Keeping High Availability of Connected End-point Devices in Smart Grid

Hiroshi Isozaki[1, 2], Jun Kanai[1]
[1] Corporate R&D Center, Toshiba Corporation,
Kawasaki, Kanagawa, Japan
{hiroshi.isozaki, jun.kanai}@toshiba.co.jp
[2] Graduate School of Media and Governance,
Keio University,
Fujisawa, Kanagawa, Japan

Shunsuke Sasaki, and Shintarou Sano
Center for Semiconductor Research & Development,
Toshiba Corporation,
Kawasaki, Kanagawa, Japan
{shunsuke.sasaki, shintarou.sano}@toshiba.co.jp

***Abstract*— Security is an important aspect to achieve Smart Grid success in terms of commercial deployment. Particularly, availability gets the highest priority in Smart Grids. For end-point devices, such as smart meters or concentrators, this must be true since they must always be working. We present LiSTEE^{TM} Recovery, an architecture for a fault tolerant system for end-point devices to monitor the status of the operating system and to recover even if they stop working due to unexpected behavior or cyber attack including zero-day attack. LiSTEE^{TM} Recovery provides further functions to prevent illegitimate memory modification and to notify a head-end system once a security incident occurs. We demonstrate a full implementation of LiSTEE^{TM} Recovery on a TrustZone capable ARM based processor. Our experiment shows that the performance degradation is small enough to be ignored. Furthermore, we observed that the cost of production and maintenance can be minimized.**

Keywords-Smart Grid, Smart Meter, Concentrator, Security, High Availability, TrustZone

I. INTRODUCTION

In Smart Grids, requirements about supporting various protocols and functions to network connected end-point devices, such as smart meters or concentrators, make their systems more complicated. Because a large quantity of source code is necessary to implement a complicated system in general, the risk of including vulnerability in the system increases. Moreover, since the devices are connected to home networks, the risk of devices of being attacked is high compared with legacy devices connected to managed network only. In fact, it is reported that smart meters from a variety of vendors were found to improperly handle malformed requests which could be exploited to cause buffer overflow vulnerability; allowing an attacker to cause a system to become unstable or freeze [1]. To keep devices secure under this situation, many security protocols and algorithms have been proposed to securely distribute a shared key between devices and head-end systems or to store privacy data in devices in a secure manner [2][3]. However, confidentiality and integrity are not enough to solve the security problem in Smart Grids. It is strongly desired for the devices to keep high availability since they must always be working to provide demand response service or to use consumption data for payment [4]. As only one vulnerability may cause the system to go down, it is very difficult to keep high availability in a complicated system. Furthermore, unlike interactive devices, such as PC or smart phone, it is difficult to expect that end users reset and restart devices once they freeze or hang since end users cannot recognize the status of the devices and cannot determine the device should be rebooted or not. Thus, how to keep the availability of the devices is a significant challenge in Smart Grids.

To address these problems, we propose LiSTEE^{TM} Recovery, an architecture for fault tolerant systems which automatically recovers from error status. To achieve this goal, LiSTEE^{TM} Recovery isolates a surveillance process observing the state of the system and recovery process which reboots the system when it detects the system freezes. In the LiSTEE^{TM} Recovery, surveillance and recovery processes run in an isolated secure environment while general purpose processes, including operating system, such as network or storage access run in a non-secure environment with hardware access control performed with respect to memory. Hence, a memory area where surveillance and recovery processes are arranged cannot be accessed by general purpose processes. As a result, even if the operating system is attacked and crashes, it becomes possible to prevent from interference in the surveillance and recovery processes.

The remainder of this paper is organized as follows. In Section-II, problems are defined. Section-III indicates background information. Section-IV and V propose framework and implementation of LiSTEE^{TM} Recovery. The evaluation is shown in the Section-VI; and the conclusion and future work are in at the end.

II. PROBLEM DEFINITION

In a legacy system, surveillance and recovery processes and their execution environment are monolithically configured. In other words, the reliability of surveillance and recovery processes depends on the reliability of their execution environment. In order to keep reliability high, a system needs to be implemented without vulnerability. In order to detect and eliminate vulnerability in source code, various testing methods have been proposed [5][6]. However, since end-point devices will be deployed without maintenance over a long period of time within Smart Grids, there is a large risk such devices continue operating without vulnerabilities being fixed even if those devices had no vulnerabilities at the time of shipping. Attackers may exploit the vulnerability, such as buffer overflow or malformed network input, in order to and cause the device to crash. To make matters worse, attackers are in a somewhat advantageous position in launching a large attack since the number of device vendors is limited and the software installed in the devices is uniform. Furthermore, attackers can reverse-engineer code without administrators noticing in order to find a vulnerability since, unlike a server application, devices are located at the user side. Therefore, when attackers find one vulnerability in a single device, they can exploit it on a lot of devices. Considering the above situation, the following problems are to be solved in order to keep high availability under a legacy system.

A. Difficult to Keep a High Level of Surveillance Continuity

In order to implement a complicated application program or a minor network protocol on the end point device, Linux will be used as a software execution environment. In Linux, the surveillance and recovery processes can be implemented as a user task executed on the operating system or as an interrupt handler in the operating system. When a surveillance target process is implemented as a user task running on the operating system then support functions in the operating system, such as the "cron" service in Linux, can be used to detect a failure of the user task and to automatically restart the target process. When the surveillance process is implemented as an interrupt handler in the operating system then sophisticated implementation is necessary compared to an application program; it is automatically and periodically called by a timer interrupt as long as the operating system works. Another legacy approach is implementing a monitoring and detecting mechanism in the operating system. For example, in order to find buffer overflow attacks, a protection element monitors system call frequencies, and if the frequencies are different from normal behavior, it can detect the attack [7]. However, the fundamental problem of a legacy approach is that there is no way to restart the process if the operating system itself crashes for some reason. Furthermore, the protection mechanism itself could be a target of the attack, as the result the protection mechanism could be invalidated. Thus, there is a large risk where devices in a Smart Grid breakdown and the attack may be able to cause a blackout to vast areas in the worst case. In order to prevent devices breaking down, they are required to provide a robust method to recover the system from failure in order to keep high level of availability. Still there are some existing hardware devices supporting a watchdog timer function which detects the status of the operating system and automatically reboots the system. Since not all devices support the function and it is difficult to implement complicated functions as described below inside it, a new approach is desired. To clarify the conditions, only a software failure including an attack is assumed in this paper. A physical fault, such as a hardware failure or loss of power, or a hardware attack, such as physically destroying devices or cutting cables are out of scope in this paper.

B. Difficult for an Administrator to Detect when Incident Occurs

End-point devices are connected with a head-end system through the network to provide a demand response service. When the devices detect an error status, such as a surveillance target process being stopped for an unknown reason, it is desirable for these devices to send a report to the head-end system so that an administrator can realize the situation and use the report to investigate the reason for the failure. However, for the same reason as described above, there is no way for devices to send a message to the head-end system if the operating system crashes. Even in such a case, it is desirable for devices to provide a method to send a message to acknowledge the error situation to the system administrator. Besides notification of the error situation to the system administrator, software update function is also desirable. However, since many existing hardware devices have already supported a secure firmware update function and its method highly depends on each device, it is out of scope in this paper.

In addition to the problem described above, the following business problem needs to be considered when introducing a new architecture to the market.

C. Development and Production Cost

Cost is an important aspect in evaluating the proposed security architecture. When implementing an end-point device, if the new security architecture requires a complete rebuild of software, the architecture will never be introduced to the market. Thus, it is desirable to reuse existing software asset, such as libraries, middleware and applications as much as possible to minimize the development cost including verification cost. Specifically in Smart Grids, the verification cost is large since reliability is strongly required. Besides the development cost, we need to consider the cost per device. One approach to solve the problems described above is to utilize the dedicated hardware security chip. However, since the chip is sometimes very expensive, it causes a rise of production cost per device. Therefore, it is also desirable to use widely available existing commodity hardware to minimize production cost.

III. BACKGROUND (TRUSTZONE)

In this section, we provide background information on the hardware technologies leveraged by LiSTEE^{TM} Recovery.

TrustZone is a hardware security function supported by a part of ARM processor [8][9]. General ARM processor defines two modes, user mode and privileged mode. In privileged mode, execution of all instructions and access to all memory regions are allowed while in user mode availability of instructions and accessibility of memory regions are restricted. In general system, operating system is executed in privileged mode while application programs are executed in user mode. In addition to the two modes, a TrustZone enabled ARM processor supports two worlds which are independent of the modes. One is secure world for the security process and the other is non-secure world for everything else. Fig. 1 shows the relationship between worlds and modes conceptually. The processor is executed by selectively switching the worlds if needed. For example, it is assumed that key calculation process is executed in secure world while all other general processes, such as

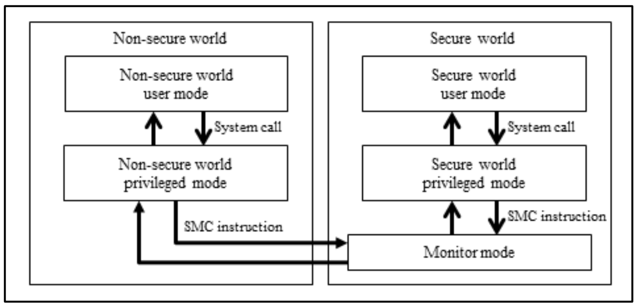

Figure 1. Mode and world in ARM.

storage access or network access are executed in non-secure world. Moreover, using TrustZone enable hardware, it is possible to make a system where a process running in secure world can access all system resources while a process running in non-secure world can access a part of system resources only. For example, when used in combination with the TrustZone Address Space Controller (TZASC), access to memory can be restricted for a process running in non-secure world even if the process runs in privileged mode to install an access control policy on TZASC.

TrustZone provides a dedicated instruction, the Secure Monitor Call (SMC) instruction, to transit between the worlds. As soon as the SMC instruction is called, the processor switches into monitor mode. A software program running in monitor mode saves a context of the program running in the current world on the memory and restores a context of the program running in the previous world, then changes the world, and finally executes the program running in the previous world. Besides the SMC instruction, hardware exceptions can be configured to cause the processor to switch into monitor mode.

IV. FRAMEWORK OF LISTEE^{TM} RECOVERY

LiSTEE^{TM} Recovery provides a method for an end-point device to automatically recover from an error status. It also provides a high level of memory protection mechanism. Hence, the recovery process is securely executed without interference. Fig. 2 shows the entire architecture of LiSTEE^{TM} Recovery. LiSTEE^{TM} Recovery consists of three components, Normal OS, LiSTEE^{TM} Tracker Application (LiSTEE^{TM} TA), and LiSTEE^{TM} Monitor.

- Normal OS: Operating system which executes eneral purpose processes, such as storage access or network communication. It is executed in non-secure world. All applications implementing smart meter functions or concentrator functions run on this operating system.
- LiSTEE^{TM} Tracker Application (LiSTEE^{TM} TA): Surveillance and recovery processes executed in secure world. LiSTEE^{TM} TA includes three modules: Watcher module, Recovery module, and Notification module. The Watcher module is an entry point of LiSTEE^{TM} TA. It is executed periodically by a timer interrupt through LiSTEE^{TM} Monitor. Whenever it is called, it investigates the status of Normal OS. If it detects Normal OS is not working, it calls Recovery module to reboot the system. Otherwise, it calls SMC instruction to switch to Normal OS. Moreover, the Notification module is called before Recovery module reboots the system. It sends a message to notify that the system is about to reboot to the head-end system through network.
- LiSTEE^{TM} Monitor: LiSTEE^{TM} Monitor is a program running in the monitor mode. It initializes configurations of TrustZone related hardware when booting the system. It also provide context switching function between worlds in hardware interrupt handler and SMC handler. Regarding the configuration of the hardware, it sets configuration register of timer interrupt so that hardware interrupt handler of LiSTEE^{TM} Monitor is called when timer interrupt is generated. Moreover, LiSTEE^{TM} Monitor manages the access control policy and installs the policy on TZASC. Regarding context switching, the SMC handler in LiSTEE^{TM} Monitor is executed when the SMC instruction is called and it transits from secure world to non-secure world. In contrast to the SMC handler, timer interrupt triggers transit from non-secure world to secure world based on the initializing configuration.

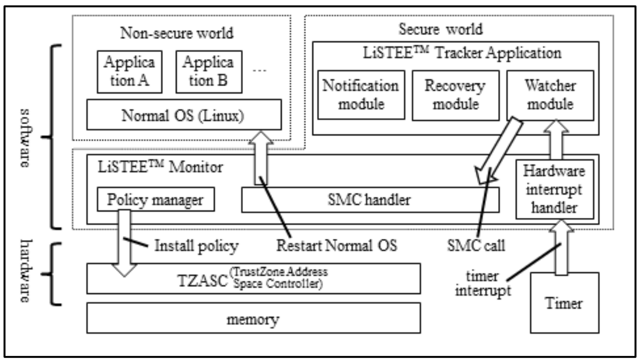

Figure 2. System Architecture of LiSTEE^{TM} Recovery.

The primary feature of LiSTEE^{TM} Recovery is to provide a method for the end-point device to detect the status of Normal OS and to recover it even if Normal OS crashes or stops working. Furthermore, it provides two additional functions. One is to enhance the security protection for LiSTEE^{TM} Monitor, LiSTEE^{TM} TA and Normal OS against attacks. The other is sending a message to the head-end system when an incident occurs. The details of these functions are described below.

A. *Periodical Surveillance and Recovery*

When booting the system, LiSTEE^{TM} Monitor is executed after executing initial program. Then, LiSTEE^{TM} Monitor loads and executes LiSTEE^{TM} TA and Normal OS respectively. While executing Normal OS, whenever the timer interrupt occurs, the processor jumps to the hardware interrupt handler in LiSTEE^{TM} Monitor. The hardware interrupt handler context switches from non-secure world to secure world and calls LiSTEE^{TM} TA. Specifically LiSTEE^{TM} Monitor saves a context of Normal OS to memory and restores a context of LiSTEE™ TA, then changes the world and finally calls the Watcher module of LiSTEE™ TA. The Watcher module checks the status of Normal OS. If it judges that Normal OS is not working, the Watcher module calls the Recovery module which reboots the system. Otherwise it calls SMC instruction. Then, SMC handler in the LiSTEE™ Monitor is executed. It first context switches from LiSTEE™ TA to Normal OS, and restarts Normal OS at the point just before the timer interrupt occurred. While executing LiSTEE™ Monitor and LiSTEE™ TA, the execution of Normal OS is suspending. That is, Normal OS continues to be processed as if nothing was executed during the execution of LiSTEE™ TA. Fig. 3 shows the flowchart of periodical surveillance and recovery process.

There are many ways for the Watcher module to determine whether Normal OS is working or not. One of the methods is to check the data area of Normal OS. In general, when an operating system is working, there must be a certain data area which is updated regularly. By checking this data area, it is possible for the Watcher module to judge whether Normal OS is working or not.

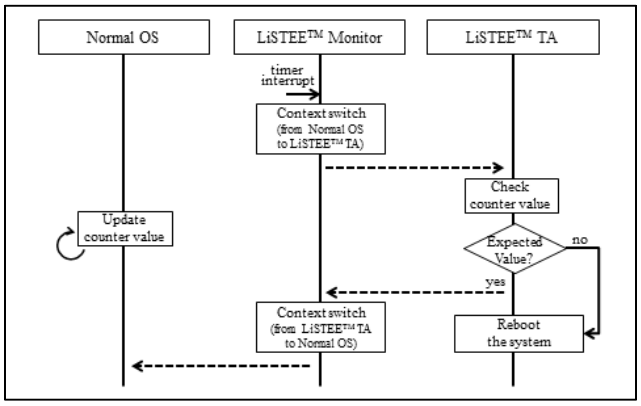

Figure 3. Flowchart of periodical surveillance and recovery.

B. Memory Protection

LiSTEE™ Recovery provides two memory protection mechanisms. Fig. 4 shows how these memory protection mechanisms work. One is protection for the kernel area of Normal OS. To realize this protection, LiSTEE™ Monitor

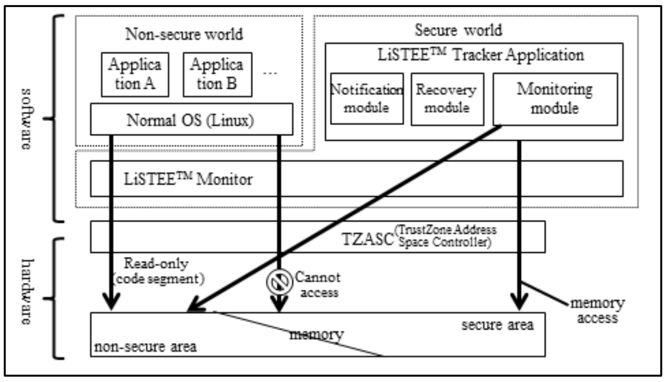

Figure 4. Memory protection mechanism.

provides virtual read-only memory. In virtual read-only memory, Normal OS running in non-secure world cannot overwrite the content on the memory while LiSTEE™ TA and LiSTEE™ Monitor running in secure world can access it using ordinary SRAM as the memory which is, of course, physically writable memory. In general, when a program is loaded into memory, a data region (data segment) and a code region (code segment) are assigned. In the initial state before booting the system, all regions are allowed to be accessed from Non-secure world by default. In order to allow the boot loader to write the code segment into the memory, LiSTEE™ Monitor leaves the memory region as is until the code segment is loaded. Just after executing the kernel of Normal OS, LiSTEE™ Monitor sets the memory region as read-only for kernel code segment of Normal OS. As the result, even Normal OS is prohibited from overwriting its own code segment.

Another mechanism is protection for LiSTEE™ Monitor and LiSTEE™ TA. To realize this protection, LiSTEE™ Monitor installs an access control policy where Normal OS cannot access the memory area allocated to LiSTEE™ Monitor and LiSTEE™ TA while LiSTEE™ TA and LiSTEE™ Monitor can access all areas when booting the system. For this policy, LiSTEE™ Monitor and LiSTEE™ TA can be protected from illegitimate falsification by Normal OS, even if Normal OS is attacked and under control of an attacker.

C. Message Notification

LiSTEE™ Recovery provides a function to notify the head-end system that Normal OS has stopped working and is rebooting the system by sending a message through network. The Notification module has the role of sending a message. Although Normal OS has network connectivity function, such as TCP/IP stack, LiSTEE™ TA cannot use the function since it is not working when sending a message. Thus, LiSTEE™ TA supports network connectivity function including network application, network protocol stack and network driver.

V. Prototype Implementation

We used ARM C/C++ Compliler 5.01 as a compiler to build LiSTEE™ Monitor and LiSTEE™ TA. We used gcc 4.4.1 to build Linux 3.6.1 as Normal OS. We chose Motherboard Express uATX with CoreTile Express A9x4 processor which supports TrustZone as an execution environment.

Regarding memory map, from 0x48000000 through 0x4A000000 is assigned for SRAM, and from 0x60000000 through 0xE0000000 is assigned for DRAM. Tab. I shows memory map with access control policy of the memory. In Tab. I, Normal OS (code) indicates Linux kernel code. Normal OS (data) includes Linux data, application code and application data. For clarification, Read/Write access is applied from Non-secure world for the area not described in Tab. I.

In order for the LiSTEE™ Monitor to install an access control policy on TZASC, the start address and the size of each memory region are predefined. After the boot loader loads Linux at the predefined value, LiSTEE™ Monitor installs the access control policy on TZASC. As shown in Tab. 1 the access to the memory regions allocated to LiSTEE™ Monitor, LiSTEE™ TA and code segment of Normal OS is restricted for the Normal OS running in non-secure world while the access to the region allocated to the data segment of Normal OS and shared memory is not. For clarification, LiSTEE™ Monitor and LiSTEE™ TA run in secure world can access all regions. Furthermore, since LiSTEE™ Monitor sets the configuration registers of TZASC to prohibit Normal OS from accessing them, Normal OS cannot change this configuration.

TABLE I. MEMORY MAP

Data	Start Address	Size	Access Policy (Non-sec world)
Vector table + Initialization code + LiSTEE™ Monitor + LiSTEE™ TA	0x48000000	0x01B00000	Cannot Access
Normal OS (code)	0x60000000	0x002FE000	Read Only
Normal OS (data)	0x602FE000	0x3EF02000	Read/Write
Shared memory	0x9F200000	0x00C00000	Read/Write

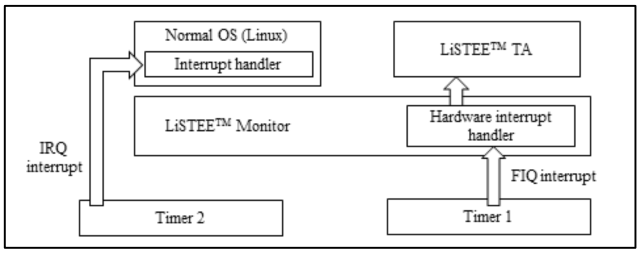

Figure 5. Assignment of timer interrupt.

Fig. 5 shows the assignment of the timer interrupt. We allocated a timer interrupt caused by timer (timer 1) to Fast Interrupt Request (FIQ) and timer interval was set to 1 second. The FIQ interrupt is handled by hardware interrupt handler in LiSTEE™ Monitor, and then it calls LiSTEE™ TA, as the result LiSTEE™ TA is periodically called. We used another timer (timer 2) and allocated to Interrupt Request (IRQ), and timer interval was set to 4 milliseconds. The IRQ interrupt is handled by interrupt handler in Linux. Since Linux assumes timer interrupt is allocated to IRQ, it is not necessary to modify the Linux source code to adopt LiSTEE™ Monitor. Tab. II shows a configuration of hardware interrupt. We configured Secure Configuration Register (SCR) and Current Program Status Register (CPSR) so that FIQ handler of LiSTEE™ Monitor is called when FIQ handlers occurs while IRQ handler in Linux is called when IRQ interrupt occurs during executing Linux. In the same manner as the configuration register of TZASC, we set FIQ and IRQ configuration registers to prohibit Normal OS from accessing them. Thus, Normal OS cannot change the configurations. For example, Normal OS cannot mask FIQ to stop timer 1, or cannot change the timer interval of timer 1.

In order to determine whether Linux is working or not, we made a small application program which runs on Linux and communicates with LiSTEE™ TA. Shared memory is used to exchange data between LiSTEE™ TA and Normal OS. The application program writes a counter value into the shared memory periodically. Then LiSTEE™ TA reads the counter value from the shared memory. When Normal OS is crashed, the application program cannot update the counter value. If the counter value is not updated in certain amount of time or the counter value is not an expected value, LiSTEE™ TA determines that Normal OS is not working.

TABLE II. RELASHONSHIP BETWEEN WORLD AND INTERRUPT

World when interrupt occurs	Interrupt	Jumps to
Non-secure world	FIQ	Hardware interrupt handler (FIQ handler) in LiSTEE™ Monitor
	IRQ	IRQ handler in Normal OS (Linux)
Secure world	FIQ	Pending FIQ
	IRQ	Pending IRQ

When LiSTEE™ TA determines that Linux is not working, it sends the head-end sever a message. In order to send a message to the head-end system when LiSTEE™ TA detects that Linux is not working, we ported a network driver and UDP/IP stack to LiSTEE™ TA. We defined a proprietary protocol and data format over UDP to notify the head-end system that LiSTEE™ TA starts reboot the system.

VI. EVALUATION

In this section, we describe the result of the evaluation in terms of security to verify the problems of the legacy system defined in Section-II can be solved. Performance and cost analysis of LiSTEE™ Recovery is also described below.

A. Security Analysys

1) Surveillance and Recovery: LiSTEE™ Recovery can recover from a failure to reboot the system even if Normal OS crashes. The reason for the crash could be a software bug or a cyber attack including zero-day attack caused by unknown vulnerabilities. In either case, since hardware timer interrupt continues working regardless of the state of Normal OS, LiSTEE™ TA is always periodically called and can detect a failure of Normal OS. At the next level, it is desirable to detect the failure as soon as possible. Detection time depends on how frequently LiSTEE™ TA checks the status of Normal OS. Since the execution time of LiSTEE™ TA and context switching by LiSTEE™ Monitor is very short, LiSTEE™ Recovery can detect the crash of Normal OS very quickly. Some attackers may continue to attack just after rebooting the system. One possible approach to a countermeasure for the attack is to let LiSTEE™ TA have the minimum function like the "safe mode", we have not implemented that though.

2) Attack Prevention: The proposed system provides two levels of attack prevention mechanism. The first level is to prevent Normal OS from illegitimate modification. When an attacker gains full control of Normal OS to misuse the vulnerability, the attacker may overwrite the code segment of Normal OS to directly overwrite the memory. In fact, many vulnerabilities (e.g., CVE-2013-4342, CVE-2013-1969, and CVE-2008-1673) allowing a remote attacker to execute arbitrary code are reported [10]. In the case of Linux, for example, once arbitrary code is executed with an administrator privilege by an attacker, it is possible for the attacker to overwrite an arbitrary area of code segment through /dev/mem, resulting in system crash or misbehavior. Overwriting the code segment in memory is difficult in general though it is relatively easy in the case of end-point devices since hardware configuration is fixed. As the result, system may go down. However, since LiSTEE^{TM} Monitor sets the access control of the memory region for the code segment of Normal OS as read-only, and its configuration can be changed only from secure world, it is impossible for Normal OS to overwrite the code segment of Normal OS. An advantage is the protection does not cause any side effects. Since data segment is used to store the state of the program, Normal OS updates the content of data segment frequently during its execution. In contrast to the data segment, since code segment is used to store program code, it is not expected to update its content after booting the system. Particularly because devices, such as smart meters or concentrators are not expected to change their function after being deployed, the dynamic update function is not required. Thus, this protection mechanism can protect Normal OS from illegitimate modification without side effects. The second level is to protect LiSTEE^{TM} Monitor and LiSTEE^{TM} TA from illegitimate modification and suspension. Since the first level of protection is effective on code segment of Normal OS only, an attack which overwrites a data segement cannot be prevented. Thus, there are still possibilities that control of Normal OS is gained by an attacker. Even in such cases, thanks to TZASC, since Normal OS is prohibited from overwriting the content of memory where LiSTEE^{TM} TA and LiSTEE^{TM} Monitor are allocated, illegitimate modification is prevented. Moreover, since the interrupt configuration register is accessible only from secure world, there is no way for Normal OS to stop the timer interrupt.

3) System Reliability: In a legacy system, one single bug could affect the entire system to cause a critical failiure. Considering a defensive viewpoint, the entire system including operating system must be bug free to archieve high availability ideally. However, it is not practical to build a system without bugs in a complicated system. In fact, Linux 3.6.1 consists of over 15 millions of lines of code and a lot of new bugs causing critical crash are reported frequently (e.g., CVE-2013-4563, CVE-2013-4387, and CVE-2012-2127) even though it is carefully reviewed by many professionals [10]. Thus, the smaller the critical component that has to be robust within a system, the better. In the case of LiSTEE^{TM} Recovery, the critical components corresponds to LiSTEE^{TM} TA and LiSTEE^{TM} Monitor. In contrast to Linux, the code size of LiSTEE^{TM} Monitor and LiSTEE^{TM} TA is relatively small. In fact the volume of source code for LiSTEE^{TM} Monitor is about 700 lines and its code and data size are 2.1 KB and 1.6 KB respectively. Similarly the volume of source code of LiSTEE^{TM} TA is about 41200 lines and its code and data size are 1.09 MB. Compared to the volume of source code of Linux, the risk where LiSTEE^{TM} Monitor and LiSTEE^{TM} TA includes bugs is small.

4) Response to Failure: The Notification module in LiSTEE^{TM} TA sends a message to the head-end server just before rebooting the system. The message, which tells which the particular devices are about to reboot, is sometimes useful information for administartors. For example, if messages are sent by devices having a particular software version number, the reboot could be caused by an attack which aims at the vulnerability specific to the software. If messages are sent by devices located in one particular network, the reboot could be caused by a network worm distributed in the specific network. Although LiSTEE^{TM} Recovery cannot prevent an attack in advance, the notification feature can help the administrator to investigate the reason of the failure during or after the incident. The attackers try to block sending the message to circumvent the notification. However, Normal OS cannot interfere with Notification module in sending a message to the head-end server since Notification module is executed inside LiSTEE^{TM} TA. Moreover, since LiSTEE^{TM} TA is processed in an isolated environment from Normal OS, security processes, such as encrypting a message, are easy to implement in LiSTEE^{TM} TA. In the next step, it is possible to include a firmware update feature to implement functions receiving data from the head-end system and writing the data into the file system to extend the function of Notification module. In combination with “safe mode” described above, this function is effective against a countinuous attack which occurs just after the system recovers.

B. Performance Analysis

As well as the implementation environment, we used Motherboard Express uATX which contains ARM Cortex-A9x4 processor running at 400MHz as an experimental environment. The size of level 1 instruction cache, level 1 data cache, and level 2 cache are 32 KB, 32 KB, and 512 KB respectively. It contains 1 GB DRAM as the main memory and we assigned the same memory map described in Section-V.

First, we measured the execution time of LiSTEE^{TM} TA during execution of Normal OS. Precisely, the time period from the beginning of hardware interrupt handler in LiSTEE^{TM} Monitor through to the execution of the SMC instruction. Without calling the Notification module, the average time is 1.7 microseconds over 10,000 trials. However, if the Notification module is called, the average time is 4.1 milliseconds over 10,000 trials. Note that the Notification module is called when rebooting the system which is rarely occurs. Thus, this performance overhead is no problem.

Next, we measured the performance degradation of Normal OS. Since the execution of Normal OS is suspended during execution of LiSTEE™ TA, the performance of Normal OS degrades in any case. The total time of Normal OS suspension time depends on the frequency where LiSTEE™ TA is called. There is a tradeoff between the performance degradation of Normal OS and the delay for detecting the crash of Normal OS. When the frequency is increased, the performance degradation of Normal OS is also increased. On the other hand, when the frequency is decreased, the delay for detecting the crash of Normal OS becomes larger. To measure the performance degradation, we used dhrystone as a benchmark program [11].

Fig. 6 shows the result of the experiment. The bar graph shows dhrystone score and the line graph shows the performance degradation. The performance is better as the score value is high. Each bar shows timer interval which LiSTEE™ TA is called and its value is default (never called), 5 seconds, 1 second, 0.2 seconds and 0.04 seconds respectively. When timer interval was set to 5 seconds, the performance degradation was suppressed within 0.001 %. Even if the interval was set to 0.04 seconds, the performance degradation was less than 0.2 %. The result shows that although there is a trade-off between performance degradation of Normal OS and detection rate logically, the performance degradation can be ignored in practical even if the frequency where LiSTEE™ TA is called is increased.

C. *Cost Analysis*

1) Development Cost: LiSTEE™ Recovery does not require any modification to Linux to run it as Normal OS on LiSTEE Monitor. Thus, in terms of application developer's cost, since developpers can reuse all existing programs including libraries, middlewares, and applications running on Linux, no additional develping cost is necessary.

2) Production Cost: LiSTEE™ Recovery is software based technology and no additional hardware except TrustZone capable ARM processor and address space controller is required. Today TrustZone capable processors are widely available. In fact, all ARM Cortex A series processors support TrustZone. Therefore, the additional cost is mitigated. As the result, developing cost per device can be minimized.

3) Maintainance Cost: It is assumed that tremendous number of devices are deployed in the field in Smart Grid. Specifically in the case of cyber attack, since many devices could be a target of the attack and the attack could be done in a very short period of time through network, it is not practical for field service engineers to physically visit each site and reboot them in terms of both cost and time. The auto recovery feature of LiSTEE™ Recovery mitigates this problem. Moreover, the report is sent to the head-end server once the device reboots. This function contributes in reducing the cost of trouble shooting. Thus, LiSTEE™ Recovery provides opportunity to reduce maintainance cost compared with legacy systems.

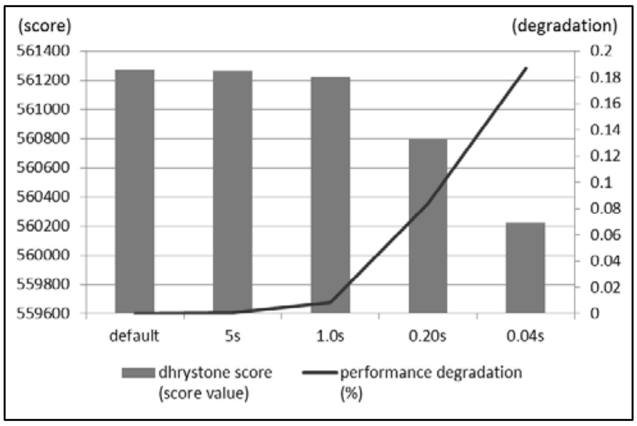

Figure 6. Result of performance degradation.

VII. Related work

To recover from an operating system failure, various approaches have been proposed.

The simplest approach is including the recovery mechanism inside operating system. One method is to use NMI as a watchdog timer [12]. Non-maskable Interrupt (NMI) is a processor interrupt that cannot be ignored. When NMI is generated, NMI handler implemented inside operating system is called regardless of the status of the operating system. Thus, NMI can be used as surveillance and recovery process to implement NMI handler so that it detects whether operating system hangs or not. Although NMI is easy to use for watchdog timer as it has already been implemented in Linux, it is vulnerable because NMI handler could be invalidated to overwrite the code segment of the operating system. Furthermore, it is not anticipated to implement a rich application in an interrupt handler, such as network communication function or data encryption function, it is difficult to realize the notification function.

Another approach to recover from the failure is to check the status of the operating system from outside using virtualization technology. It is easy to realize resource isolation environment by utilizing virtualization technology. Karfinkel developed trusted virtual machine monitor (TVMM), on which general-purpose platform and special-purpose platform executing security sensitive process run separately and concurrently [13]. The libvirt project develops a virtualization abstraction layer including a virtual hardware watchdog device [14]. To cooperate with the watchdog daemon installed in guest OS, a virtual machine monitor can notice that the daemon is no longer working when periodically trying to communicate with it. Although virtualization technology is widely deployed in PC-based systems, it is difficult to implement it in embedded devices as less hardware devices support it. Moreover, since the volume of source code for virtual machine monitor (VMM) tends to become large, the risk where VMM includes bugs also becomes large. To overcome the restriction, Kanda developed SPUMONE, which a light weight virtual machine monitor designed for working on embedded processors [15]. It provides a function to reboot the guest OS. However, SPUMONE does not provide memory protection mechanism between virtual machine monitor and the guest OS (Normal OS). Thus, it is vulnerable to an attack on the virtual machine monitor from guest OS.

VIII. CONCLUSION AND FUTURE WORK

In this paper, LiSTEE^{TM} Recovery works effectively to resist critical bugs or attacks including zero-day causing system crash in order to keep availability of end-point devices. The performance evaluation is presented to show that the degradation of the existing system is small enough. Considering the deployment in the market, we show that the development cost and production cost can be minimized. Moreover, it can save maintenance cost.

Future work includes the resistance to sophisticated attacks. One possible attack is that an attacker illegitimately modifies the shared memory area to fake as if Normal OS works correctly while almost all Normal OS functions actually stop. As the result, LiSTEE^{TM} TA misunderstands that Normal OS works correctly. One approach to solve this attack is to implement LiSTEE^{TM} TA so that it itself checks the status of Normal OS without the support of an application program running on Normal OS. For example, whenever Normal OS is running, it must update a certain data area, such as page tables or process tables. Therefore, LiSTEE^{TM} TA can determine Normal OS is working or crashed to monitor the data area. An advantage of LiSTEE^{TM} is that it is impossible for Normal OS to reverse-engineer and to tamper an algorithm of LiSTEE^{TM} TA because of memory protection mechanism. Thus, an attacker cannot know how to compromise Normal OS producing misleading information. We have not implemented this though. Another possible attack is damaging file system locating Normal OS. Network boot can be a solution where LiSTEE^{TM} TA downloads a small rescue program from the head-end system when it fails booting.

REFERENCES

[1] C4 Security. “The Dark Side of the Smart Grid”. [Online]. Available: http://www.c4-security.com/The%20Dark%20Side%20of%20the%20Smart%20Grid%20-%20Smart%20Meters%20%28in%29Security.pdf [Accessed 19 Feb 2014]

[2] D. Forsberg, Y. Ohba, B. Patil, H. Tschofenig, and A. Yegin, “Protocol for carrying authentication for network access”, IETF RFC 5191 [Online]. Available: http://tools.ietf.org/html/rfc5191 [Accessed 19 Feb 2014]

[3] F. Zhao, Y. Hanatani, Y. Komano, B. Smyth, S. Ito, and T. Kamibayashi, “Secure authenticated key exchange with revocation for smart grid”, The third IEEE PES Conference on Innovative Smart Grid Technologies (ISGT 2012), IEEE Power & Energy Society (PES), Jan 2012, pp.1-8.

[4] H. Khurana, M. Hadley, N. Lu, and D. A. Frincke, "Smart-grid security issues", IEEE Security & Privacy, vol.8, Issue.1, Jan-Feb 2010, pp.81-85.

[5] K. Li, "Towards Security Vulnerability Detection by Source Code Model Checking", Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on, Apr 2010, pp.381-387.

[6] M. Rinard, C. Cadar, D. Dumitran, D. M. Roy, and T. Leu, "A Dynamic Technique for Eliminating Buffer Overflow Vulnerabilities (and Other Memory Errors)", Computer Security Applications Conference, 2004. 20th Annual, Dec 2004, pp.82-90.

[7] S. M. Varghese and K. P. Jacob, "Anomaly Detection Using System Call Sequence Sets", Journal of Software, Vol.2 No.6, Dec 2007, pp.14-21.

[8] ARM. “ARM Security Technology” [Online]. Available http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf [Accessed 19 Feb 2014]

[9] T. Alves and D. Felton, “TrustZone: Integrated Hardware and Software Security”, Information Quarterly, Vol.3, No.4, 2004, pp.18-24.

[10] MITRE. “Common vulnerabilities and exposures” [Online]. Available http://cve.mitre.org [Accessed 19 Feb 2014]

[11] ARM. “Dhrystone Benchmarking for ARM Cortex Processors” [Online]. Available http://infocenter.arm.com/help/topic/com.arm.doc.dai0273a/DAI0273A_dhrystone_benchmarking.pdf [Accessed 19 Feb 2014]

[12] A. Kleen, “Machine check handling on linux”, Technical report, SUSE Labs, Aug 2004 [Online]. Available http://halobates.de/mce.pdf [Accessed 19 Feb 2014]

[13] T. Garnkel, B. Pfa, J. Chow, M. Rosenblum, and D. Boneh, “Terra: A virtual machine-based platform for trusted computing”, In Proceedings of the Symposium on Operating System Principles, Oct 2003, pp.193-206.

[14] "libvirt - the virtualization API." [Online]. Available: http://libvirt.org [Accessed 19 Feb 2014]

[15] W. Kanda, Y. Yumura, Y. Kinebuchi, K. Makijima, and T. Nakajima, "SPUMONE: Lightweight CPU Virtualization Layer for Embedded Systems", In Proc. of Embedded and Ubiquitous Computing, Dec 2008, pp.144-151.