MANTECH Publications

Journal of Android iOS Development and Testing
Volume 2, Issue 3, September-December 2017

Using Embedded Linux to Implement a Secured System

P. Sivakumar[1], R. S. Sandhya Devi[2], B. Vinod[3]

Department of Electrical Engineering

PSG College of Technology, Tamil nadu, India

Corresponding Authors' email id: *psk@eee.psgtech.ac.in[1], sandhyadevi.rs.eee@kct.ac.in[2], bvin@rae.psgtech.ac.in[3]*

Abstract

Now a day's Security is a major concern in case of Embedded Systems since almost all the systems are to be connected with the Internet. Embedded system serves as one of crucial components needed for various applications and services in pervasive computing environment. Security problems related to embedded systems directly influence credibility of these applications and services. In order to effectively eliminate weaknesses in current embedded systems and strongly enhance safety practices of these systems. With the ever-increasing presence of Linux implementations in embedded devices, there is a strong demand for defining the security requirements and augmenting, enhancing, and hardening the operating environment. In most of the systems this issue is addressed with the help of proprietary software and operating systems which ultimately increase the cost of product. Since the proprietary software is having its own advantage cost incurred by that is much when compared to open source. Hence in the embedded device development the very good alternative is the Linux operating system which is royalty free and the source code is freely available which comes under the GNU public license. The security of Linux is already proven and is currently used in millions of devices. Hence the embedded devices with the Linux operating system is Low cost, highly secured and can be designed in a short span of time. Thus it reduces the cost of the product. In this paper the Linux kernel is configured for the ARM core based System on Chip and an authentication system is implemented. The

host development environment is the Ubuntu Linux. The kernel image is cross compiled using the GCC compiler

Keywords: *Linux, Secure System, Kernel, File System, Linux Security Module, Target Image Builder*

INTRODUCTION

The specific constraints that must be satisfied by embedded systems, such as timeliness, energy efficiency of battery operated devices, dependable operation in safety-relevant scenarios, coupled with the never-ending pressure to increase the functionality, lead to an enormous growth in the complexity of the design at the system level. These issues in the system design leads to increase in the design time which ultimately increases the time-to-market and the production cost.

Now a day's Security is a major concern in case of Embedded Systems since almost all the systems are to be connected with the Internet. In most of the systems this issue is addressed with the help of proprietary software and operating systems which ultimately increase the cost of product. As the competition among the manufacturers increases they will try to sell their products in a competitive price and high quality.

As for as embedded devices are concerned the cost of the product is decided by the software used and development time. Hence if an embedded device is designed using the proprietary software's royalty should be paid to the software vendor which increases the product cost. Since the proprietary software is having its own advantage cost incurred by that is much when compared to open source.

Hence in the embedded device development the very good alternative is the Linux operating system which is royalty free and the source code is freely available which comes under the GNU public license. The security of Linux is already proven and is currently used in millions of devices [8].

Hence the security in embedded systems can be easily achieved by Embedded Linux, and wide availability of patches and ports for various architectures will reduce the design complexity and time to market.

Hence the embedded devices with the Linux operating system is Low cost, highly secured and can be designed in a short span of time. Thus it reduces the cost of the product.

Embedded Security Requirements Figure.1 shows security requirements of an end-user, where use the term basic security functions to denote the set of confidentiality, integrity and authentication requirements. Embedded system access should be restricted to a selected set of authorized users, access to a network has to be provided only if the device is authorized. These may use the user or host authentication mechanisms provided by the basic security functions, as well as other mechanisms such as biometrics and access control [7].

Another security function is the availability of the embedded system. Embedded system security often requires protecting critical or sensitive information including making sure that it is properly erased at the end of its lifetime. Tamper resistance maintain security requirements even when the device falls into the hands of malicious parties, and logically probed. In this paper an authentication system is designed using Embedded Linux.

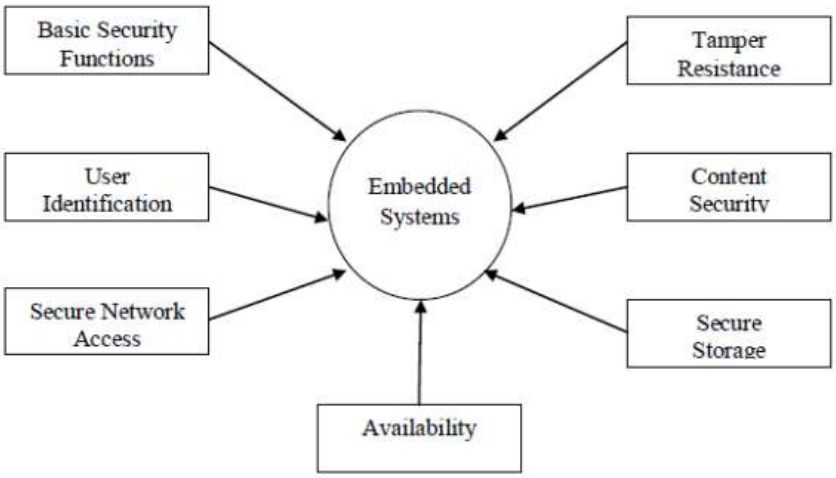

Figure 1: Common security requirements of embedded system user perspective

LITERATURE REVIEW

The evolution of computing and information technology is resulting in migration beyond centralized enterprise and desktop systems and out to increasingly sophisticated embedded devices in the field [4]. This creates significant opportunities for equipment manufacturers to have real-time insight into customer operations and to offer additional value-added services and technologies. Many opportunities exist for better use of limited communications channels by leveraging the computation power now resident in existing and emerging network-enabled devices, weaponry, sensors, and situational awareness systems. Therefore, implementing security in embedded systems and providing secure communications paths back into the enterprise are paramount. Prototyping and implementation of embedded security features involve the integration of real-time operating systems, protocol stacks, and cryptographic algorithms within resource-limited hardware platforms [4].

Embedded Linux System Security

MATLAB /Simulink is a graphical environment for simulation and Model-Based Design for multi-domain dynamic and embedded systems. Simulation tools are necessary to assist designers in developing such model-based concept. To evaluate a concept, the very common approach used by PSA consists on linking the model with a global dynamic vehicle model. Running on Matlab/Simulink tools and parameterized by a configuration file mass, inertia, tire model. To evaluate the concept in the car, the model is embedded to code generator tools like Target Link with associated physical models such as networks.

Open source Linux systems have served as the leading platforms for the development of embedded systems. Prevailing embedded Linux systems include μCLinux that manages microcontrollers without Memory Management Units (MMUs), RTLinux that supports real time applications, and ARMLinux that administers microcontrollers with MMUs, and so on [1]. An embedded Linux kernel is a derivative of Linux kernel and inherits most advantages from Linux operating system. Embedded systems successfully provide many applications such as routers, STB, PDA, intelligent appliances, instruments and facilities for Aeronautic and astronautic researches [5].

Access Control Kernel Logic Based On LSM

Embedded Linux operating system usually runs in single user mode. A user's operation is executed by a process. A process is the only secure subject and objects include all objects within the operating system. With Linux Security Module (LSM) framework's imbedding into Linux2.6 kernel, there exits a uniform measure for implementation of mandatory access mechanism Embedded Linux operating system with security enhancement uses LSM framework, adopts security policy of BLP model and DTE model, utilizes security module stacking technology and assigns security label for process and resource in the system to implements mandatory access control. The security label can be defined as a two-element array (domain/type: security level), where domain denotes sign of users' privilege of process operation, type denotes class sign of resource, security level denotes security level of use process and objects. Domain/type implements security policy of DTE. The system control access resource by domains and types to ensure integrity of the system. Security level implements security policy of BLP to improve confidentiality of system. Multi-policy mandatory access control based on LSM and its kernel logic are illustrated in Figure 2.

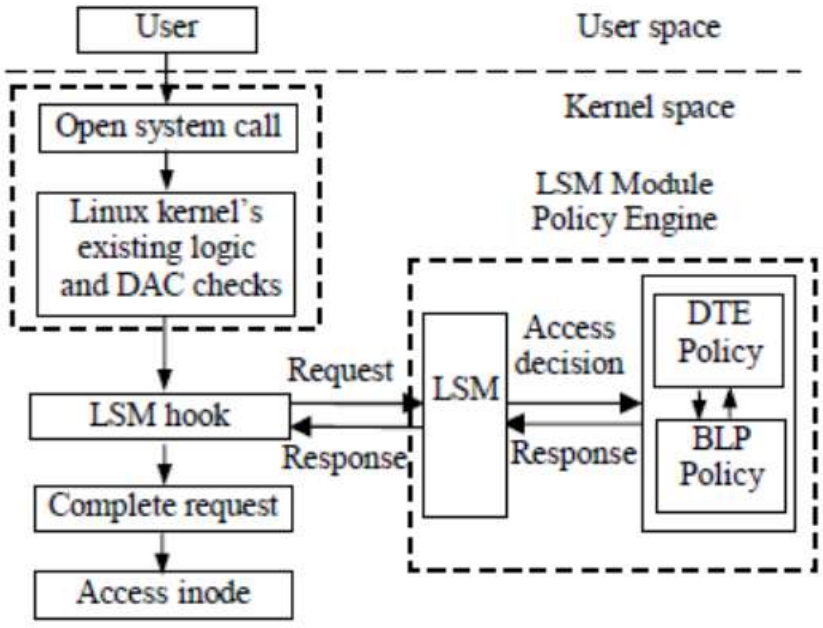

Figure 2: Multi-policy mandatory access control kernel logic based on LSM

Linux Kernel

The Linux kernel is an operating system kernel used by a family of Unix-like operating systems. The term Linux distribution is used to refer to the vario- us operating systems that run on top of the Linux Kernel. The Linux kernel is released under the GNU General Public License version 2 (GPLv2) and developed by contributors worldwide; Linux is one of the most prominent examples of Free / Open Source software.

The Linux kernel was initially conceived and created by Finnish software engineer Linus Torvalds in 1991. It is debated whether Loadable Kernel Modules (LKMs) should be considered derivative works under copyright law, and thereby fall under the terms of the GPL.

Linux Security

Module Linux Security Modules (LSM) is a framework that supports a variety of computer security models while avoiding toward any single security implementation. The framework is licensed under the terms of the GNU General Public License and is standard part of the Linux kernel. LSM designed to provide the specific needs of everything needed to successfully implement a mandatory access control module, while imposing the fewest possible changes to the Linux kernel. LSM avoids the approach of system call interposition as used in Systrace because it does not scale to multiprocessor kernels. LSM's access control goal is very closely related to the problem of system auditing. LSM cannot deliver that, because it would require a great many more hooks, so as to detect cases where the kernel "short circuits" failing system calls and returns an error code before getting near significant objects.

GNU Public License

The GPL was written by Richard Stallman in 1989 for use with programs released as part of the GNU project. Stallman's goal was to produce one license that could be used for any project, thus making it possible for many projects to share code. Prominent free software programs licensed under the GPL include the Linux kernel and the GNU Compiler Collection (GCC). Some other free software programs are dual-licensed under multiple licenses, often with one of the licenses being the GPL.

The second version of the license, version 2, was released in 1991. Some members of the FOSS (Free and Open Source Software)

community came to believe that some software and hardware vendors were finding loopholes in the GPL, allowing GPL-licensed software to be exploited in ways that were contrary to the intentions of the programmers.

Survey of 32-Bit Architectures

Power PC

Performance Optimization With Enhanced RISC – Performance Computing is a RISC architecture created by the Apple– IBM– Motorola alliance, known as AIM. Originally intended for personal computers, PowerPC CPUs have since become popular embedded and high-performance processors. PowerPC was the cornerstone of AIM's PReP and Common Hardware Reference Platform initiatives and while the architecture is well known for being used by Apple's Macintosh lines from 1994 to 2006, its use in video game consoles and embedded applications far exceeded Apple's use. The history of the PowerPC begins with IBM's 801 prototype chip of John Cocke's RISC ideas in the late 1970s. 801-based cores were used in a number of IBM embedded products, eventually becoming the 16-register ROMP processor used in the IBM RT. The RT had disappointing performance and IBM started the America Project to build the fastest processor on the market. specification and a new reference platform for servers called PAPR (Power Architecture Platform Reference).

ARM

The ARM is a 32-bit Reduced Instruction Set Computer (RISC) Instruction Set Architecture (ISA) developed by ARM Holdings. It was known as the Advanced RISC Machine, and before that as the Acorn RISC Machine. The ARM architecture is the most widely used 32-bit ISA in terms of numbers produced. They were originally conceived as a processor for desktop personal computers by Acorn Computers, a market now dominated by the x86 family used by IBM PC compatible computers. The relative simplicity of ARM processors made them suitable for low power applications. This has made them dominant in the mobile and embedded electronics market as relatively low cost and small microprocessors and microcontrollers [1].

ARM processors are used extensively in consumer electronics, including PDAs, mobile phones, digital media and music players, hand-held game consoles, calculators and computer peripherals such as hard drives and routers.

The ARM core has remained largely the same size throughout these changes. ARM2 had 30,000 transistors, while the ARM6 grew to only 35,000. ARM licensed about 1.6 billion cores in 2005. In 2005, about 1 billion ARM cores went into mobile phones. As of January 2008, over 10 billion ARM cores have been built, and iSuppli predicts that 5 billion a year will ship in 2011.

The architecture supported on smart phones, personal digital assistants and other handheld devices is ARMv5. XScale and ARM926 processors are ARMv5TE, and are now more numerous in high-end devices than the Strong ARM, ARM9TDMI and ARM7TDMI based ARMv4 processors, but lower-end devices may use older cores with lower licensing costs. ARMv6 processors represented a step up in performance from standard ARMv5 cores, and are used in some cases, but Cortex processors (ARMv7) now provide faster and more power-efficient options than all those previous generations. Cortex-A targets applications processors, by smart phones that previously used ARM9 or ARM11 [Seal, 2001]. To improve code-density, processors have the Thumb mode. In this mode, the processor executes 16-bit instructions. In situations where the memory port or bus width is constrained to less than 32 bits, the shorter Thumb opcodes allow increased performance compared with 32-bit ARM code.

MIPS

Microprocessor without Interlocked Pipeline Stages (MIPS) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Computer Systems. The early MIPS architectures were 32-bit, and later versions were 64- bit. The current revisions are MIPS32 (for 32-bit implementations) and MIPS64 (for 64-bit implementations). MIPS32 and MIPS64 define a control register set as well as the instruction set. MIPS implementations are currently primarily used in Windows CE devices and embedded systems such as, routers, residential gateways, and video game consoles and PlayStation Portable. One of the more interesting applications of the MIPS architecture is its use in massive processor count supercomputers.

SHARC

The SHARC is a Harvard architecture word-addressed VLIW processor. This means that the processor knows nothing of 8- bit or 16-bit values, and is thus neither

big-endian nor little-endian. The word size is 48-bit for instructions, 32-bit for integers and normal floating-point, and 40-bit for extended floating-point. Code and data are normally fetched from on-chip memory, which the user must split into regions of different word sizes as desired. Off-chip memory can be used with the SHARC. This memory can only be configured for one single size. If the off-chip memory is configured as 32-bit words to avoid waste, then only the on-chip memory may be used for code execution and extended floating-point.

Operating systems may use overlays to work around this problem, transferring 48-bit data to on-chip memory as needed for execution. A DMA engine is provided for this. True paging is impossible without an external MMU.

The SHARC has a 32-bit word-addressed address space. Depending on word size this is 16 GB, 20 GB, or 24 GB. SHARC instructions may contain a 32-bit immediate operand. Instructions without this operand are generally able to perform two or more operations simultaneously. The SHARC processor has built-in support for loop control. Up to 6 levels may be used, avoiding the need for normal branching instructions and the normal bookkeeping related to loop exit. The SHARC has two full sets of general-purpose registers. Code can instantly switch between them, allowing for fast context switches between an application and an OS or between two threads.

IMPLEMENTATION OF THE SYSTEM

In this paper the Linux 2.6.27 kernel is configured for the ARM926EJ-S core based LPC3250 System on Chip and an authentication system is implemented which runs from the ARM processor. The hardware used here is the LPC3250 based Embedded Artists QVGA Development board and the software tool is Linux Target Image Builder. The host development environment is the Ubuntu Linux 9.04 version. The kernel image is cross compiled using the GCC compiler.

Target System

Key features of LPC3250

- 266MHz, 32-bit ARM9EJ-S core
- Vector Floating Point coprocessor
- Up to 256KB of internal SRAM and 32KB I- cache/32KB D-cache

- External memory controller for DDR and SDR SDRAM, SRAM, and Flash
- Available 10/100 Ethernet MAC
- USB OTG with full-speed host and device capabilities
- Available 24-bit LCD controller supports STN and TFT panels
- Comprehensive set of serial interfaces

Built around a 90-nm, 266MHz ARM926EJ-S CPU core and a Vector Floating Point (VFP) coprocessor, the NXP LPC32x0 family is designed for applications that require high performance, high integration, and low power consumption.

The VFP coprocessor increases the speed of calculations by a factor of four to five in scalar mode. Multiple interfaces for serial communications increase design flexibility, provide larger buffer size.

A seven-layer, 32-bit, 104-MHz AHB matrix provides a separate bus for each of the seven AHB masters (D-cache, I-cache, two DMA, Ethernet MAC, USB controller, and LCD controller).

QVGA Display

The Quarter Video Graphics Array (also known as Quarter VGA, QVGA, or qVGA) is a popular term for a computer display with 320 × 240 display resolution. QVGA displays are most often used in mobile phones, PDAs and some handheld game consoles. QVGA video is typically recorded at 15 or 30 frames per second.

QVGA mode describes the size of an image in pixels called the resolution; numerous video file formats support this resolution. In QVGA the "Q" prefix commonly means quad (ruple) four times higher display resolution. To distinguish quarter from quad, lowercase "q" is sometimes used for "quarter" and uppercase "Q" for "quad".

Structure for an Embedded LINUX

Structure of Embedded Linux as shown in Figure 3 has the following sub components Device Driver: In computing, a device driver or software driver is a computer program allowing higher-level computer programs to interact with a hardware device. A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program

invokes a routine in the driver, the driver issues commands to the device, once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system-specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

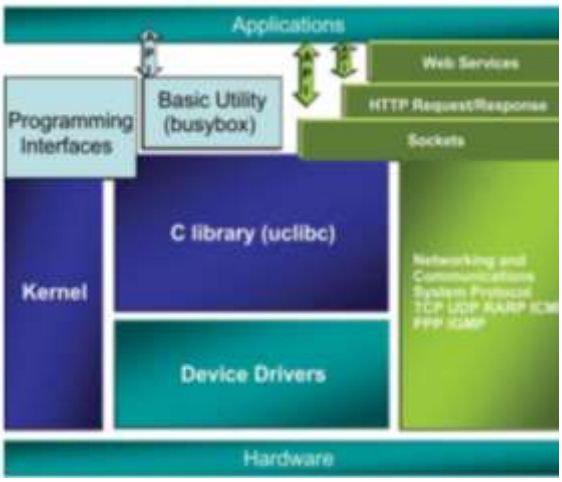

Figure 3: Structure of an Embedded Linux Kernel

BusyBox: BusyBox is a software application released as Free software under the GNU General Public License that provides many standard Unix tools, much like the larger (but more capable) GNU Core Utilities. BusyBox is designed to be a small executable for use with the Linux kernel, which makes it ideal for use with embedded devices.

Programming Interfaces: An Application Programming Interface (API) is an interface implemented by a software program which enables it to interact with other software. It is similar to the way the user interface facilitates interaction between humans and computers. An API is implemented by applications, libraries, and operating systems to determine their vocabularies and calling conventions, and is used to access their services. It may include specifications for routines, data structures, object classes, and protocols used to communicate between the consumer and the implementer of the API.

Memory Management Unit

A memory management unit (MMU), sometimes called paged memory management unit (PMMU), is a computer hardware component responsible for handling accesses to memory requested by the CPU as shown in Figure 4. Its functions include translation of virtual addresses to physical addresses (i.e., virtual memory management), memory protection, cache control, bus arbitration, and, in simpler computer architectures (especially 8-bit systems), bank switching.

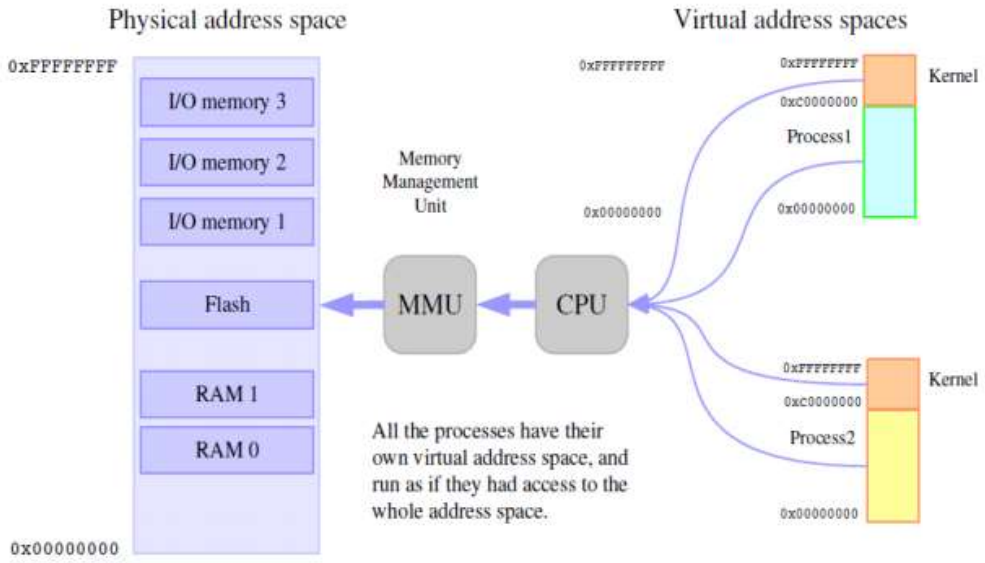

Figure 4: Physical and Virtual Memory map

Linux File System

The Linux file system as shown in Figure 5.

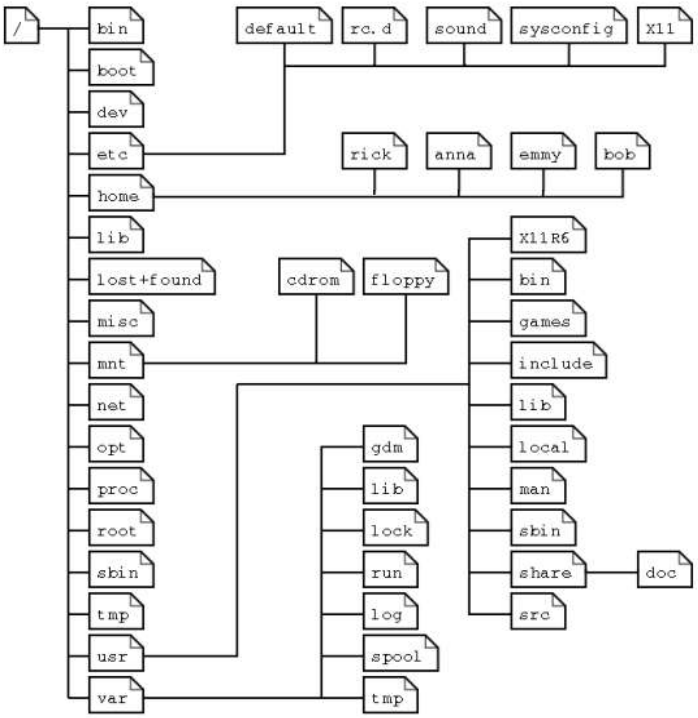

Figure 5: Linux file system layout

Journal ed File systems

Linux supports various file systems like ext2, ext3, ext4, NFS, JFS, CRAMFS etc

Here we are using the Journal file system because of the following advantages

1. Designed to stay in a correct state even after system crashes or a sudden power off.
2. All writes are first described in the journal before being committed to files.

The structure of the journal file system is as shown Figure 6

LINUX TARGET IMAGE BUILDER

The LTIB (Linux Target Image Builder) project is a tool that can be used to develop and deploy BSPs (Board Support Packages) for a number of embedded target platforms including PowerPC, ARM, Coldfire. Flow diagram for LTIB as shown in Figure 7.

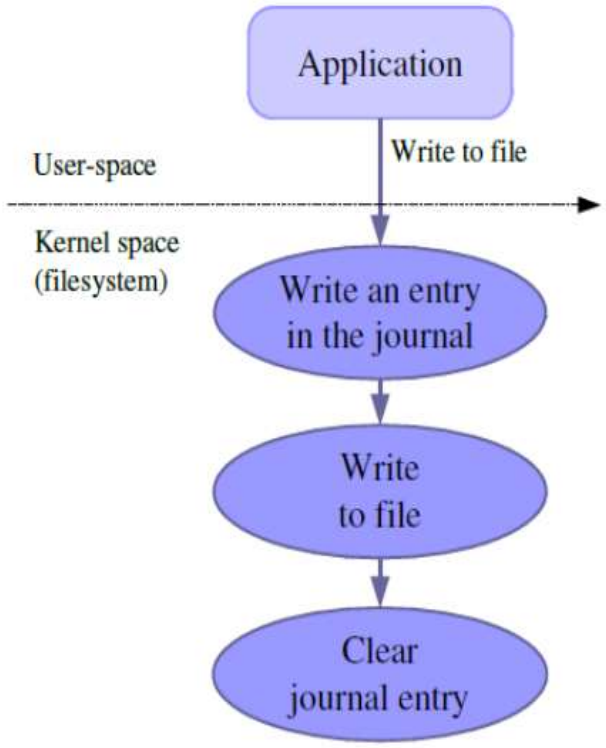

Figure 6: Structure of the Journal File System

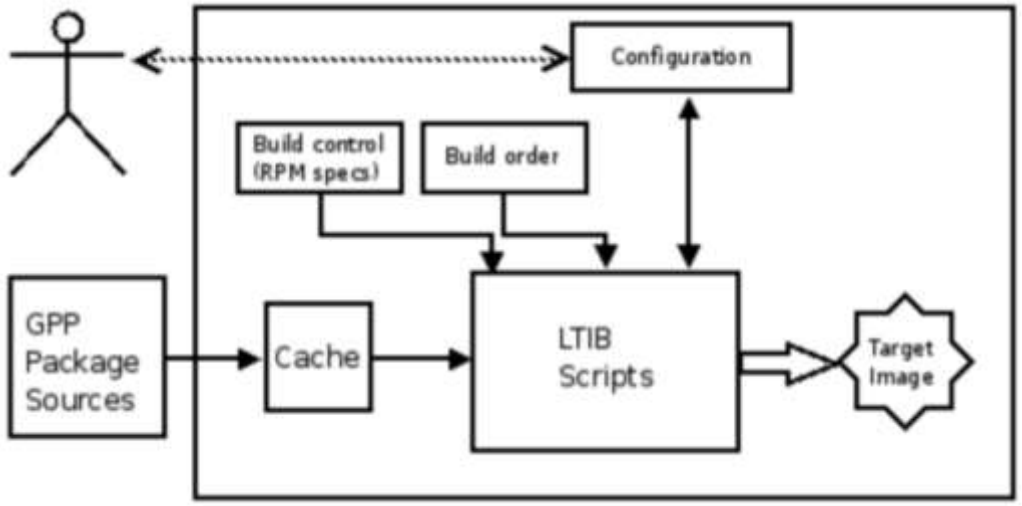

Figure 7: Flow diagram of LTIB

Features of LTIB

- Open source (GPL) and Runs on most popular Linux host distributions (x86 32/64 and some PPC)
- Command line interface, with curses configuration screens
- Support for multiple target architectures (PPC, ARM, Coldfire)
- Common root file system package set across architectures
- Supports building of boot loaders and kernel images
- Auto re-install/de-install of packages by changes in dependency tree
- Toolchain selectable at configuration time Configuration of the Linux kernel using it's native config language
- All packages are built as rpms and managed using rpm
- Target image files managed using a private rpm database per LTIB instance on the host Provides a means of source capture (patches) and auto update of spec files

- LTIB is meta-data only, all sources are pulled using http and locally cached in a common area per-host.

- Support for glibc and uclibc

- Provides a release mode, this encapsulates an LTIB project into an iso images that will not require network access.

Boot loader

In computing, booting (also known as "booting up") is a bootstrapping process that starts operating systems when the user turns on a computer system. The initial set of operations that the computer performs when power is switched on is take care by boot sequence [6]. The boot loader typically loads the main operating system for the computer. The operating system is stored on the server, and certain parts of it are transferred to the client using a simple protocol such as the Trivial File Transfer Protocol. After these parts have been transferred, the operating system then takes over control of the booting process. In DSP systems booted by another processor is host processor. The most notable systems with such a design are cell phones, modems, audio and video players, etc where a DSP and a CPU/microcontroller are co-existing.

Kernel Image

The kernel's responsibilities are managing the system's resources and provide the lowest-level abstraction layer for the resources that application software must control to perform its function. Operating system tasks are done differently by different kernels, depending on their design and implementation. Monolithic kernels execute all the code in the flat address space to increase the performance of the system, microkernels run most of the operating system services in user space as servers, to improve maintainability and modularity of the operating system. The kernel's primary purpose is to manage the computer's resources and allow other programs to run and use these resources. Kernel process management must take into account the hardware built-in equipment for memory protection. To run an application, a kernel typically sets up an address space for the application, loads the file containing the application's code into memory, sets up a stack for the program and branches to a given location inside the program, thus starting its execution.

Older versions of Windows and Mac OS both used co-operative multitasking but switched to pre-emptive schemes as the power of the computers to which they were targeted grew. The operating system might also support multiprocessing in that case, different programs and threads may run on different processors. A kernel for such a system must be designed to be re-entrant, meaning that it may safely run two different parts of its code simultaneously. To perform useful functions, processes need access to the peripherals connected to the computer, which are controlled by the kernel through device drivers. A kernel must maintain a list of available devices. This list may be known in advance and configured by the user or detected by the operating system at run time. Device management is a very opearting system specific, these drivers are handled differently by the kernel design, but in every kernel has to provide the I/O to allow drivers to physically access their devices through some port or memory location.

File System

A file system is a method of storing and organizing computer files and their data. Essentially, it organizes these files into a database for the storage, organization, manipulation, and retrieval by the computer's operating system. File systems are used on data storage devices such as a hard disks or CD-ROMs to maintain the physical location of the files. They provide access to data on a file server by acting as clients for a network protocol (e.g., NFS, SMB, or 9P clients), or they may be virtual and exist only as an access method for virtual data. Most file systems make use of an underlying data storage device that offers access to an array of fixed-size physical sectors, generally a power of 2 in size (512 bytes or 1, 2, or 4 KB are most common). The file system is responsible for organizing these sectors into files and directories, and keeping track of which sectors belong to which file and which are not being used. A file system can be used to organize and represent access to any data, whether it's stored or dynamically generated.

Prepration of the Target Images

The actual target image that is to be dumped into the system is configured, compiled and built using the Linux Target Image Builder. This tool is configured in the Ubuntu 9.04 Linux desktop environment. The images created for the target are u-boot.bin, uImage, and rootfs.jffs2 u-boot.bin: The Universal Bootloader, known as u-boot for

short. uImage: The Linux kernel image. rootfs.jffs2: JFFS2 formatted root file system to be stored in NAND flash.

Using the Linux Target Image Builder Platform In this step the target platform for which the image is to be created is selected as shown in Fig.8. Package Configuration Menu and Selction Menu cofiguration as shown in Fig.9. Here the packages are configured according to the system requirements. Creation of root file system and kernel image as shown in Fig.10. Creation of Bootloader is shown in Fig.11.

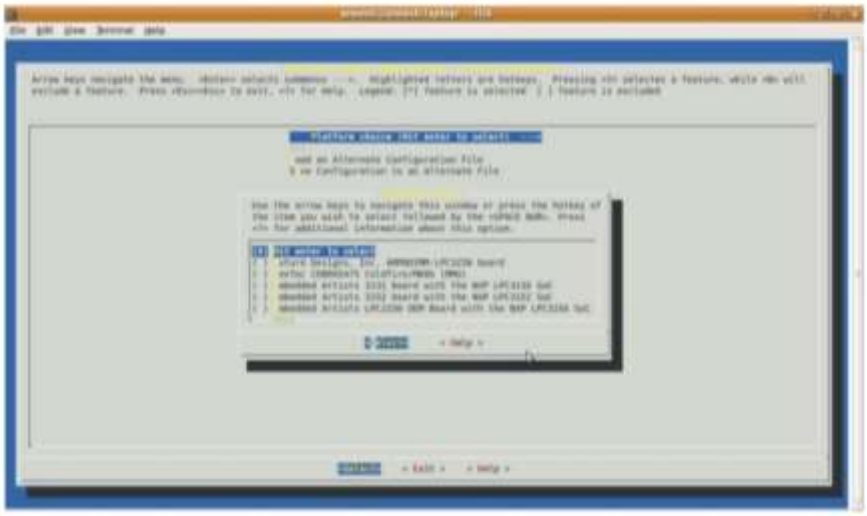

Figure8: Target Platform choice Menu

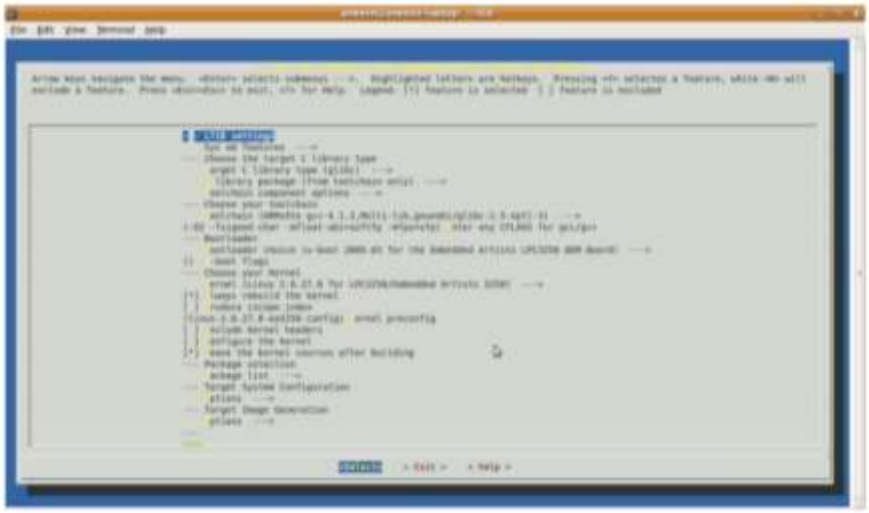

Figure 9: Package selection Menu

The newly created images for the target system will be created in the following locations Boot loader: /home/aneesh/ltib/rootfs/boot/u-boot.bin Kernel Image: /home/ aneesh//ltib/rootfs/boot/uImage File System : /home/aneesh/ltib/rootfs.jffs2 The complete compiled root file system is in the following path /home/aneesh/ltib/rootfs

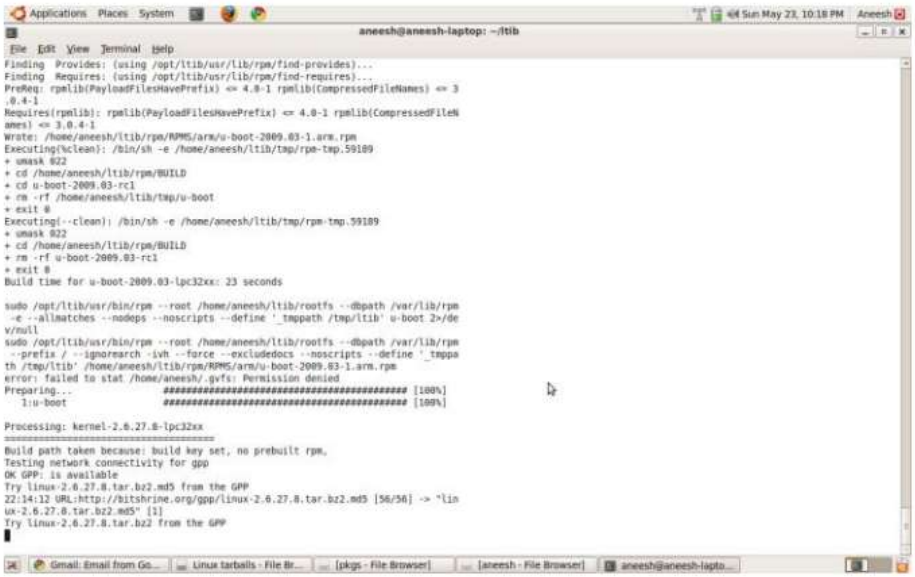

Figure10: Creation of Boot Loader

Figure 11: Building of kernel Image and File system

The images can be transferred to the target via UART using the Terminal emulator. Here the terminal emulator called Tera Term is used and the baud rate is fixed as 115200. The boot loader is loaded in this way and the Kernel image and the file system is 11 loaded from the FAT formatted USD memory stick. Now the kernel will be booted from the ARM plat form and the output can been viewed in the terminal. Simulation Outputs: The images are cross compiled and build using the LTIB and the compilation result is as shown below. Transfering the Bootloader Of the Target: The boot loader image is transferred to the target as shown Fig.12.

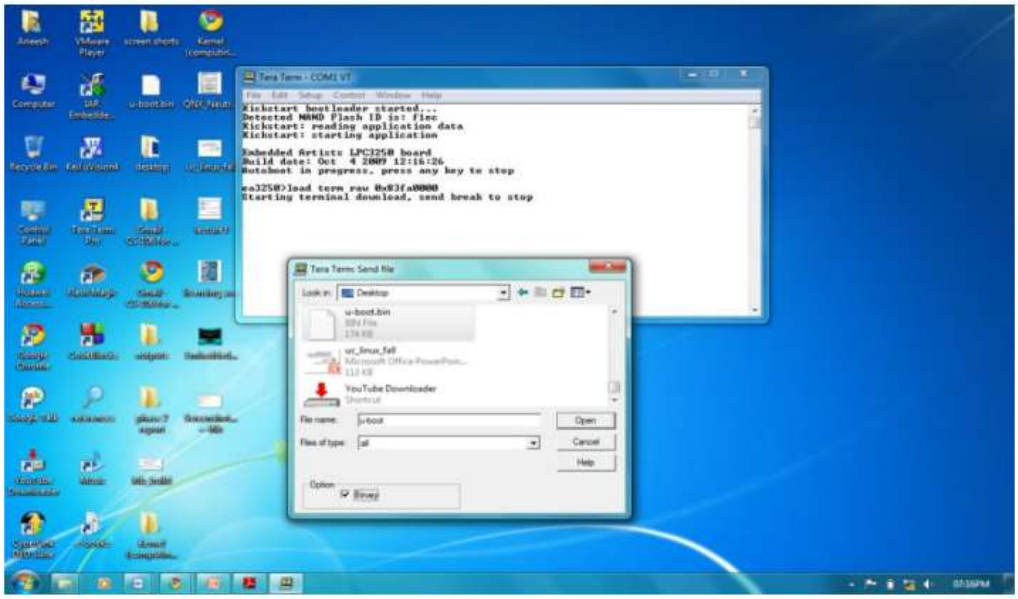

Figure 12: Transferring boot loader to the target

Linux Kernel Is Booting from the Target as ahown in Figure 13. Authentication system running from ARM9 as shown in Figure 14

Figure 13: Linux Kernel running on the ARM target board

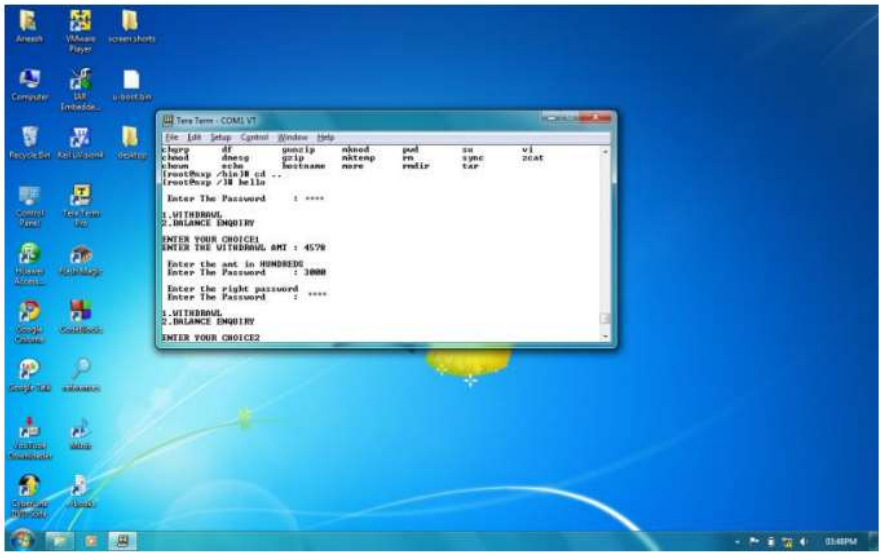

Figure 14: Simulation of an authentication system running from the ARM9

CONCLUSION

In this paper a secured system is designed using the ARM processor and the Linux kernel. Here the Linux kernel 2.6.27 is configured according to our requirement and the footprint of the kernel image is reduced to 1.56 MB as compared to the original size of 48.1MB. This is achieved because of the freely available Linux kernel source code and the freedom to modify it.

More over the Linux operating system highly secured and it is very difficult for unauthorized access. This Linux operating system is nowadays widely used in Embedded Systems because it supports a wide variety of platforms. Hence the development cost, time and complexity can be highly reduced by using the FOSS. This work can be extended to implement a fully functional system with low booting time which is an important factor in Embedded System running on Linux OS. The foot print of the kernel image should also be reduced to achieve the cost reduction of the product.

REFERENCES

[1] Son, Sang Hyuk, Ravi Mukkamala, and Rasikan David, "Integrating security and real-time requirements using covert channel capacity," IEEE Transactions on Knowledge and data Engineering, 2000 Vol. 12. no6, PP. 865-879, 2000.

[2] Steffen, Andreas, and Zürcher Hochschule Winterthur, "Secure Communications in Embedded Systems," CRC Industrial Information Technology Handbook 2004.

[3] Seal, David, ARM architecture reference manual. Pearson Education, 2001.

[4] Yan-Ling, Xu, Pan Wei, and Zhang Xin-Guo, "Design and implementation of secure embedded systems based on trustzone, " Embedded Software and Systems, ICESS'08. International Conference on. IEEE, 2008 July, pp. 136-141.

[5] Irvine, Cynthia, and Timothy Levin, "Toward a taxonomy and costing method for security services, " Computer Security Applications Conference Proceedings.IEEE, 1999,pp. 183-188.

[6] Narari, H, "Trusted Secure Embedded Linux: From Hardware Root of Trust to Mandatory Access Control," Proceedings of the Linux Symposium, Ottawa, Ontario, Canada. 2007.

[7] O'Flynn, Colin, and Zhizhang David Chen. "Chipwhisperer: An open-source platform for hardware embedded security research." In International Workshop on

Constructive Side-Channel Analysis and Secure Design, 2014, pp. 243-260.

[8] Papp, Dorottya, Zhendong Ma, and Levente Buttyan. "Embedded systems security: Threats, vulnerabilities, and attack taxonomy.", IEEE 13th Annual Conference on In Privacy, Security and Trust (PST), 2015 pp. 145-152.