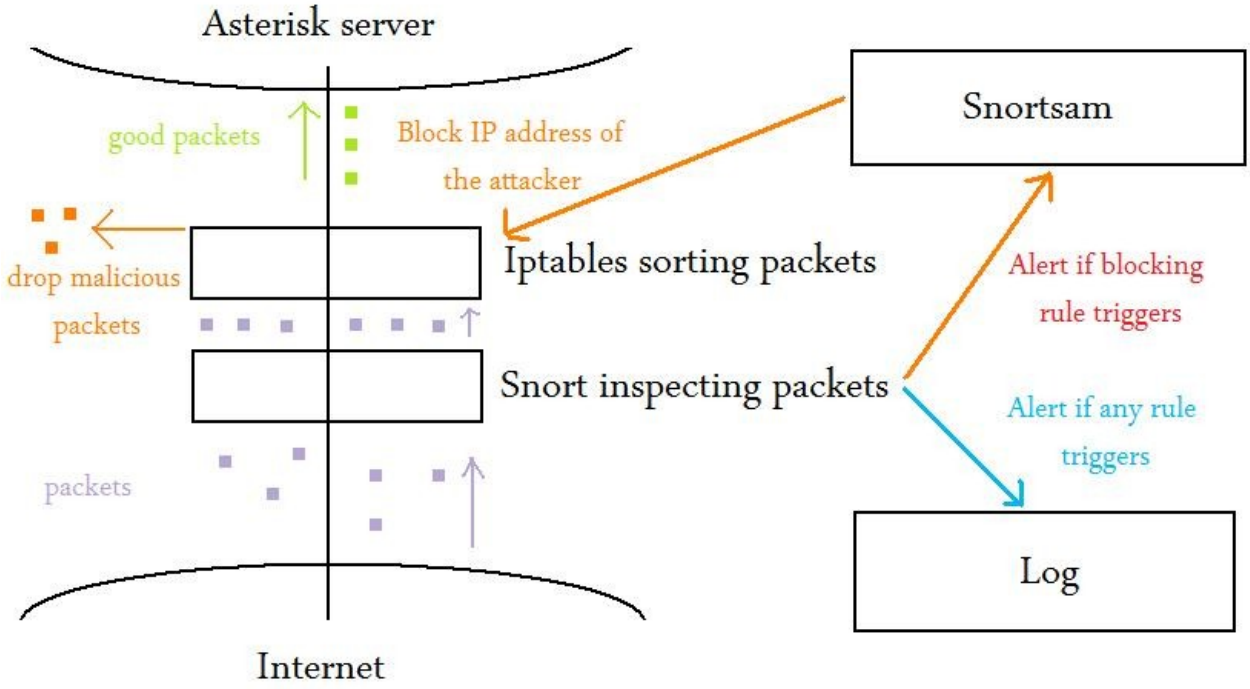
All these roles make the Asterisk server quite vulnerable, because it needs a lot of ports to be opened for the applications to be accessible.
The current defense consists of firewall iptables, IDS(intrusion detection system) Snort and IPS(intrusion prevention system) plug-in Snortsam. For the roles of these, we can say that iptables blocks the unused ports, Snort detects the attacks and Snortsam is able to block the attacker, if his attack is discovered by Snort. The configuration of these will be shown later in this document with the explanation why the configuration is the way it is.
The hardware of the Asterisk server is Intel Pentium 4 at 3.6Ghz with 2GB RAM running Fedora Core 6 operating system.

Firewall – iptables and its configuration

If you are not familiar with firewalls, reading [2] is recommended. If you know what a firewall is and what is its purpose, but are not familiar with iptables, reading [3](en) or [4](cz) is recommended before continuing in reading this document.
As mentioned earlier, iptables firewall is the basis of the Asterisk server security. Its purpose is to block unused ports e.g. to open the needed ports and block the others (obviously it is much easier to close everything and open just these you need instead of closing every unused from the 65536 ports;). The script containing the set up for iptables along with the description for each part of the script is enclosed in Appendix A, available only for the members of V2W project.

Snort + Snortsam – the IPS solution

Iptables, mentioned in the previous text, is the basis of the Asterisk server security but it is not enough to stop potential attackers. Iptables is a powerful tool but it is able only to divide traffic according to the source and destination addresses and ports and let the traffic pass or drop it according to the rules. It cannot inspect the data in the packet and it is vulnerable to spoofed addresses. Also if one uses many rules, it can easily become an unreadable mess. Because of this, Snort [5] was chosen as the next step in securing the server. Snort itself is able to check not only the ports and addresses as iptables do but it is also able to inspect the data inside the packet. Furthermore, Snort is really widely used and there are many rules already prepared and sorted for almost any type of attack. Why not use only Snort instead of iptables? Because Snort itself does not have the possibility to block the traffic. It can inspect it, alert the administrator, if something suspicious happens but it cannot block the traffic in any way. Because of that fact, Snortsam plug-in [6] is used together with Snort. Snortsam itself does not block the traffic, but it is able to forward the blocking rule to almost any type of firewall. It is starting to look a bit confusing so maybe a picture will show it better.
As we can see, the incoming packet is first inspected by Snort. Snort has two sets of rules – blocking rules and classic rules. A blocking rule is just a rule that also sends an alert to the blocking daemon which is in this case the Snortsam. If a rule is triggered (that means if the incoming packet matches the rule pattern) then this event is logged. If the rule was the blocking rule, the message to Snortsam is sent with the attackers IP address and ID of the rule. If Snortsam receives an alert, it prepares a blocking request for the iptables, containing the IP address of the attacker, which will be blocked completely (not only the incoming packets from the attacker but also the outgoing packets from server to the attacker, if he succeeded in infiltrating it beforehand). When the packet arrives at iptables, it is either passed in or dropped. and that is it.



Snortsam bug

Some problems with Snort unexpectedly crashing occurred during the testing phase of installed security solution. After some investigation I realized that there probably is a problem in the communication between Snort IDS and Snortsam plug-in. I traced the bug, made a description of it and sent it to the developers of Snortsam plug-in. I also created a work-around this bug so our security solution is safe but since the information about the bug might be exploitable, they are enclose in Appendix B, available only for the members of V2W project.

Security tests – part I. - general tests

Tests done in this part are aimed at the general defense abilities of the Asterisk server. Because there is an almost infinite amount of possible attack types I tried to choose the types that are most widely known, easily done with automated publicly available tools or that I already observed to be tried against the servers I used in the past by unknown attackers.

Portscan

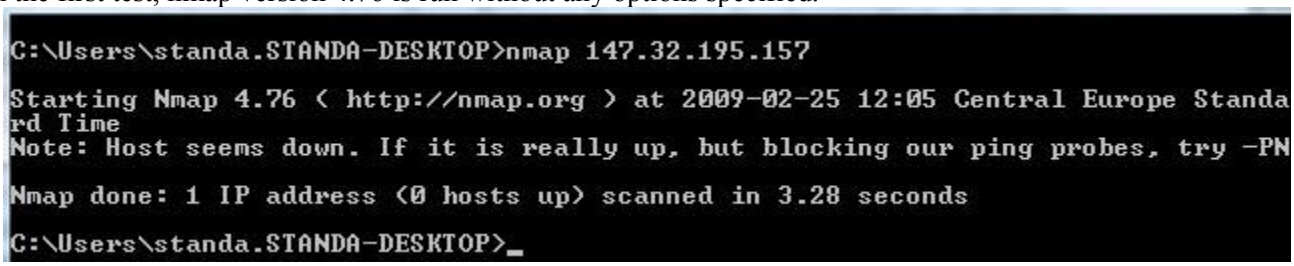
Although portscan is not an attack in particular, it is often a first step that the attacker does to check whether the target server is online and what ports are opened on it. There are many tools that provide automated portscan. These tools often have many usable options and are able to provide the attacker not only with the list of open ports but also with the derived information about the operating system that is probably running on the target server as well as with the list of applications running on target server often even with the version numbers. Once the attacker collects these information he can plan and target his attacks precisely according to the servers configuration. It is almost impossible to block the portscans absolutely but it might be very useful to at least halt them for a while. For the test were chosen three widely used automated tools – Nmap [9], SolarWinds Port Scanner [10] and Nessus [11].

Nmap portscan

Nmap is often the first choice tool for port scanning because it is most widely known, small, fast and one can get it in a few minutes without any registration or other annoying delays. Many linux distributions also already have nmap installed. It is a multi-platform tool and although it was originally console only many GUIs have been developed for every widely used graphical interface so almost every user is able to use it without problems. Nmap has pretty many options but it is often used as is in a most simple mode so I tried that first too. Note that nmap has also the options changing it to a state where it tries to determine the version of operating system, running applications etc. but it does not affect the results we are concerned of so I decided not to use them.

Nmap portscan – test 1

In the first test, nmap version 4.76 is run without any options specified.



From the screen above is obvious that this version of portscan was unsuccessful. The reason for that is indirectly our configuration of iptables. More information about the reason of this unsuccessful attempt are available in Appendix C / test 1.
Fine, the simplest version of portscan is unsuccessful but the reason is clear and the attacker can find it out as simply as us. Still some amateur attackers or automated attack scripts may fail on it. If we have a look at the results written by nmap, we can see that it itself notices that this behaviour (sending ICMP request and waiting for response) might be changed using -PN (some versions offer -P0 instead) option. Lets try that one.

Observation

There is a possibility that nmap without any option specified succeeds and gives us the same results as the next test. This behaviour was observed only in the situation when the scanned server and the machine running nmap are on the same private network. In that case nmap automatically skips the ICMP echo request.

Nmap portscan – test 2

In this test we use nmap version 4.76 with -PN option.

```
C:\Users\standa.STANDA-DESKTOP>nmap -PN 147.32.195.157

Starting Nmap 4.76 ( http://nmap.org ) at 2009-02-25 22:07 Central Europe Standa
rd Time
Interesting ports on bolek.feld.cvut.cz (147.32.195.157):
Not shown: [redacted] closed ports
PORT     STATE SERVICE
[redacted] open  [redacted]
           open
           open

Nmap done: 1 IP address (1 host up) scanned in [redacted] seconds
```

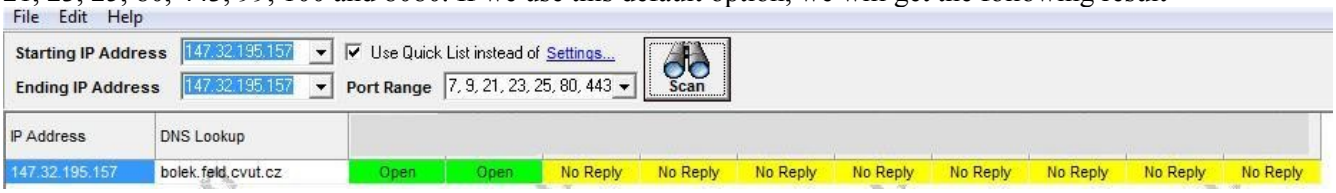
As we can see from the results, nmap was successful now. But is it really true? Lets have a look at the results when we switch our protection off.

```
C:\Users\standa.STANDA-DESKTOP>nmap -PN 147.32.195.157

Starting Nmap 4.76 ( http://nmap.org ) at 2009-02-25 22:01 Central Europe Standa
rd Time
Interesting ports on bolek.feld.cvut.cz (147.32.195.157):
Not shown: [redacted] closed ports, [redacted] filtered ports
PORT     STATE SERVICE
[redacted] open  [redacted]
           open
           open
           open
           open

Nmap done: 1 IP address (1 host up) scanned in [redacted] seconds
```

As we can see, the results differ quite a lot. The reason for this, information about the security approach to someone who tries this type of portscan against our server and uncensored version of screenshots are available in Appendix C / test 2. Complete list of all ports and their designed services is available at [12].

SolarWinds Port scanner portscan

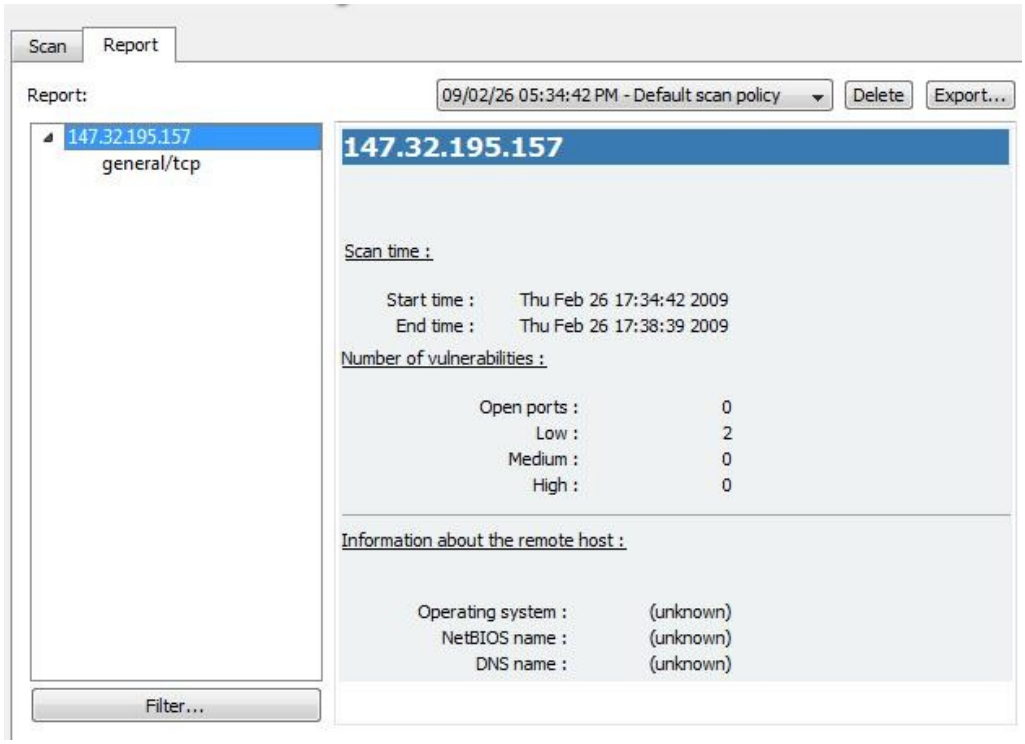
SolarWinds Port scanner is part of the SolarWinds Engineer's Toolset. It is a collection of network testing tools available after free registration at [10] for 30 days. There are three pre-defined port ranges to be probed and the user might also specify his own list of ports to be tested. The default is the first port range that tests ports 7, 9, 21, 23, 25, 80, 443, 99, 100 and 8080. If we use this default option, we will get the following result



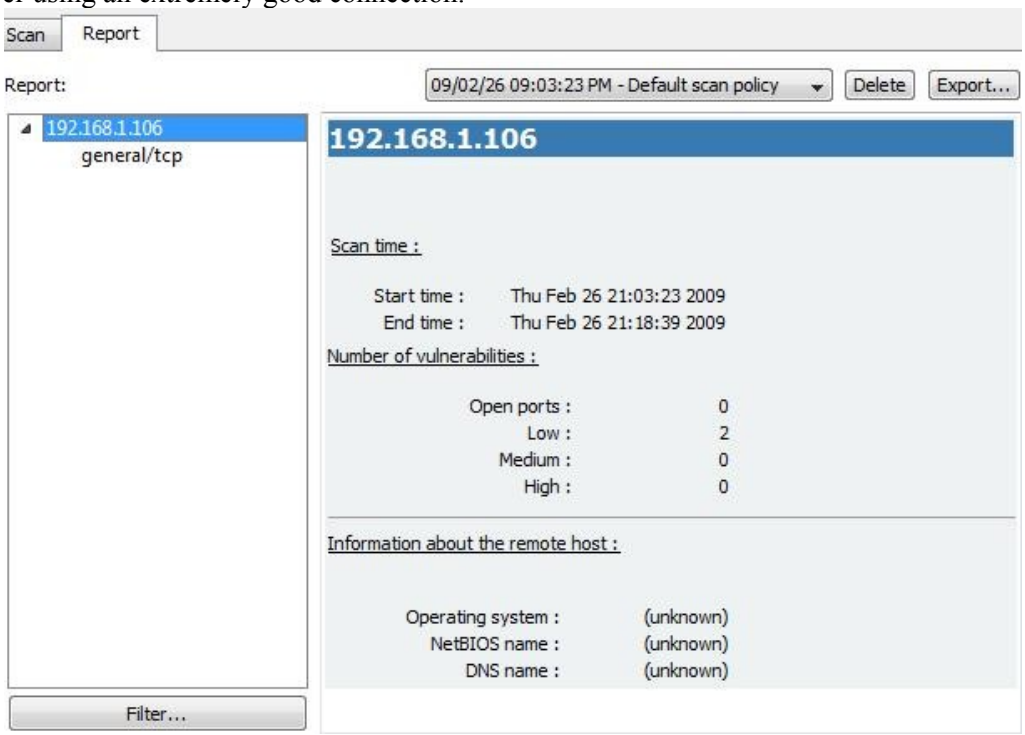
This result is not accurate, but for security reasons, I will not write down the numbers of ports that have / have not been discovered. These information are available in Appendix C / SolarWinds Port scanner.

Nessus portscan

Nessus is not just a port scanner but a vulnerability scanner. It tries to find opened ports, determine which application is running on them and if it finds out, it also tries some of its in-build exploits to check the security flaws of the running application. It is the second most used after nmap and its popularity comes from the fact that it offers plenty of additional information about discovered security flaws (attackers really appreciate that:) and recommendations how to prevent the attacker from using them (administrators really appreciate that:). Lets see the results of the newest Nessus v 3.2.1.1 on our server.



As we can see, the situation is similar to the previous scanners. Nessus found just two open ports and its results are obviously incomplete. If we try to figure out why that happened, we might be surprised that it is not because of our security solution but mainly because of the unreliable network. The rate of lost packets plus the dropping policy of iptables make the Nessus scanner stop prematurely providing no useful information. I have to add that the authors of Nessus are not to blame because they point out in the documentation that Nessus is only for private use in private networks. The quality of my home Internet connection might have influenced the results too so I decided to run the scanner on my personal home server for testing purposes. This testing server is very similar to the Asterisk server so the results should be similar to these you can get if you use Nessus against the Asterisk server using an extremely good connection.



Now this is a bit similar to the previous one, isn't it? The result is the same but the reason is different. More information about it can be found in Appendix C / Nessus.

Portscan summary

The current security solution is able to log every portscan attempt and block a fair amount of typical portscans. There still remain some ways to use a smart portscan technique that cannot be blocked by actual security solution but that is probably unchangeable. Since every portscan attempt is logged by Snort the security may be easily improved by modifying the ruleset according to the results of log analysis in the future.

SSH Brute force attack

One of the most widely used attacks. The attacker continuously contacts the ssh port of the server (22) with the request to create a ssh connection with different usernames and passwords trying to enumerate user accounts on the target server and to guess the passwords for these accounts. These attacks are often based upon a known dictionary of possible usernames and passwords and therefore there are many variants – from somewhat silly exhaustive ones using every known word in a role of username and password (that means trillions of variants) to the precisely planned attacks where the username list is derived from the information gained from open or stolen sources (many users use their name, surname, email etc. as a username) and the password list is created accordingly (users hobbies, family members, phone numbers etc.). There are plenty automated tools used for SSH brute force attacks but the most widely used and accepted is THC Hydra [13] and this tool will also be used in the following tests (Hydra version 5.4).

SSH Brute force attack – root is the target

The first variant of SSH brute force attack is trying to avoid the necessity of enumerating user accounts on the target machine assuming that there must be an administrator/root account. These attacks often use one of the following usernames and try just different passwords for the one chosen username. The usernames are root, admin and administrator.
Because there is no admin nor administrator account on the Asterisk server I will not try these because the attacker will get nothing using them (but he is free to try of course:). Quite different is the root account variant because there always is a root account on a linux server and our Asterisk server is no exception. Lets try that one and see what we can get. Note that I will first simply try to connect to the Asterisk server once with the root username and random password. There is no need to be using the Hydra tool yet as can be seen in a moment.

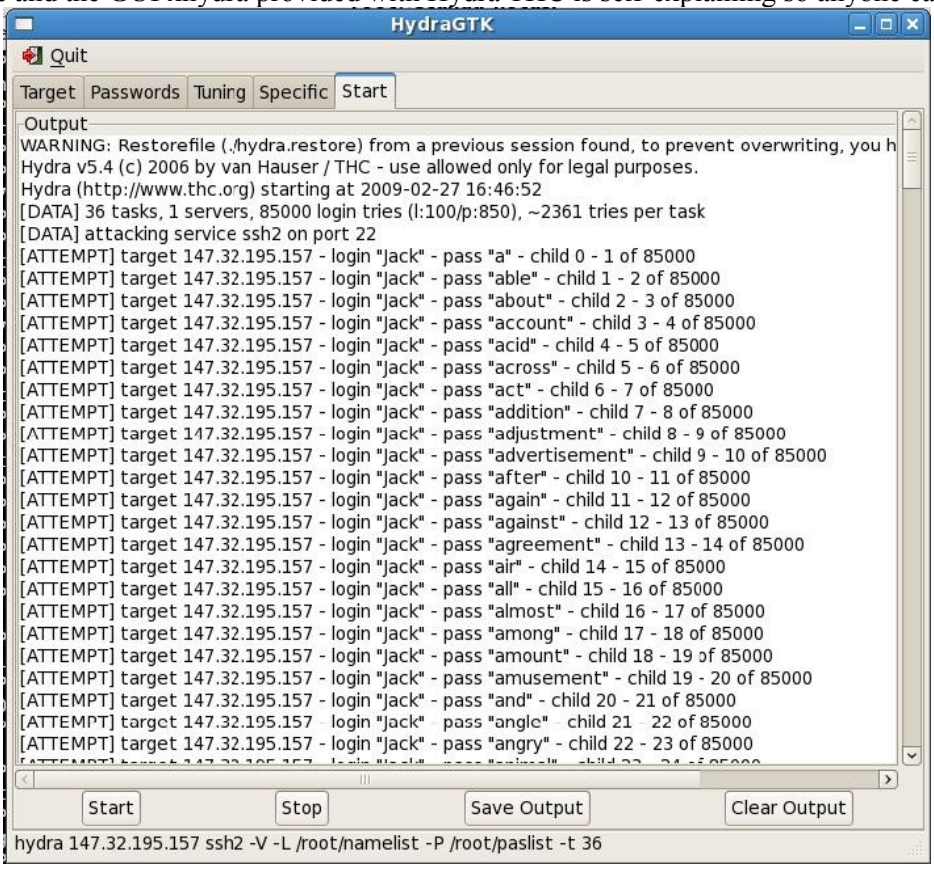
As we can see, I was not successful and I did not manage to guess roots password. Is that so? Well, yes, but nothing will change even if I used the roots real password. Lets have a look at the tail of the security log of the Asterisk machine (it can be found in /var/log/secure and is readable only with roots permissions)

```
22:18:52 bolek sshd[521]: User root from r4a126.net.upc.cz not allowed because not listed in AllowUsers
22:18:52 bolek sshd[522]: input_userauth_request: invalid user root
22:18:55 bolek sshd[521]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser=
22:18:57 bolek sshd[521]: Failed password for invalid user root from 84.42.165.26 port 12048 ssh2
```

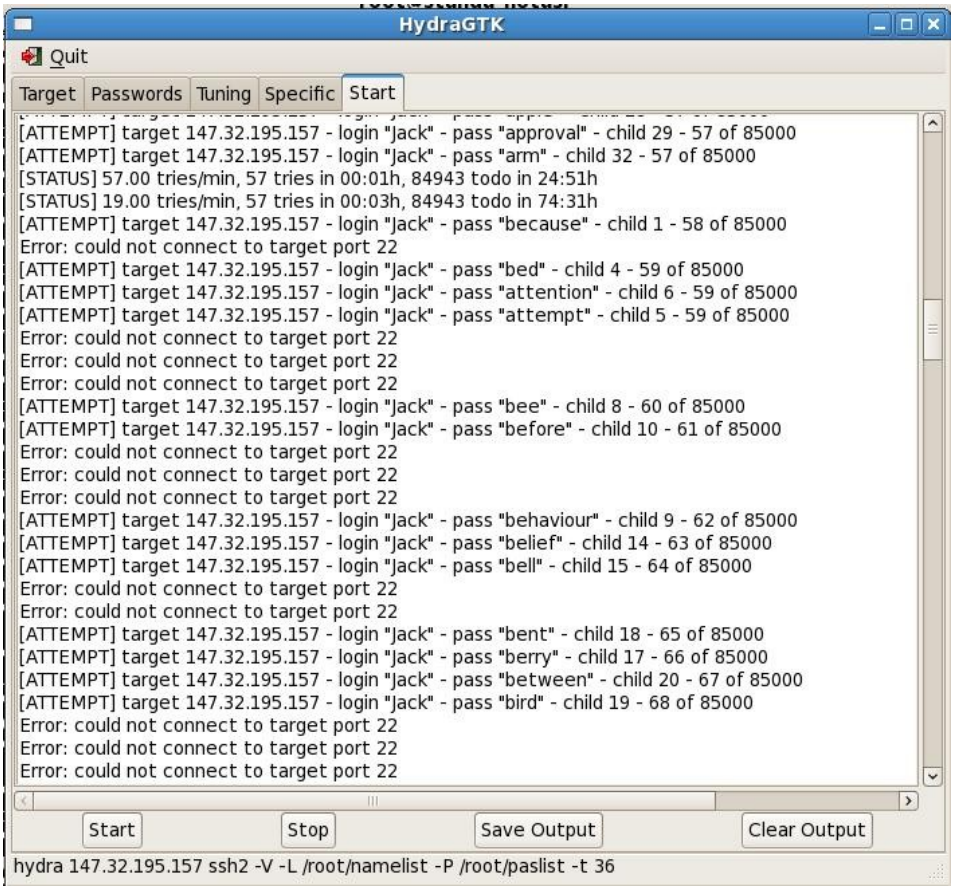
User root is not allowed to log remotely. That is the reason why my naive attempt to log as root did not succeed and it is even the reason why the attacker might try all the passwords he wishes for the root account and will not succeed. If we want to find the source of this behaviour we have to have a look at /etc/ssh/sshd_config file. There is a line beginning with AllowUsers and only users listed on this line are able to login remotely to the server using ssh. Since the root account is not listed, root cannot login remotely using ssh and the only way is to log as an allowed user and then use the su command to switch to user account. So we can say that the Asterisk server is fully-protected against this type of ssh brute force attack and continue to the other variants.

SSH brute force attack – open dictionary

This variant of ssh brute force attack is based upon open dictionaries which are to be found all over the Internet. As a username list I will use for example the list of most common boy baby names in the UK prepared in [14] and as a password list the 850 basic english words from [15]. This is just a very simple example because if one has a couple of minutes for browsing then there are plenty of username lists and password lists prepared especially for this purpose on the Internet. But this is not supposed to be an attackers how-to manual, is it?;) Once I have the lists prepared and Hydra THC installed there is no obstacle in my way to test this attack. Using Hydra is intuitive and the GUI xhydra provided with Hydra THC is self-explaining so anyone can use it easily.



As we can see, Hydra began the attack without hesitation – there were 100 usernames and 850 passwords which means 85000 login attempts. According to the screen everything seems to be fine, but if we scroll down a bit.



Obviously there are some problems. The first status – 57 tries in 1 minute is not much but acceptable, but the same 57 tries in 3 minutes indicate that something does not work the way it should. More information about the reason of this behaviour and the consequences for the attacker can be found in Appendix D.

SSH brute force attack – public information based

This is the most dangerous type of ssh brute force attack because, if prepared properly, one does not need to try so many times and has a higher possibility of success. It is very similar to the previous type but the username and password lists are based upon openly accessible information about the people using the server. Where to get these information? On the Internet of course. If we take our Asterisk server as an example - if we know its IP address and its purpose, we probably know that it is a part of Voice2Web project at RDC (since this information is not secret and it will be even publicly announced once the Voice2Web project is opened for public use). Finding the RDC homepage [17] is a matter of few seconds and few more second to find the part about Voice2Web project [18]. And what do we have here? A list of persons currently working on the project. That means we just acquired a list of possible usernames quite fast and easily and the probability that there will be an account with the name or surname of at least one of these persons is really high. And the password list? Once we know the people using the server we can make a list of possible passwords from their home pages, community pages, accounts in various Internet applications etc. That does not guarantee anything since the passwords used by the users might have nothing in common with their casual live but... I will not continue by any practical example since I do not want to accidentally reveal my colleagues passwords (since I can get these in an easier way being the administrator of the Asterisk server:). But I had to write down this short article so this type of attack will not be overseen in the future because it is practically unblockable by software or hardware protection. On he other hand, careful and intelligent users using strong passwords are more then enough protection.

SSH brute force attack summary

The current security solution is able to block any attempt of breaking roots account password remotely. It is also able to rate-limit any ssh brute force attack making automated tools with default configuration generate long error logs and fail. There still remains the possibility that the attacker reconfigures the tool to generate the tries at a permitted rate but that mean just so few tries per day and every one of these is logged so the administrator has a high chance of discovering the attacker before the attacker gets a working pair of username and password.

SNMP brute force attack

This attack is one of the less used and its target are often weakly-defensed hardware parts of network infrastructure. More about SNMP is to be found at [19]. Using this attack against a firewall-protected server is the same as speaking to deaf ear because the target ports (161 and 162) are usually blocked but there are some good automated tools to perform this attack globally at the whole network and so it might be useful to find out that something like that is happening in our network even if it does not harm our server. The most widely used specialized tool is probably the PROTOS [20] but many other tools have this functionality – like Hydra THC we used earlier or SNMP Brute Force Attack which is a part of SolarWinds Engineer's Toolset. Trying this attack at current Asterisk server configuration would not get us anywhere because it will be blocked by the firewall and it will not be logged. I prepared a simple Snort rule which can be used to detect this attack

```
alert udp any any -> 147.32.195.157 161 (msg:"probable SNMP attack,161/UDP
";sid:1101001;rev:1)
```

After I wrote this rule down I wanted to test it on the Asterisk server (on my home test server it worked fine and generated many alerts when I tried to use SNMP attack using SolarWinds tool). The result was surprising – no alert at all. Afterwards I used Hydra and again, no alert was generated. I inspected the reason and found out that someone, probably the administrators of ČVUT network, prohibited packets targeting port 161 from entering their network. Well, the result is that the Asterisk server and surrounding network is fully protected against SNMP brute force attack. If the general block of packets for port 161 are ever removed, the proposed Snort rule might be used to detect the probably malicious SNMP traffic.

DoS and DDoS attacks

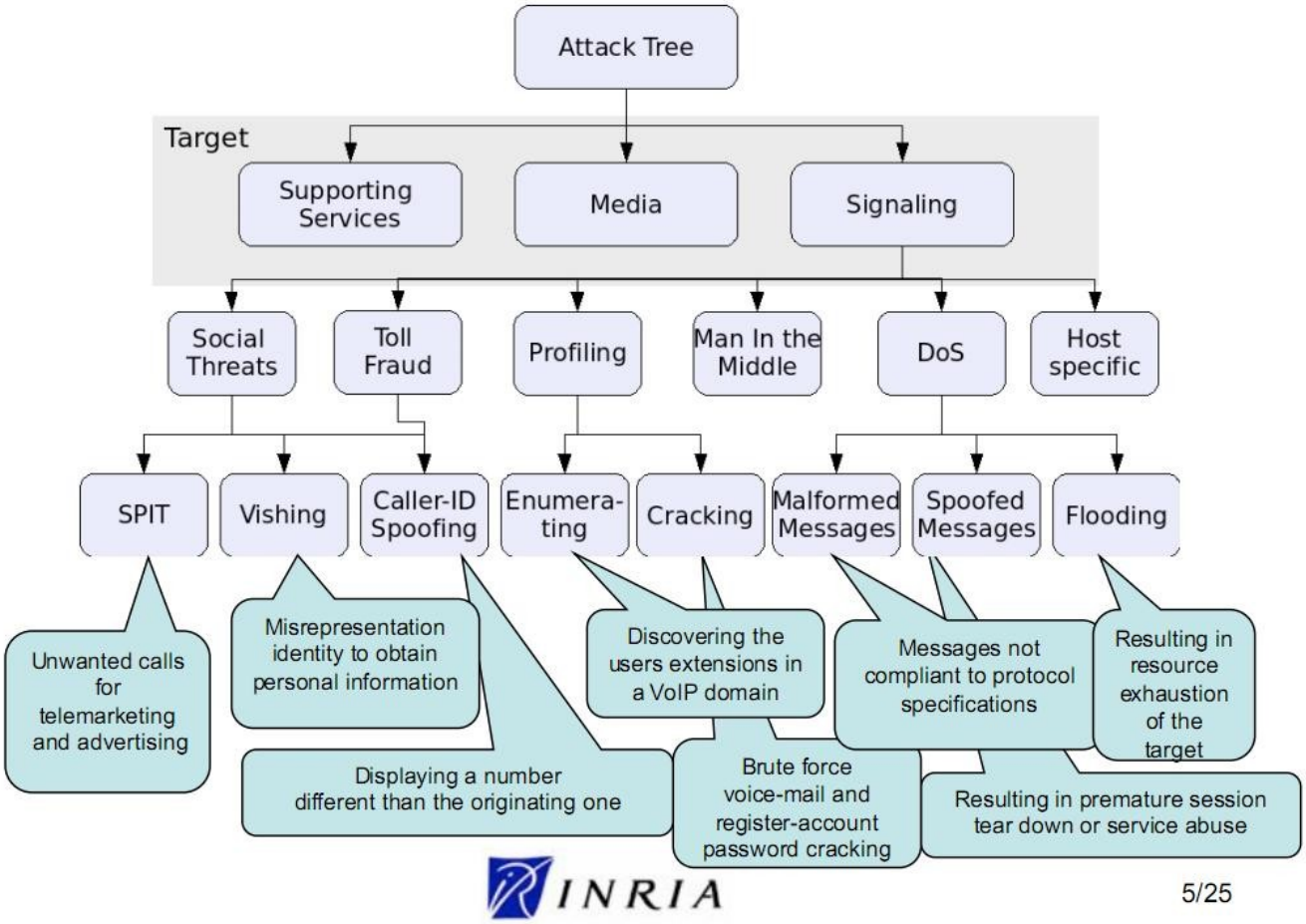
This big group of attacks consists of plenty of various attack types. The basic principle and classes of attacks are described in [21]. The first and possibly the biggest group is formed by flood type DoS and DDoS attacks. The second group is formed by malformed packet attacks. Although this division to two big groups is unofficial and these groups overlap greatly, I decided to use it because there is still a big difference between these groups. From this point of view the flood type attacks are these using a huge amount of packets that blocks the servers resources by forcing the server to react to them somehow and the malformed packet type of attacks that uses packets with spoofed ports and addresses and unusual payload trying to stop the servers services by producing an unexpected situation that leads to service or even the operating system crash.
Because the information about DoS and DDoS attack tests are very sensitive and full of detailed information about our security solution set up, all the information about DoS and DDoS attacks were moved to the Appendix E.

Security tests - part II. - SIP specific attacks

Tests in this part are aimed at the SIP protocol [25] and its implementation in the currently used Asterisk PBX application [26]. Defense against the denial of service attacks aimed at this application is very important because the Asterisk server is primarily dedicated to be the phone switch providing the users with the possibility to contact the voice applications developed in the Voice2Web project. Therefore if this application fails, users will not be able to contact any voice application which may lead them to abandon the Voice2Web project. That will be crucial for the project which means the possibility of that happening must be lowered as much as possible.

Common threats in VoIP

The following picture was prepared by Inria institute for research in computer and science [32] and shows the attack tree of general threats to VoIP [27].



We can see that there are many different types of attacks on the field of VoIP. Thankfully not all of these are quite relevant in the Voice2Web project. Lets analyze these types from the leftmost one in the picture and evaluate the risk level of the attack type in the environment of the Voice2Web project.

SPIT - stands for Spam over Internet Telephony. This type of attack tries at first to enumerate the users of the target corporation using VoIP and then automatically calls them offering unwanted products or services. This type of attack is a low risk one in our project because there are no official possibilities for the users to call each other. The users are supposed to call only the created applications. Because there is an established connection between the mobile network and the Asterisk server, there is a possibility to misuse this connection and spread the spam to the mobile network but that would require the attacker to take over the server and reconfigure it at first and that is other type of attack, already mentioned earlier.

Vishing [33] - another low risk threat in our project environment. Since our user accounts do not contain any important secret information and the voice applications have testing purpose and therefore should not contain any of these information either, the attacker should not get the information he seeks even if he succeeds in breaking users account or impersonates another user. Maybe we should attach a warning to the project page so the users are warned explicitly not to put any personal or secret information into their applications.

Caller-ID spoofing - not interesting in our project environment. We plan that anyone interested in building his own voice application will get a free account and there will be no gain in pretending to be another user when calling the applications.

Enumerating - another low risk threat. The numbers of voice applications are publicly accessible.

Cracking - low to medium risk threat. On one hand the voice-mails are not used on the other hand the users account to the vxmlide application grants the rights to change their applications at will. This threat is not important in the VoIP field but should be considered carefully for the user accounts to the vxmlide application.

Malformed messages - medium risk threat. Malformed messages that are dropped by the Asterisk application propose no threat. Malformed messages that invoke a false call that occupies resources are a low threat in case the amount of concurrent generated false calls from one source can be observed. Malformed messages killing the Asterisk application or invoking an unexpected behaviour are high level threats. Thankfully none of the latter type were observed on the Asterisk server yet.
Spoofed messages - medium-to-high risk threat. The possibilities of prematurely ending users session with the voice application or preventing the user from contacting voice applications are crucial.
Flooding - high risk threat. Flooding the Asterisk application or the Asterisk server itself results in unavailability of Voice2Web project functionality.
The result of the analysis shows that the most important is the defense against malformed and spoofed messages and flooding. I will focus on these in the following part. The first step will be using automated vulnerability scanners to stress-test the Asterisk application and possibly find its vulnerabilities.

SIP vulnerability test tools

The SIP protocol is very popular solution on the field of VoIP in the last ten years and there are plenty of implementations of it. The developers working on these implementations tested their own creations but there was a growing call for an automated tool able to test any SIP implementation and find its weak spots so the people can choose the best solution to use in their business. That lead to the creation of quite a few groups who began to work on a tool like that and many tools were really finished. The most successful tools were the ones providing the possibility of testing many aspects of the SIP protocol along with the possibility to add new test cases easily. Together with these tools were often written scientific and technical reports about the theoretic part connected to the SIP protocol security and the tool itself. I tried to choose the best candidates from these sources and used them against the Asterisk server to see which security flaws will be discovered so I can eventually prevent the attackers from using them.

SiVuS

SiVuS stands for SIP Vulnerability Scanner. This tool was developed by Voice over Packet Security Forum group and is available for download after free registration at their homepage [28]. I used the current version 1.10 for this test. It is one of the most widely used tools because it offers thousands of test cases that are tried in an exhaustive continuous test, it is multi-platform and has a simple GUI providing plenty of additional settings. After the test is finished it also offers transparent report in html format instead of the other tools that often offer just the passed/failed result. But not everything is gold that shines. As I was astonished by the possibilities SiVuS offers it gave me a really tough time in the end. Lets see the results of our Asterisk server.

Summary of Findings

Risk Level	Number of Findings
High	1
Medium	0
Low	1424
Informational	0
Passed	2170
Total Number of checks.	3480

The test configuration was to test the udp traffic only and to use all the test-cases available. I tried the tcp and tls settings too but it did not work for an unknown reason. SiVuS just reported that it is not able to contact the server using nor tcp nor tls (but common softphones with the tcp support works just fine). No matter, lets have a look at the udp results. The result seems that there is one major problem and a good bunch of not so important ones. Lets have a look at the problem listed as having high risk level.

147.32.195.157(5060/UDP)	[High] : Check No [13100.1] UDP
Description	This check verifies the ability of the UA to authenticate INVITE requests.

According to the test result, SiVuS reports that our Asterisk application is not set to authenticate INVITE requests. After some browsing in Asterisk documentation I found out that this behaviour can be configured setting the insecure option in the sip.conf configuration file to yes or very. The information about the account I reconfigured and used in the testing along with the packet analysis of the situation is available in Appendix F. By the analysis of the packets is proved that the report of the SiVuS scanner was inaccurate and our server requires the authentication of an INVITE request before proceeding to the call. This false positive finding might also have a connection to the problems with result inaccuracy explained later.

If we go through the rest of the report we will discover that all of the low ranked findings are of the same type - it is always a sign that the server did not manage to process a malformed packet containing a defined amount of different characters in an unexpected place. When I was reading through these findings I was surprised that there were a bit weird results. An example of these can be seen in the following picture.

[Passed] : Check No [10101.30] UDP
This check verifies the ability of the UA to handle 50 of space characters between the method field and the
The remote UA handled this check appropriately. This configuration is desired and it should be maitained.
[Low] : Check No [10101.31] UDP
This check verifies the ability of the UA to handle 100 of space characters between the method field and th
It appears that the target UA could not handle SIP (over UDP) requests of 100 space characters between the UA can accept malicious requests that contain 100 space characters between the ACK URI and the (no SIP response was received) during this test, but after verification the process is still running. This r process to close the remote connection prematurely. Remote host did not respond (NETERROR)
[Passed] : Check No [10101.32] UDP
This check verifies the ability of the UA to handle 500 of space characters between the method field and th
The remote UA handled this check appropriately. This configuration is desired and it should be maitained.

According to the result, the Asterisk application is able to process a message that contains 50 and 500 space characters between the method field and a version in a SIP message, but is not able to process the same message containing 100 space characters in the same place. That looked strange so I decided to try sending the packet manually, using Nemesis packet injector.

Everything seemed to be fine, the response from the server captured by Wireshark was the same as the one received in the passed cases. That only confused me more and I decided to run the scan again. And then again a few times. All the results indicated above 1000 low findings but these were not in the same cases e.g. the malformed message that passed in one test failed in the other and vice versa. I analyzed the results and came to the conclusion that this behaviour is not caused by the fact that the server is not able to process the malformed requests. If you look again at the picture above, you can see that the low finding contains the reason (NETERROR) at the end. This indicates that the probable reason of the test-case failure might be a packet that got accidentally lost on its way through the network. I analyzed the other low findings and found one different reason (Caused by Socket timedout). I tried to run the scanner on my private network against my home test server to prevent the packets from losing. Yet the results were very similar to these gotten earlier. I used Asterisk management console to see what is going on on the server and finally found the true reason. The scan generated about 100 concurrent requests all the time and it was about 450 requests in the peaks. The Asterisk application was not able to handle so many requests concurrently so some of these were postponed. This behaviour made SiVuS mark the test-case as failed and proceed to the next one. The Asterisk server on the other hand did not know that the session ended for the scanner and it tried to contact it when the request was processed. Because the scanner did not respond to the Asterisk messages (since it already marked the test-case as failed and did not wait for it) the Asterisk tried to resend its message for a defined number of times for retries and then ended with the following warning.

```
Maximum retries exceeded on transmission
ZTNMEOVEE6oVY3S0rMaf@147.32.195.157 for seqno 71000 (Critical Response) --
See doc/sip-retransmit.txt.
```

I concluded that there is no malformed packet in the test that is really dangerous for the Asterisk application

because the application did not crash during the test nor it stopped responding. I tried to make a legitimate call during the test a few times and I always succeeded - there was sometimes a small delay at the beginning of the call but otherwise the call went smoothly to the end.
The outcome of this testing using SiVuS was not what I expected. I hoped to get the list of malformed packets that cannot be processed by the Asterisk application and are eventually able to kill it or make it run unexpectedly so I can make new Snort rules to block these packets before delivery and tighten the security of the server this way, but I ended with many different lists with different but inaccurate findings. I did not tested all the different findings manually because it would take too much time and I found it unnecessary in the end. It is possible that in the test-cases there are some malformed messages that cannot be processed by the Asterisk application but since the application was fine after the test ended and all the channels were freed automatically a few seconds after the test finished, these messages might be able to block some resources for a short time at most and that does not posses a high threat. It showed up that the malformed messages will probably not be as big problem as it looked at first and that the biggest problem will be the flooding e.g. too many concurrent requests. To prove this conclusion to be right I used another vulnerability scanner based on the same principle as SiVuS - continuous flow of malformed messages trying to take the application down.

c07-sip

This awkward name is a name for the test suite designed to test the SIP protocol as a part of the PROTOS project [29]. All the information about this test suite are available at its homepage [30]. The test suite is available in two formats - standalone test-cases in a binary format that must be delivered to the target by a third party application and in a jar format that can be easily run from a shell/commandline if one has Java installed. I decided to use the jar format as it is easier to run and there should be no difference in the test results. I tried to run the test from my home but it had some difficulty caused by the network so I used one of the servers in the local network of the Asterisk server.

```
[root@obelix ~]# java -jar c07-sip-r2.jar -touri 1001@147.32.195.157 -fromuri st
anda@147.32.195.157 -teardown -validcase
single-valued 'java.class.path', using it's value for jar file name
reading data from jar file: c07-sip-r2.jar
Sending Test-Case #0
    test-case #0, 481 bytes
   Received Returncode: 401
 Sending CANCEL
    test-case #0, 246 bytes
   Received Returncode: 487
   Received Returncode: 200
 Sending ACK
    test-case #0, 240 bytes
Sending valid-case
```

In the picture above is a snapshot of successfully beginning test. If the test begins writing Timeout messages before the valid-case message it is probably caused by the network between your machine and the server - in my case the problem was my home router as I found out later. If the test successfully pasts the zeroth test-case but begins to write the Timeout message after some latter test-case it means that very probably the target application was killed/stopped by the last test-case. Contrary to the SiVuS, this tool does not provide any report at all. If the test runs to the last test-case without ending in a loop of Unreachable messages then the target application survived all attempts made by this tool and is invulnerable using the flaws tested by this tool. There is also one error that can make the test stop prematurely. An example of this error can be seen in the following picture.

```
Sending CANCEL
   test-case #16, 252 bytes
  Received Returncode: 487
  Received Returncode: 200
Sending ACK
   test-case #16, 246 bytes
java.lang.RuntimeException: Internal error, invalid test case file 'testcases/0
00017'
   at FI.protos.ouspg.wrapper.BugCatZero.parseJarFile(BugCatZero.java:483)
   at FI.protos.ouspg.wrapper.BugCatZero.parseTestCases(BugCatZero.java:334)
   at FI.protos.ouspg.wrapper.BugCatZero.run(BugCatZero.java:306)
   at FI.protos.ouspg.wrapper.SIPBugCat.main(SIPBugCat.java:380)
```

This does not mean that the test-case 16 failed but that there is a problem with using one of system variables by the test tool. Changing system variable LANG to c should solve the problem.

```
[root@obelix ~]# LANG=c
[root@obelix ~]# echo $LANG
c
```

After this small change the test ran through all of the 4527 test-cases without stopping in a loop or killing the Asterisk application which means that the implementation of SIP in this version of Asterisk is quite resistant. I tried to make some classic test calls while the test was running and I experienced some problems - approximately one of five calls failed. I looked to the Asterisk log and found out that the reason is that all the RTP ports were currently used (nowadays the number of concurrent RTP streams e.g. calls is limited). Because of the huge number of calls created by the tool during some parts of its run, all the RTP ports were taken and every new call was rejected. This behaviour is not a problem of SIP implementation but it points to the possibility of flooding the resources (in this case the RTP ports). That might be a possibility for a DoS attack and will be considered later in this report. After this I temporarily increased the number of allowed concurrent RTP streams and reran the test to see whether it was not influenced by this limitation. The test again ran through all the test-cases without problems.
Once again the result shows that malformed messages are less dangerous for the Asterisk application than the flooding and so I will address the flooding problem next and try to propose a decent defense against it.

Flooding

The results of previous tests concludes that there are two dangerous types of VoIP connected flooding attacks that might affect our recent configuration of the Asterisk server. The first type is the SIP flooding e.g. flooding the Asterisk application itself with numerous SIP request messages (even malformed) and the second is call flooding e.g. evoking more then the defined threshold of concurrent calls that results in the fact that all currently opened ports for RTP streams are used and no new calls can be established. The information about the second type along with the information about set up security protection against these attacks is available in Appendix G. The conclusion is that the Asterisk server is now well-protected against these attacks.

Conclusion

After the tests are finished I can say that the Asterisk server is now protected against many of the typical attacks that were tested. DoS flooding at some ports still remain a problem but I hope that CAMNEP will aid us with this weakness. Of course there still remain plenty of attacks that were not tested in this phase and further testing in the security field is recommended. I hope that thanks to my somewhat original approach to block the portscans the attackers trying to scan the server will be discouraged. The next steps of security in the Voice2Web project will be aimed on the cooperation with the Faculty of Cybernetics ČVUT and their CAMNEP IDS, the DoS flooding attacks will be inspected more carefully and maybe we will finally find the phone able to communicate with the Asterisk application with the TLS support.

Resources

[1] Staněk, J., Rudínský, J. - Secure Voice2Web servers from misuse, technical report, Prague, 2008
[2] http://en.wikipedia.org/wiki/Firewall_(networking)
[3] http://www.root.cz/serialy/vse-o-iptables/
[4] http://iptables-tutorial.frozentux.net/iptables-tutorial.html
[5] http://www.snort.org/
[6] http://www.snortsam.net/
[7] http://www.snortsam.net/maillist.html
[8] http://www.snortsam.net/files/snort-2.8-plugin/snortsam-2.8.3.diff
[9] http://nmap.org/
[10] http://www.solarwinds.com/downloads/index.aspx
[11] http://www.nessus.org/nessus/
[12] http://www.iana.org/assignments/port-numbers
[13] http://freeworld.thc.org/thc-hydra/
[14] http://www.babynames.org.uk/top-100-baby-names-uk.htm
[15] http://www.langmaker.com/wordlist/basiclex.htm
[16] http://forums.theplanet.com/lofiversion/index.php/t57628.html
[17] http://www.rdc.cz
[18] http://www.rdc.cz/cz/projects/Voice2Web/
[19] http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol
[20] http://www.ee.oulu.fi/research/ouspg/protos/testing/c06/snmpv1/
[21] http://en.wikipedia.org/wiki/Denial-of-service_attack
[22] http://agents.felk.cvut.cz/projects/camnep/
[23] http://www.packetfactory.net/projects/nemesis/
[24] http://www.hping.org/
[25] http://en.wikipedia.org/wiki/Session_Initiation_Protocol
[26] http://www.asterisk.org/
[27] http://en.wikipedia.org/wiki/Voice_over_IP
[28] http://www.vopsecurity.org/index.php
[29] http://www.ee.oulu.fi/research/ouspg/protos/
[30] http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/
[31] http://www.wireshark.org/
[32] http://www.inria.fr/index.en.html
[33] http://en.wikipedia.org/wiki/Vishing
[34] http://cvs.snort.org/viewcvs.cgi/snort/doc/README.thresholding?rev=1.5
[35] http://sipp.sourceforge.net/