Automation Techniques of Building Custom Firmwares for Managed and Monitored Multimedia Embedded Systems

J. Slachta, J. Rozhon, F. Rezac, M. Voznak

***Abstract*—**One of the biggest challenges facing network administrators is the management of increasing amount of devices that are under their administration. The article deals with solution how to automatically build a system image with communication server and with advanced techniques of automated configuring such devices based on OpenWrt Linux distribution. The solution is built as a universal open source modular system and the server has been developing within the framework of a BESIP project (Bright Embedded Solution for IP Telephony) since May 2011. This open-source modular system with overall concept and the architecture is described in detail in this paper.

***Keywords*—**Build system, Provisioning, OpenWrt, VoIP Security, SIP.

I. Introduction

THE aim of the BESIP project is the development and implementation of embedded SIP communication server. Among all desired characteristics mainly belongs an easy integration into the computer network based on open-source solutions. This project serves as a secure and robust SIP IP telephony infrastructure available for anybody. It offers the prepared solution with integrated key components and the entire system is distributed as a firmware image or individual packages that might be installable from repositories. The main goal is to provide a solution which should be easily installable and configurable even without the deep knowledge of the technologies that are used by our key components. Also it aims to be scalable solution with unified configuration in mind [1].

Several open-source applications were adopted and implemented into developed modules, however within the implementation many modifications were required, especially in the core module (OpenWrt) due to complicated porting of applications into OpenWrt Buildroot. Our patches were verified and accepted by OpenWrt community. The speech quality monitoring tool was developed from scratch and implemented in Java. BESIP can run on embedded devices as well as on high performance devices. It requires at least 32 MB RAM and runs on the majority of OpenWrt supported devices [2],[3].

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

J. Slachta is a M.S. student with Dept. of Telecommunications, Technical University of Ostrava and he is also a researcher with Dept. of Multimedia in CESNET, Zikova 4, 160 00 Prague 6, Czech Republic (e-mail: slachta@cesnet.cz)

J. Rozhon is a PhD. student with Dept. of Telecommunications, Technical University of Ostrava and he is also a researcher with Dept. of Multimedia in CESNET, Zikova 4, 160 00 Prague 6, Czech Republic (e-mail: rozhon@cesnet.cz).

F. Rezac is a PhD. student with Dept. of Telecommunications, Technical University of Ostrava and he is also a researcher with Dept. of Multimedia in CESNET, Zikova 4, 160 00 Prague 6, Czech Republic (e-mail: filip@cesnet.cz).

M. Voznak is an Associate Professor with Dept. of Telecommunications, VSB-Technical University of Ostrava (17. listopadu 15, 708 33 Ostrava, Czech Rep.) and he is also a researcher with Dept. of Multimedia in CESNET (Zikova 4, 160 00 Prague 6, Czech Rep.), corresponding author provides phone: +420-603565965; e-mail: voznak@ieee.org.

.

II. State of the Art

As mentioned in the introduction, we discuss the implementation of a SIP communication server solution which would be an alternative to several current implementations. The main advantage of our solution is the ability to easily and quickly set up a full featured PBX on almost any hardware. We can presume that almost all implementations are based on open-source Asterisk PBX, web-interface for Asterisk and with a GNU/Linux distribution on the base layer.

At present, there are several projects that offer multipurpose IP telephony solutions for embedded devices and for household or enterprise platforms. The initial project of a GNU/Linux distribution which offers an easy set-up of IP telephony in a few steps is the Asterisk@Home project. The first version of this project was released on 29 April 2005. This project integrated a web interface for Asterisk, Flash Operators Panel to control and monitor PBX in real-time and also offered a full FAX support within one bootable image for almost any x86 PC. On 3rd May 2006 the development of this project was discontinued and was replaced by its successor Trixbox. However, the development of Trixbox does not seem to continue any more. Two existing projects - AsteriskNOW and Elastix – now offer an alternative to Trixbox.

The former, AsteriskNOW appears to be similar to Trixbox – a packed GNU/Linux distribution with Asterisk with a FreePBX web interface on top of it.

The latter, Elastix, is a bit more modular. Compared to any other project, it offers a slightly more modular hierarchy to facilitate the applicability to a multiple service server. The increasing popularity of embedded devices, such as Raspberry Pi, is the reason why the Micro Elastix distribution was born. However, all of those projects are either prepared for x86 machines only or for specific hardware. Micro Elastix only supports three platforms, namely PICO-SAM9G45, Raspberry Pi and MCUZONE.

None of the projects includes a security module that would offer a complete IPS and IDS system to prevent attacks against the SIP Registrar server. Also there is no module that would monitor the quality of voice calls transmitted through an integrated PBX. Thanks to the portability of the OpenWrt distribution we can prepare a BESIP bootable image for almost any device.

III. Architecture of BESIP

One of the biggest challenges during BESIP development was to create or modify any existing Linux distribution to serve our expectations. We needed to create an environment that would be fully customizable to any purpose and also to be easily maintainable through the time the BESIP would be developed. The advantage of portability to any platform was also welcomed. The choice of Linux distribution we wanted to modify fell on OpenWrt Linux distribution. The reason why we chose that system was the approach for building firmware, the toolchain, crosscompiler and all applications are downloaded, patched and built by scratch. This means that OpenWrt does not contain any source code, it does only have its build system with templates, patches and Makefiles with procedures how to build a system and its packages for targeted device. This approach allows us to create custom procedures for build system and packages that can be modified at any stage.

A simplified view on BESIP architecture is depicted in Fig. 1 which describes how the architecture is designed. The first block, the build system, is a wrapper on the top of the OpenWrt build system. It is designed for easy creation of firmware images within the single text file which describes what should be built for specific architecture and device we are targeting on. With the build system comes also several BESIP packages that are customizable from the OpenWrt buildroot, e.g. before the firmware is built.BESIP packages consists of several modules which provides functionality as:

- the PBX module to accomplish VoIP functionality,
- the Monitoring module to monitor speech quality and the system itself,
- the Security module to provide IPS/IDS system,
- the Core module as a glue to all services among themselves and to provide intuitive interface to them.

The security module is based on SNORT; SNORTSam and iptables [4]. In addition to this, the Kamailio ratelimit and pike module is used for defending attacks.

The monitoring module exploits a tshark package and our java code which interprets its results and gives information about particular speech quality. The Zabbix agent is used to report basic states of the entire system and finally the PBX module is made from Kamailio in conjunction with Asterisk.

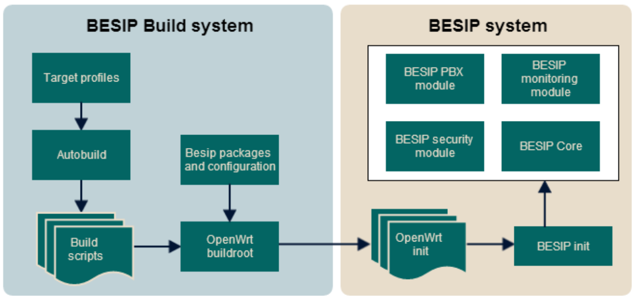

Fig. 1 Architecture of BESIP system.

The Core module is a shell library (providing functions for all executable BESIP scripts) with executable files which makes all mentioned services fully working.

IV. Build System for Besip

Before we describe the concept of BESIP system, it is necessary to introduce the BESIP build system which makes automation of creation system images much easier. As said above, BESIP is based on GNU/Linux distribution OpenWrt which is built on top of the OpenWrt Buildroot. Buildroot is a set of Makefiles and files that allows to compile cross-compilation toolchain and to generate by that toolchain resulting cross-compiled applications into a root filesystem image to be used in a targeted device. Cross-compilation toolchain is compiled by host compilation system which is provided by any GNU/Linux distribution.

In the beginning of BESIP development, we met issues that were holding us back. We could not test all changes immediately, we had to recompile all code and generate images nearly always when we ported new application, modified post installation scripts or when cross-compilation toolchain has changed. Also, the system behaves differently during testing if it is new root filesystem image, or modified root filesystem that has been run more than once. At least those issues led us to create an easy interface that will ease the creation, automation and functional testing for system images.

BESIP build system is a set of scripts, Makefiles and definition files that make an easy interface to OpenWrt Buildroot. We can consider the main Makefile to be as a core of the BESIP build system. It performs all atomic operations with OpenWrt Buildroot, works with source code management systems (to update/revert/any operation with local copies of OpenWrt source codes), patches OpenWrt Buildroot and executes images as virtual machines. Those commands might be used by any user or by autobuild scripts, which will be described after.

On the top of the core Makefile is autobuild script. This script calls all atomic operations within more complex parameterized operations whose variables are defined in specific target files. Those target files are user defined and on the basis of those files are configuration files for OpenWrt

Buildroot created. Once we have configuration files the system images could be created by calling autobuild.sh script with command *build* and parameter containing the name of the target file.

The following simplified example shows, how to create target file.

```
TARGET_CPU=x86
OWRT_NAME=trunk
TARGET_NAME=virtual_\$(BESIP_VERSION)-
owrt_\$(OWRT_NAME)
TARGET_QEMU=i386
TARGET_QEMU_OPTS=-m 512
OWRT_IMG_DISK_NAME=openwrt-\$(TARGET_CPU)-
generic-combined-squashfs.img
BESIP_PACKAGES= gnugk=y suricata=y
EMBEDDED_MODULES += SATA_AHCI VMXNET3
```

The following example shows how to build system images based on the target file.

```
#./autobuild.sh build virtual-x86-trunk
```

Such techniques can be used for any purpose of automated building system images for any device or platform supported by OpenWrt. Those could be firmware images for campus access points, specialized network probes, virtualized multimedia servers or any other devices.

V. Core Module

The role of the Core module is to provide a glue among all services that served by all BESIP modules. The most important part of the Core module is the BESIP shell library that provides functions for all utilities and scripts used by BESIP system. Functionality of a Core module complements utilities for configuration management and for simplified configuration of system image. With all those utilities comes along also default configuration which prepares all module services into fully functional state with all BESIP modules running and operational. Also, the role of this module is to switch any existing OpenWrt environment to BESIP environment while the device is booted the first time or the BESIP environment is used and ran the first time.

A. BESIP Environment

BESIP environment handles tasks which has to be performed at several stages in OpenWrt operating system. After the BESIP firmware image is built at this stage the operating system behaves as clean operating system with installed dependencies required by BESIP package and its submodules. On the first boot the init script is performed and it waits until the overlay filesystem is mounted. When the filesystem is mounted the *first_boot* procedure is performed. This procedure incorporates the initial setup of the system and preparation configuration files. The main advantage of this procedure is the applicability to any existing setup of OpenWrt system.

Another mandatory part of BESIP system is an executable application that provides functions to manage following procedures:

- Generates provisioning data for connected phones,
- resetting system image into factory defaults,
- performs system upgrade,
- controls internal BESIP modules,
  - configuration and management of security module,
  - importing and setting up a dial plan for PBX module,
- collects information for crash reporting to be used for debugging.

B. Provisioning Client

The impetus for development of provisioning tool arose during the period when firmware images created by BESIP build system were deployed to computers, routers and wireless access points. Those machines were not configured for target networks, which were supposed to be deployed on. Because the target configuration does not depend on a person which builds the system, but on the network administrator, then configuration should lay outside of a BESIP firmware image. The creation of such tool bring a question how should the target device should fetch and apply its configuration.

In the build system, we can pass static information about our provisioning server which provides configuration (during build time). We can also change this information in firmware image. This information can be used for protocols which translates one kind of information to another. As an example we can use DNS protocol and its TXT records. The target configuration could be stored on a server designated within an URI in a variable from TXT record which is obtained from static URL provided by BESIP build system. This solution is replicable for any protocol which allows distribution that kind of information (LLDP, DHCP or any other else).

An example how to resolve UCI provisioning URI:

```
host -t txt provdomain
provdomain descriptive text
"provuri=http://12.34.56.78/uciprov/"
```

If a device knows where to obtain configuration from then the device can construct all provisioning URI addresses for each device state it needs. This approach is needed when system administrator needs to differentiate configuration for devices which starts up the first time, if those devices are refreshing its common device configuration on a regular basis or if it is the configuration that is obtained after device startup. UCI provisioning client written for BESIP currently handles only configuration files that are handled by UCI system (Unified Configuration Interface) for centralized configuration. If a device knows where to obtain configuration from, then the device can obtain configuration data from ordinary transport protocols designated in provisioning URI. The benefits that BESIP draws from OpenWrt builds upon the UCI configuration system which is based on plain text configuration files with firmly defined structure. This configuration is obtained using software for file retrieval from network resources, e.g. wget, and immediately imported into UCI.

The client side of UCI provisioning currently has several stages:

1. Waiting for the system to be ready to be provisioned,
2. Stage 1 - receive provisioning URI using supported protocols,
   - For each supported protocol try to receive provisioning URI address,
   - Construct a list of URI provisioning addresses based on received URI address.
3. Stage 2 - obtain configuration from URI received in stage 1,
   - For each interface try to obtain configuration data.
4. Stage 3 - apply received configuration.

The server side of UCI provisioning is currently solved by providing static file structure with files which consists of export provided by UCI system. See sequence diagram depicted in Fig. 2 to see how the UCI provisioning works.

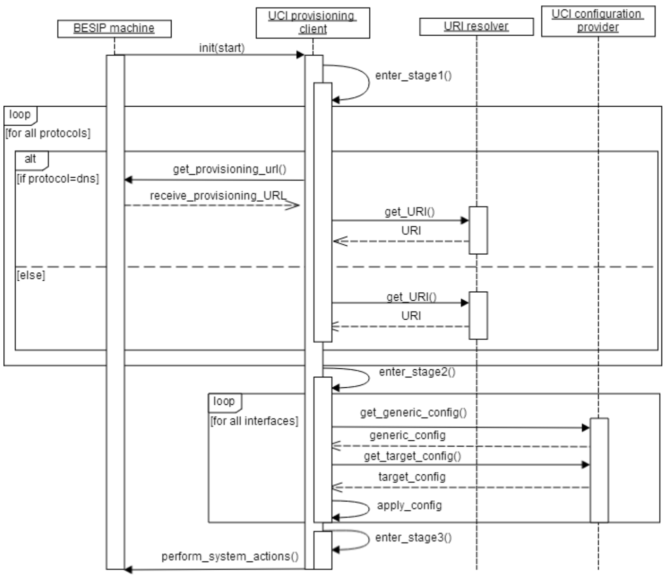

Fig. 2 Sequence diagram of UCI provisioning client.

VI. PBX Module

The PBX module is a key part of the BESIP project. It operates as SIP proxy or SIP B2BUA, depending on configuration, and ensures a call routing. Asterisk is used for call manipulation and for the PBX functions. Kamailio is used for the proxying SIP requests, the traffic normalization and for the security [5]. There are always two factors when developing VoIP solution the first one is high availability and reliability, the second one is an issue of advanced functions. Many developers try to find a compromise, we have implemented both, and our BESIP is able to adapt to the users requirements. More complex system can handle many PBX functions such as a call recording or an interactive voice response but due to the bigger complexity it is more susceptible to fault. On the opposite side, pure SIP proxy is easier software, which can perform call routing, more fault tolerant, but it is more difficult to use the advanced PBX functions [6].

VII. Security Module

Security module is very important part of BESIP and all the time, it was considered to make the developed system as secure as possible. Next to this, entire system has to be fault-tolerant, monitored and protected from attacks. It means that if the device is under attack, only attacker has to be blocked, not entire system or other users. If there is some security incident, BESIP immediately solves the situation and notifies this event in a detailed report to administrator.

The attack are recognized and processed by SNORT rules, the source IP address is automatically sent into the firewall by SNORTSam and the intruder’s IP is blocked. This is very flexible, reliable and effective implementation. Dropping attack based on IP directly in the Linux kernel is much more efficient than to check messages on the application level. Only first messages are going to SNORT filter. When SNORT identifies a suspicious traffic, next messages from the same IP are blocked.

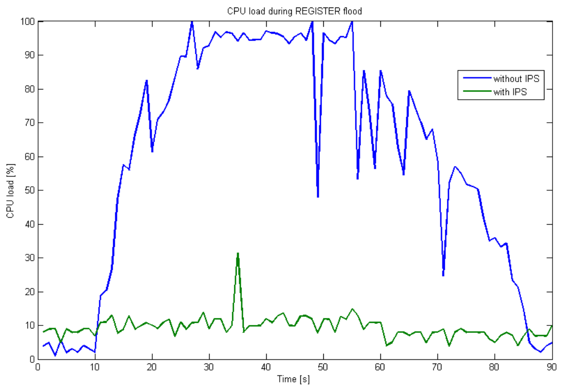

Fig. 3 Attack effectiveness based on REGISTER flood.

If more soft faults appear from some IP, it is blocked at the IPTABLES level; this approach can effectively block incorrectly configured clients and servers. For example, if a client sends REGISTER with proper credentials, it is not obviously security attack but the client attempt to register again and again, with every registration requires computing sources at SIP REGISTRAR server. Such attempts can be denoted and blocked for a time interval. Administrators can use Zabbix agent inside BESIP to gather all information directly into their monitoring system. The monitoring is very important part of the security module and BESIP team was already focused on the issue in early design [7]. Partially, BESIP is resistant to some kind of DoS attacks. It depends on hardware used. If the hardware is strong enough to detect some security incidents on application level, the source IP is immediately dropped. Low performance hardware cannot handle such detection on application level. In such case, it is better stop DoS attacks before it reaches BESIP. For example, SNORT on a dedicated machine will be much more flexible than if is an integral part of VoIP system. Therefore, we recommend using an external IPS system to make VoIP service robust and secure. Nevertheless BESIP includes own IPS/IDS system [8], [9].

The features of our security module were verified in test-bed and results are depicted in Fig. 3 [7]. The CPU load was monitored during trivial SIP attacks. The line SSI (Snort, SnortSam, IPtables) represents the response in case of active security module in BESIP whereas next dependencies were measured without SSI. There were emulated only two types of DoS attacks, namely REGISTER flood and INVITE flood. In order to generate these attacks, we used sipp generator and in case of INVITE also inviteflood tool. The dependencies in both figures clearly prove the ability of security module to mitigate the performed attacks.

VIII. Conclusion

As we have mentioned, BESIP consists of several components, which are distributed under GPL as an open-source solution. A few of them have been fully adopted such as components in Security and PBX modules, some of them modified, concerning the Core module and finally we have developed own tool for Speech quality assessment. The contribution of our work is not only hundreds of hours spent on the development, on the coding BESIP system, we bring a new idea of the unified configuration management, with unified CLI syntax which enables to configure different systems, Asterisk and Kamailio in our case.

BESIP is distributed as a functional image for several platforms, mainly for x86 platform, which is also possible to run it on any virtualization x86 software. There are several example firmware images for several target devices, such as TP-Link access points or Raspberry PI computer. Configuration is available through web-browser, SSH client or to be provisioned using supported provisioning protocols. After the testing, version 2.0 will be released; a new release 2.0 will be based completely on NETCONF with one API to configure the entire system. Next to this, CLI syntax has been developing and will be connected to NETCONF. CLI will be independent of internal software so if some internal software is modified, there will be no change in configuration. Even more, CLI and NETCONF configuration will be independent on hardware and version. To export configuration from one box and to import it to the next one will be a simple task. Users will modify only one configuration file to manage entire box. Project pages are available at [10], binary images from the auto-build system can be downloaded from [11] and source codes can be checked out via SVN from the same page as well [11].

References

[1] M. Voznak, F. Rezac: “Threats to voice over IP communications systems”. *WSEAS Transactions on Computers*, Volume 9, Issue 11, 2010, pp. 1348-1358.

[2] F. Abid, N. Izeboudjen, M. Bakiri, S. Titri, F. Louiz, D. Lazib: “Embedded implementation of an IP-PBX/VoIP gateway”. *24th International Conference on Microelectronics*, December 2012, IEEE, Article number 6471377.

[3] N. Titri, F. Louiz, M. Bakiri, F. Abid, D. Lazib, L. Rekab: “Opencores /Open-source Based Embedded System-on-Chip Platform for Voice over Internet”. *INTECH: VOIP Technologies*, pp. 145-172.

[4] J. Safarik, F. Rezac, M. Voznak: “Monitoring of Malicious Traffic in IP Telephony Infrastructure”. *Technical Report*, 10p., December 2012.

[5] M. Voznak, J. Safarik: “DoS attacks targeting SIP server and improvements of robustness”. *International Journal of Mathematics and Computers in Simulation*, Volume 6, Issue 1, 2012, pp. 177-184.

[6] J. K. Prasad, B. A. Kumar: “Analysis of SIP and realization of advanced IP-PBX features”. *3rd International Conference on Electronics Computer Technology*, Volume 6, 2011, IEEE Article number 5942085, pp. 218-222.

[7] M. Voznak, K. Tomala, J. Vychodil, J. Slachta: “Advanced concept of voice communication server on embedded platform”. *Przeglad Elektrotechniczny*, Volume 89, Issue 2 B, 2013, pp. 228-233.

[8] D. Endler, M. Collier: “Hacking Exposed VoIP”. McGraw-Hill Osborne Media, 2009.

[9] 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.

[10] Management of BESIP Project, LipTel Team, 2014, https://besip.cesnet.cz

[11] Project BESIP, https://homeproj.cesnet.cz/projects/besip/wiki/Download