Secure Voice2Web servers from misuse

Technical report

Jan Staněk, Jan Rudínský

Research & Development center(RDC) for Mobile Applications
Czech Technical University in Prague
Technická 2, 166 27 Prague 6, Czech Republic

December 2008

Abstract:

This document describes the first steps in protecting the Voice2Web project network against possible attacks incoming from the Internet. All V2W servers have had public IP addresses until now so the first task is to migrate them to a private network, so they will be unaccessible directly from the Internet. There has also been written a report about the possibility of using TLS support in the Asterisk PBX application earlier but the first tests were unsuccessful so the second task is to try to set up TLS support in Asterisk PBX properly. Because one of the servers has to stay connected directly to the Internet so it can serve as a gateway for the rest of the private network and maintain the functionality which requires direct Internet connection, the final task is to analyze the possibilities of open source IPS and find a security solution to be used on this vulnerable server.

Table of Contents

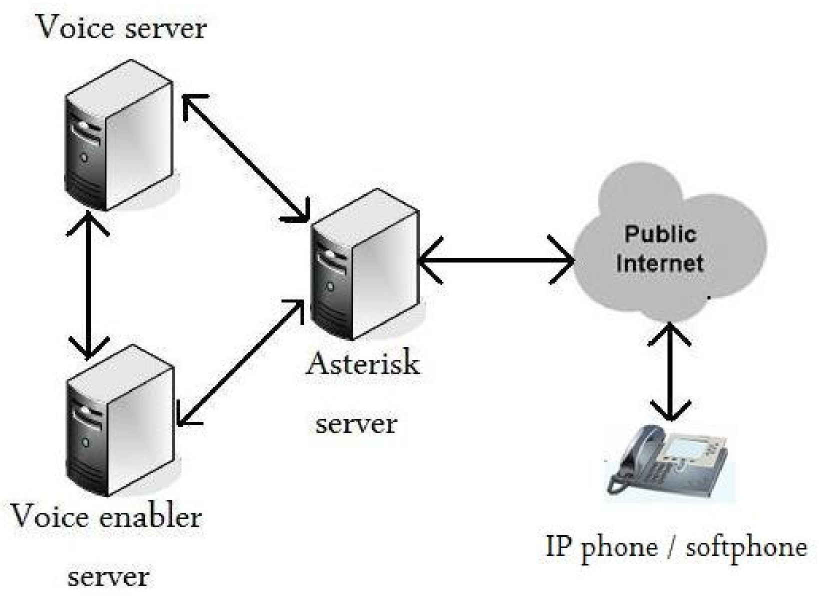
Acknowledgement

We wish to thank IBM Research for generous support of the project and Jan Kleindienst, Borivoj Tydlitat and Jan Sedivy for many thoughtful suggestions.
We also thank CESNET for the generous lending of project equipment and Vodafone Foundation Czech Republic for the kind student scholarship support.

Introduction

Voice2web project is one of RDC projects and it aims on voice applications. It gives the user the opportunity 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. Until now the servers were almost unprotected, can be reached directly from the Internet and there was no protection against possible attacks. To change this situaction we decided to migrate V2W servers into a private network, leaving only one server on the border, directly accessible from the Internet, for there are some applications that need direct Internet connection (the most important one is Asterisk PBX used for call switching). This step should be enough to protect the servers inside the private network because the attacker will have no way to access them unless he takes over the server left on the border. Because of the main functionality, let's call this bordering server the Asterisk server. It is obvious that the Asterisk server remains the weak point in the new network architecture and it is very important to protect it as good as possible. To create this protection we analyzed some open IPS softwares and decided to use the combination of Snort IDS + Snortsam plugin + iptables firewall. We also decided to try to set up the proper TLS support in the Asterisk PBX application so the call signalization itself will be secure and the attacker will not be able to use captured packets from the call in any form of repeat or disguise attack.

Voice2Web servers

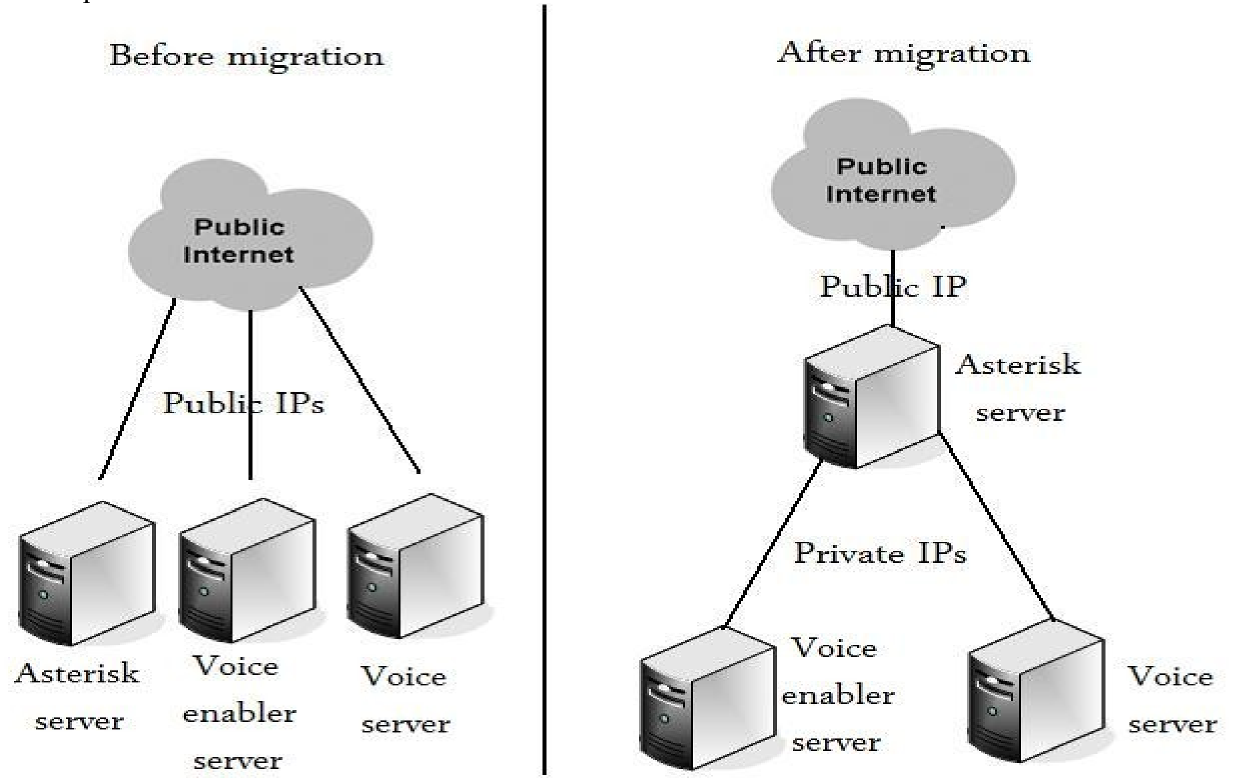


Nowadays there are three servers, each one with its specific functionality. Asterisk server takes care of call initiation and management, voice enabler server and voice server takes care of the information management (e.g. voice recognition and synthesis etc.). The communication between servers and outside world can be seen in the next picture. The Asterisk server is also connected to the mobile network via SS7, which is not in the picture because it is not important for this work.
The short description of servers and their interoperability is documented in the next table.

Server	Hostname	Cummunicates with
Asterisk server	bolek.feld.cvut.cz	Internet, adela, wvs
Voice enabler server	adela	wvs, bolek
Voice server	wvs	adela, bolek

Migration to private network

At the beginning, there are three servers with public IP addresses and so freely accessible from the Internet. That is unnecessary and in this case even unwanted because the servers are accessible for every attacker. To protect the servers but also keep their functionality we decided for a new network architecture that can be seen in the next picture.



Two servers (Voice Enabler and Voice Server) are now in a private network, inaccessible from the outer world. The only machine they need to contact is the Asterisk server and therefore their functionality remains unchanged while the security is highly improved. The Asterisk server must be accessible from the Internet and so it remained on the border between private network and public Internet.

TLS and Asterisk

TLS support in Asterisk

For more secure VoIP communication using Asterisk PBX and SIP protocol we decided to try to use TLS. TLS stands for Transport Layer Security and it is a cryptographic protocol that provides security and data integrity. More about TLS can be found at [1]. The new versions of Asterisk PBX nowadays has implemented a support of TLS, but there occured some problems with using it so after some investigation into the problem I wrote a simple how-to about Asterisk TLS support activation and the needed certificates generation, which you can find later in this text.

Upgrading Asterisk

Because the bolek server was previously running an older version of Asterisk PBX that did not support TLS, we had to upgrade to a newer version. Thankfully Asterisk provides good support for upgrading to a newer version so anything that was needed was downloading a new tarball, extracting it and running

```
$> ./configure
$> make
$> make upgrade
```

There were just some minor flaws that you should mind:

- do not forget to stop current version of Asterisk before upgrading
- check the result of configure and make scripts, it might need some changes like making symbolic links to libraries or installing additional dependencies
- do not run `$> make samples`, it would overwrite all your configuration files
- do not run the Asterisk like `$>asterisk` but always use a safe-asterisk script or `$>asterisk -vvvg`
- if you are using configuration scripts from previous version, take check twice whether they contain all necessary options (for example for using tls, also tcp must be enabled)

Certificates

When we want to use TLS, we need to have a certificate. The certificate is in simple words a string that proves that we are really us so that noone can confuse the other side and retake our place in communication. The certificate can be self-signed or signed by a CA(certificate authority). Because we do not really need to persuade the clients that we are truly us and we just need to use the certificate so that TLS support can be enabled, we will use a self-signed certificate. That may be dangerous when used openly, because anyone can generate its own self-signed certificate, but it is enough for our testing purposes. If you want to know more about the certificates and the work of TLS, consult [1].

How-to generate a self-signed certificate

One can easily generate his own self-signed certificate using OSSL (OpenSSL). OSSL is present in almost every Linux distribution and it can be also downloaded in the form of sources at [2] or in a form of precompiled binaries for Windows at [3]. Using OSSL for self-signed certificate generation is fast and simple and it can be done in the next few steps:

```
# first generate a private key, used method RSA, length 1024bits
$> openssl genrsa -out key.pem 1024
Generating RSA private key, 1024 bit long modulus
..............++++++
...................++++++
e is 65537 (0x10001)

# create a certificate request that has to be signed by CA, using our
previously generated private key
# answering the questions about Country Name and so on are not crucial,
the Common name is and must be
# the same as the servers IP or FQHN
$> openssl req -new -key key.pem -out request.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:CZ
State or Province Name (full name) [Berkshire]:Czech Republic
Locality Name (eg, city) [Newbury]:Prague
Organization Name (eg, company) [My Company Ltd]:RDC
Organizational Unit Name (eg, section) []:RDC-ACC
Common Name (eg, your name or your server's hostname) []:bolek.feld.cvut.cz
Email Address []:root@bolek.feld.cvut.cz

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

# self-sign the request
$> openssl x509 -req -days 360 -in request.pem -signkey key.pem -out
certificate.pem
Signature ok
subject=/C=CZ/ST=Czech Republic/L=Prague/O=RDC/OU=RDC-
ACC/CN=bolek.feld.cvut.cz/emailAddress=root@bolek.feld.cvut.cz
Getting Private key

# these two lines are very important, because Asterisk needs both key and
certificate in the same .pem file
$> cat key.pem > asterisk.pem
$> cat certificate.pem >> asterisk.pem
# now we have the asterisk.pem file which can be provided to Asterisk to
enable TLS support
```

Comments:

- `$>` is the unix prompt
- every command at the prompt is followed by unix response, so this is the way it should look when done properly (or in a very similar way depending on OS and version of OpenSSL)
- # is preceding a commentary line
- mind that the certificate is valid only for the period set up during generation (option -days)

If you do not want to transcribe all these commands, just use copy & paste and copy following lines into a script:

```
#!/bin/sh
openssl genrsa -out key.pem 1024
openssl req -new -key key.pem -out request.pem
openssl x509 -req -days 360 -in request.pem -signkey key.pem -out
certificate.pem
cat key.pem > asterisk.pem
cat certificate.pem >> asterisk.pem
```

How-to enable TLS support in Asterisk

Once we have the signed certificate and the private key, we need to provide these to Asterisk and tell him to use the TLS protocol. That is done via altering a configuration script /etc/asterisk/sip.conf in the following way:

```
[general]
tlsenable=yes
tlsbindaddr=147.32.195.157 ; use the address asterisk is bound to
tlscertfile=/~path/asterisk.pem  ; use the adress to the certificate
generated in previous part
tlscafile=/~path/asterisk.pem ; use the adress to the certificate generated
in previous part
```

It is highly recommended to backup the configuration script first and to take a look where we insert these new lines (it must be placed in the [general] section). DO NOT CREATE the [general] section, it is already present in the configuration file.
Since it is not a part of TLS, TCP must also be enabled or else TLS will not work properly (funny thing is that it will look like working properly until you try to use it:). To enable TCP, two more lines in the /etc/asterisk/sip.conf must be added in the [general] section:

```
tcpenable=yes
tcpbindaddr=147.32.195.157 ;
```

Now restart asterisk and everything should be ok.
To check whether TLS is set up, we can run the following commands:

```
$> netstat -an | grep 5060
tcp        0      0 147.32.195.157:5060         0.0.0.0:*
LISTEN
udp        0      0 147.32.195.157:5060         0.0.0.0:*
$> netstat -an | grep 5061
tcp        0      0 147.32.195.157:5061         0.0.0.0:*
LISTEN
```

The results should look like these above. Important is tcp listening on port 5061 and also tcp listening on port 5060. If one of these is not present, you probably forgot to set it up in the /etc/asterisk/sip.conf configuration script.

Open source IPS analysis

The result of our investigation on the field of open source IPS might look scary – there are none easy-usable open source IPS solutions. Every complete IPS solution is either paid or the project looks dead (no recent updates in few months) or there are some nontrivial hardware requirements. On the other side, there exists some open source IDS projects with many plug-ins that let you use an IDS, with some other software and a firewall, like an IPS. The most known and used is Snort IDS that was in the end also chosen as the security solution for our project.

Snort IDS

Snort IDS is the most often used open source IDS. IDS stands for Intrusion detection system and is used to detect any malicious attempts to scan, attack or misuse the server. Snort works on the base of packet inspection, if you want to know more about the principles, have a look at the project homepage [4]. Because it is the most often used, it is also a very alive project with periodically updated rules specifying new forms of attacks and in fact it does not have competitor in the field of open source IDS. The only problem with Snort is, that it is only an IDS...if an attack comes, Snort will send alert and his job is done. It is not able nor designed to block the attack in any way. Obviously that is not sufficient for improving security, because it is fine to know that there is an attack, but if you cannot stop it you are doomed anyway. Thankfully there were many people thinking the same way and it gave birth to many projects allowing to use Snort with some kind of plug-in as an IPS. IPS stands for Intrusion prevention system and it is able not only to find the attack but also to stop it. After a short investigation into existing plug-ins, Snortsam was chosen as an appropriate one to be used.

Installing Snort & Snortsam

Because Snortsam is just a plug-in for Snort, it must be installed once we already have a working installation of Snort itself. The installation of Snort is not as easy as it might look and there are many how-to documents available on the Internet. Some of them are good and some of these good are even working:). The problems often do not occur in the installation of Snort itself but in the fact that Snort needs some other software so one can be able to work with it (at least to work with it in an easy, human-like way). I have found one of these working how-to documents, that can be used, with some minor changes, for the newest version of Snort. This how-to can be downloaded from [5]. Beyond Snort it also describes how to install snort-rules, barnyard, php, apache, mysql, base, ntop and even the operating system itself. Some of these are not really necessary, but if you go along and do exactly what is written in the how-to, you will end with a friendly behaving and working Snort IDS.

Installing Snort

First we need the how-to document, that can be downloaded from [5] or we can use the version enclosed to this report. As I mentioned before, some minor changes must be done during installation so lets summarize them:
Even if you already have OS installed, do not skip the part about OS installation. Read it through, it contains some very useful settings you might need to use before installing Snort.
Do not use the old version of Snort, instead go to [4] and download the newest but STABLE release.
It is highly recommended to download new version of snort rules. That means do not use the download link mentioned on the page 8 of the how-to but register for free at [4] (you must be registered to get the new rules) and then download appropriate newest version of rules for your version of Snort (take care, choose according to the version of Snort you installed and do not take the CURRENT version).
While modifying snort.conf (page 9 of how-to) do not use preprocessor4_reassemble. Newer versions of Snort are using preprocessor5. If you want to change this, do so according to Snort manual, one simple change mentioned in how-to would not work.
When installing barnyard (page 11 of how-to) do not use the offered startup script or modify it before you use it. Problem is, that script was obviously written on Windows and has bad line endings. Easiest workaround is

```
$> dos2unix barnyard
```

supposing barnyard is the name of the script this will convert the line endings to unix style.
Installing NTOP is optional...you do not need it but it might be handy.
If you have done everything according to the how-to with the changes mentioned here, you should have Snort installed and active. I highly recommend to test it before installing Snortsam.

Installing Snortsam

Once we have Snort installed we can proceed to Snortsam. Do not forget to stop Snort before patching it. There is a how-to available for Snortsam installation at [6] but it is partly obsoleted so I recommend to use following two steps instead of steps 1 and 2 in this how-to.

1. Go to [7] and choose the tarball with newest version of Snortsam. Download it, extract it, make it and copy the binary to appropriate location(nowadays the newest version is 2.57, binaries are installed in /usr/local/bin).

    ```
    tar -xzf snortsam-src-2.57.tar.gz
    cd snortsam
    chmod 775 makesnortsam.sh
    ./makesnortsam.sh
    cp snortsam /usr/local/bin/
    ```

2. Go to [7] and choose the snort-X.X-plugin where X.X corresponds to your version of Snort installed (for Snort-2.6.1.3 it is snort-2.6-plugin). Download the diff file corresponding to your Snort version (again for Snort-2.6.1.3 it is snortsam-2.6.1.3.patch) into the Snort installdir. Go to Snort installdir and patch Snort sources

   ```
   patch -p1 <  snortsam-2.6.1.3.patch
   chmod 775 autojunk.sh
   ./autojunk.sh
   ```

   And recompile Snort.

   ```
   ./configure –with-mysql --enable-dynamicplugin
   make
   make install
   ```

The rest of the how-to is usable but very specific for local IP addresses, directories etc.
There is no need to use barnyard plug-in provided at [8] because it is already contained in the previously done patch.
Everything is ready, let's test it. The basic test of Snort and Snortsam is available in the how-to for pf plug-in at [9] in the Testing part.
On the testing machine, everything worked like charm:). Time to adapt the rules for specific attacks and SIP attacks...it will be documented in the next report.

Conclusion

The first steps in protecting the V2W servers were successfully finished. Servers were moved to the private network and only the Asterisk server remained directly accessible from the Internet which leaves him the only vulnerable target in the V2W project environment. TLS support was successfully enabled and tested on the Asterisk server so the call signalization is protected against eavesdropping and misuse. Snort IDS + Snortsam plug-in were chosen as an IPS solution and were installed on a test server. Basic tests proved functionality of Snort, preparations to deploy and test it on Asterisk server are in progress. The future plans are to deploy the proposed security solution on the Asterisk server and test it using some common simulated attacks, find the weak points and tighten the security.

Resources

[1] http://en.wikipedia.org/wiki/Secure_Sockets_Layer
[2] http://www.openssl.org/source/
[3] http://www.openssl.org/related/binaries.html
[4] http://www.snort.org/
[5] http://www.internetsecurityguru.com/
[6] http://doc.emergingthreats.net/bin/view/Main/SnortSamINSTALL
[7] http://www.snortsam.net/files/
[8] http://www.snortsam.net/files/barnyard-plugin/
[9] http://doc.emergingthreats.net/bin/view/Main/SnortSamREADMEpf