**INDEX**

1. **INTRODUCTION** 
   1. **Description**

Hospitalized patients whose physiological status requires close attention can be constantly monitored using IoT-driven, noninvasive monitoring. This type of solution employs sensors to collect comprehensive physiological information and uses gateways and the cloud to analyze and store the information and then send the analyzed data wirelessly to caregivers for further analysis and review. It replaces the process of having a health professional come by at regular intervals to check the patient’s vital signs, instead providing a continuous automated flow of information. In this way, it simultaneously improves the quality of care through constant attention and lowers the cost of care by eliminating the need for a caregiver to actively engage in data collection and analysis.

The rising cost of healthcare, the increase in elderly population, and the prevalence of chronic diseases around the world urgently demand the transformation of healthcare from a hospital-centered system to a person-centered environment, with a focus on citizens’ disease management as well as their wellbeing. The development of personal mobile devices such as smartphones and tablets is helping establish a model of mobile health (mHealth) that can facilitate a continuum of person-centered care by relying on these mobile devices as a medium of sensing, interaction, and communication. Although, smartphones are embedded with an array of sensors that can track a user’s motion, location, activity, and so forth, these devices still lack the capability to collect fine-grain information of a user’s bodily health. A wide array of wearable devices has recently been developed to extend the capabilities of mobile devices, especially in the area of body and behavior sensing

* 1. **Problem Formulation**

// not edited

It is believed that Linux systems are more protected and secure than Microsoft Windows. It is also a known fact that a very small amount of malicious code has been developed to compromise Linux systems whereas the code written to compromise a Windows system is extensive. There are a number of legitimate reasons for this. Linux systems have never been as popular or widely used as Windows systems have. As a result, attackers have targeted Windows systems over the years for the simple reason that the infection will affect a large amount of users. Also, Linux systems support multiple user environments wherein different users have different access rights and privileges with respect to the system. As a result, a piece of malicious code has to gain high privileges to perform actions that may severely damage the system. Lastly, as Linux is an open-source operating system, developers can quickly patch a vulnerability in the system much before it causes any serious harm to it or spreads to other systems.

However, in spite of all the above-mentioned advantages, a Linux system too, can be compromised by an adversary. A classic example of a security attack on a Linux system is gaining unauthorized access to it through system calls. This is only possible if the adversary’s software gains root access to the system. Once the adversary has gained root access to the system, he can not only traverse the file system, but also read and manipulate sensitive, confidential data contained in the system. Certain network and file system related attacks can also be performed without needing root access to the system. As the popularity of Linux systems is on the rise, the number of malware being developed to compromise them is also increasing.

Misuse of system calls to compromise a system is a potential threat and to avoid it, sandboxing is used. In computer Security, Sandboxing is a mechanism that allows unknown or untrusted code into the system, and yet does not let it damage the system. A sandbox isolates the running program from the rest of the system by imposing restrictions on network resources and file system access, and keeps the host system safe. A sandbox system heavily restricts the program from inspecting the host or reading from the input device. Sandboxes can be used to analyse applications or test their legitimacy by allowing them to execute in a highly controlled environment so that they cannot cause any harm to the rest of the system. A sandbox can restrict the operations of a code by allowing it only as many permissions as required, without granting additional permissions that can be used to harm the system. Thus, sandboxing provides security by isolating a software so as to prevent malicious code from abusing the system. This can be done by specifying security policies for the applications or isolating each application in a separate virtual machine.

* 1. **Motivation**

The Internet of Things has the potential to transform healthcare, changing how we perform diagnosis and treatment, and, ultimately making care more accessible. However, the global nature of clinical trials remain a challenge for researchers, with only about 13 percent of trials reporting results within the federal guidelines – delaying access to potentially lifesaving treatments. This is due in part to the lack of reliable connectivity around the world, making it a struggle for patients and clinical sites to report critical data regularly and reliably.

Today, various corporations, collect electronic data for clinical research from patients to make more efficient managing worldwide clinical trials through real-time access to patient experience data. Over the next three years, iOT healthcare is going to improve society’s fitness & health overall which would lead to better life & more mortality in the clinical hospitals around the world. Many devices allow trial participants to record symptoms and experiences securely, and connects to a management a data portal where all patient trial data will be available for easy review, reporting and comparison. Not only will this give researchers a more accurate view of the whole trial, it could speed the development of safe and innovative treatments. So for the data analysis of the patients it becomes so much important to segregate the data first & analyze it for the doctor, care takers & patient. So in the end efficiently managing of health data & awareness for a better healthy lifestyle could lead to efficient way of living life in a healthy way.

* 1. **Proposed Solution**

// not edited

Keeping in mind the drawbacks of the tools that have already been developed for sandboxing, we envision a sandboxing mechanism for Linux systems which possesses the best possible combination of features. We aim at developing a tool that provides best possible performance without causing excess overheads and yet remaining user-friendly. The system we propose will use seccomp-BPF which allows the use of configurable policies for filtering system calls. This is much faster than the methods used in other tools and thus causes less overheads. Our tool provides a graphical user interface which simplifies using the system. Our toll will use process namespaces and user namespaces. Process namespaces would help isolate two processes running in the system. User namespaces on the other hand, will enable the child process to behave as the privileged user, but only within the namespace. Our tool will support a per app security policy as it will enable us to have more flexibility. We can choose restrictions for each individual app. This will allow us to be more specific and impose restrictions separately for every new app as per our needs. All the restrictions imposed on a program and the modifications made by it should be logged in a file so that it can be used for post-execution analysis. By evaluating the logged records of the processes, we can see which resources each application tried to access. From this we can judge which applications can be harmful by determining which applications tried to access resources that it need not have accessed. Keeping this in mind, we intend to include a logging mechanism in our tool. The most important advantage of our tool however, is that it will let a process have access to a virtual file system. All changes made to files can be shown to the user, and committed to the actual system only if the user wants to.

* 1. **Scope of the project**

We aim to achieve the following in the project:

* A smart band which is based on Arduino Yun Board as our hardware board which connects health related sensors to send information on the server.
* It aims for clinical patients who want to track their daily fitness information.
* The user can easily distinguish his personal analysis with comparison to the normal consensus within his age group.
* The system will track the pulse rate within every 5 minutes & will send the data to the Internet on the servers for analysis purposes. The history of the data could be analyzed on the Android Application installed on the user’s smart phone.
* If any catastrophic result has been found then the server would immediately send the request to predefined doctors for attending the patient in real life. So the doctors & the healthcare medics would have less overhead & be prepared with more agile solution.

1. **REVIEW OF LITERATURE**

The proliferation and popularity of the Internet has led to average Internet users downloading various utilities and applications from the Internet very frequently. Often, these applications are downloaded from untrusted users and websites, or from unverified third parties and suppliers. Due to this, it has become very important for a casual user to differentiate between a malicious and a benign application. This has become excessively difficult because of the rise in number of malicious applications on the Internet. In computer Security, Sandboxing is a mechanism that allows unknown or untrusted code into the system, and yet does not let it damage the system. A sandbox isolates the running program from the rest of the system by imposing restrictions on network resources and file system access, and keeps the host system safe. A sandbox system heavily restricts the program from inspecting the host or reading from the input device. In this paper, we review existing tools that provide sandboxing mechanisms. We compare what features have been used by each, and highlight the advantages and disadvantages of each. In the end, we propose a system that will incorporate the best features of these tools, yet be user-friendly.

**Tracking:**

Tracking is the function aimed at the identification of a person or object in motion. This includes both real-time position tracking, such as the case of patient-flow monitoring to improve workflow in hospitals, and tracking of motion through choke points, such as access to designated areas. In relation to assets, tracking is most frequently applied to continuous inventory location tracking (for example for maintenance, availability when needed and monitoring of use), and materials tracking to prevent left-ins during surgery, such as specimen and blood products.

**Identification and authentication:**

It includes patient identification to reduce incidents harmful to patients (such as wrong drug/dose/time/procedure), comprehensive and current electronic medical record maintenance (both in the in- and out-patient settings), and infant identification in hospitals to prevent mismatching. In relation to staff, identification and authentication is most frequently used to grant access and to improve employee morale by addressing patient safety issues. In relation to assets, identification and authentication is predominantly used to meet the requirements of security procedures, to avoid thefts or losses of important instruments and products.

**Data collection**:

Automatic data collection and transfer is mostly aimed at reducing form processing time, process automation (including data entry and collection errors), automated care and procedure auditing, and medical inventory management. This function also relates to integrating RFID technology with other health information and clinical application technologies within a facility and with potential expansions of such networks across providers and locations.

**Sensing:**

Sensor devices enable function centered on patients, and in particular on diagnosing patient conditions, providing real-time information on patient health indicators. Application domains include different telemedicine solutions, monitoring patient compliance with medication regiment prescriptions, and alerting for patient well-being. In this capacity, sensors can be applied both in in-patient and out-patient care. Heterogeneous wireless access-based remote patient monitoring systems can be deployed to reach the patient everywhere, with multiple wireless technologies integrated to support continuous bio-signal monitoring in presence of patient mobility.

**Wearable Body Area Sensors**:

Wearable body area sensors (WBAS) are frontend components of WIoT and unobtrusively envelop the body to capture health-centric data. WBAS are primarily responsible for 1) collecting the data either directly from the body through contact sensors or from peripheral sensors providing indirect information of body and its behaviors preparing the data for either on-board analysis for close-loop feedback or remote transmission for comprehensive analysis and decision support. WBAS, whether commercial or laboratory prototypes, are packaged with miniaturized sensor hardware, an embedded processor with storage capability, power management, and optional communication circuits depending on application. For example, peripheral wearable sensors such as the BodyMedia armband (Jawbone Inc., USA) are fitness monitors that work on computationally less intensive algorithms with minimum hardware requirements, and their goal is to encourage users to maintain an active lifestyle. Most of the contact-type wearable sensors contain decent electronics and computing capabilities due to the fact that they are required to provide accurate, high resolution clinical information of patients in real-time. Novel interface between sensor and body is key to successful data acquisition in wearable technologies. Ring sensor for pulse oximetry, chest-worn ECG monitor , and attachable BioPatch -all are a few examples of novel sensor locations that provide continuous access to the body’s vital signs. Smart textiles represent the forefront of wearable electronics woven into the clothes’ fabrics in order to provide unobtrusive health monitoring for patients living in their homes, away from hospitals and doctors. Smart clothes embedded with textilebased sensors were found useful in monitoring the autonomous nervous system response.

**Internet-connected Gateways:**

WBAS are rarely standalone systems due to their limited computing power and communication bandwidth. Therefore, they need to transmit data to potent computing resources that are either companion devices such as smartphones, tablets, and laptop PCs, or remotely-located cloud computing servers. In either case of data communication, companion devices are used as gateway devices, thus representing an important class of WIoT that enables the information to flow from the sensors to the cloud or server centers for storage and further analysis. The Gateway devices comprise of short-range communication

**Security & Privacy:**

As the internet evolves, it continues to pose issues of privacy and security. Earlier, desktop PCs were prone to cyber attacks, while, mobile phones and wearable devices are now under constant threat of highly-skilled, organized hackers. WIoT deal with data collected from, and provided to humans. Although collected from innocuous wearable sensors, such data is vulnerable to top privacy concerns. For example, some wearable devices collect sensitive information such as the user’s absolute location and movement activities that compromise the user’s privacy if this information is not safeguarded during the processes of storage or communication. To mitigate the risk of cyber attacks on WIoT, we need strong network security infrastructure for short- and long-range communication. In each passing layer in WIoT, from the wearable sensors to the gateway devices to the cloud, careful precautions are desired to ensure users’ privacy and security.

**Wearable IoT (WIoT):**

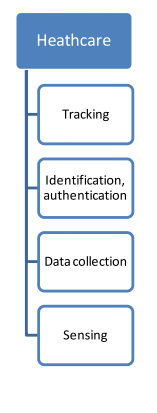
Wearable IoT (WIoT) can be defined as a technological infrastructure that interconnects wearable sensors to enable monitoring human factors including health, wellness, behaviors and other data useful in enhancing individuals’ everyday quality of life. WIoT aims at connecting body-worn sensors to the medical infrastructure such that physicians can perform longitudinal assessment of their patients when they are at home. For example, built-in motion sensors of a smartwatch are used to track disease symptoms such as tremors in Parkinson patients for tele-interventions. WIoT is still in its infancy period and therefore, demands a chain of developments in order to boost its successful evolution and to enable its widespread adoption in the healthcare industry.

**Middleware:**

The middleware is a software layer or a set of sub-layers interposed between the technological and the application levels. Its feature of hiding the details of different technologies is fundamental to exempt the programmer from issues that are not directly pertinent to her/his focus, which is the development of the specific application enabled by the IoT infrastructures. The middleware is gaining more and more importance in the last years due to its major role in simplifying the development of new services and the integration of legacy technologies into new ones. This excepts the programmer from the exact knowledge of the variegate set of technologies adopted by the lower layers. As it is happening in other contexts, the middleware architectures proposed in the last years for the IoT often follow the Service Oriented Architecture (SOA) approach. The adoption of the SOA principles allows for decomposing complex and monolithic systems into applications consisting of an ecosystem of simpler and well-defined components. The use of common interfaces and standard protocols gives a horizontal view of an enterprise system. Thus, the development of business processes enabled by the SOA is the result of the process of designing work- flows of coordinated services, which eventually are associated with objects actions. This facilitates the interaction among the parts of an enterprise and allows for reducing the time necessary to adapt itself to the changes imposed by the market evolution. A SOA approach also allows for software and hardware reusing, because it does not impose a specific technology for the service implementation. Advantages of the SOA approach are recognized in most studies on middleware solutions for IoT. While a commonly accepted layered architecture is missing, the proposed solutions face essentially the same problems of abstracting the devices functionalities and communications capabilities, providing a common set of services and an environment for service composition. It tries to encompass all the functionalities addressed in past works dealing with IoT middleware issues. It is quite similar to the scheme proposed in, which addresses the middleware issues with a complete and integrated architectural approach.

**Internet Protocol (IP):**

Internet Protocol (IP) is the primary network protocol used on the Internet, developed in 1970s. IP is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. The two versions of Internet Protocol (IP) are in use: IPv4 and IPv6. Each version defines an IP address differently. Because of its prevalence, the generic term IP address typically still refers to the addresses defined by IPv4. There are five classes of available IP ranges in IPv4: Class A, Class B, Class C, Class D and Class E, while only A, B, and C are commonly used. The actual protocol provides for 4.3 billion IPv4 addresses while the IPv6 will significantly augment the availability to 85,000 trillion addresses [22]. IPv6 is the 21st century Internet Protocol. This supports around for 2128 addresses.



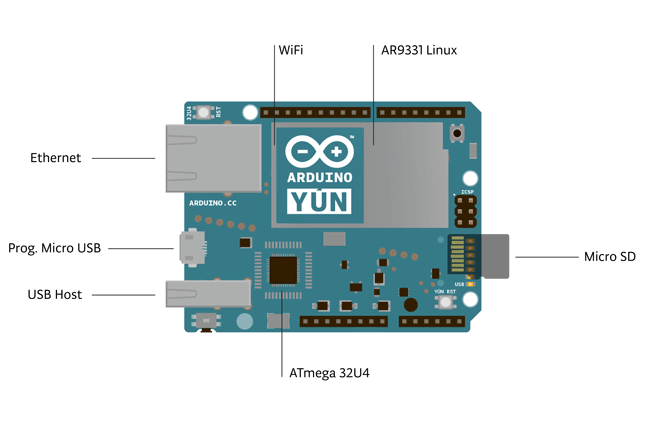
IEEE papers references

// Edited till here.

**Hardware Information:**

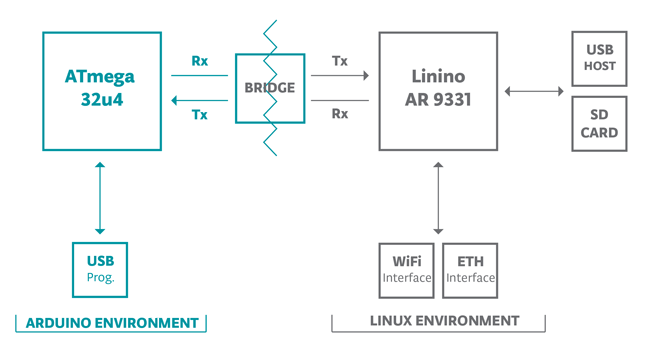
**Andruino Yun Board**

The Arduino Yún is a microcontroller board based on the [ATmega32u4](http://www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf) and the [Atheros AR9331](https://www.openhacks.com/uploadsproductos/ar9331_datasheet.pdf). The Atheros processor supports a Linux distribution based on OpenWrt named OpenWrt-Yun. The board has built-in Ethernet and WiFi support, a USB-A port, micro-SD card slot, 20 digital input/output pins (of which 7 can be used as PWM outputs and 12 as analog inputs), a 16 MHz crystal oscillator, a micro USB connection, an ICSP header, and a 3 reset buttons



The above image contains all the Architecture of the Yun board for pins & connectors locations.

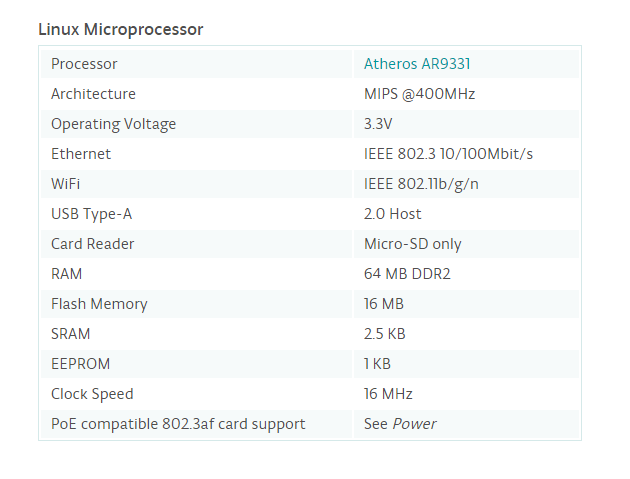
We will be using Wifi IEEE sensor for communicating with IDE of Andruino in wireless local host environment.

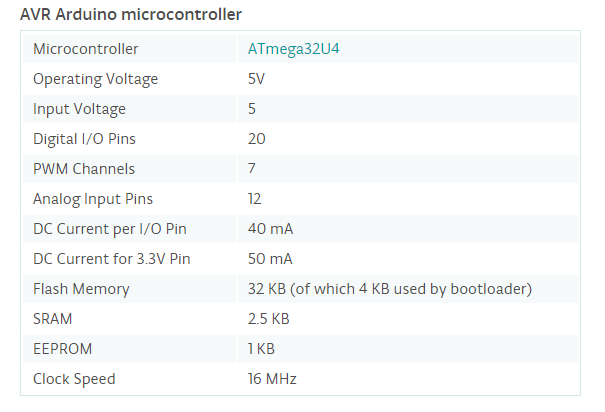


Andruino Yun board has two environments for the development purpose.

Andruino Environment is USB port interface with the IDE & the Linux environment is connected via Wifi sensor to make sure that the particular wireless hotspot is created. For flashing the program on the board using Andruino IDE we have to connect it using the wireless hotspot created by the adruino board.

The wireless password could be changed in the later stages for encryption of network data to make sure that the information is secured through the network.





**System call interposition in Linux**

System call interposition is a technique which has been used since a long time. In sandboxing, system call interposition is used because it helps the programmer to have total control over a process. Various system call interposition methods are:

* **Ptrace**: Ptrace, an abbreviation of “process trace”, is a system call using which a parent process can not only observe the behavior, but also control the execution of another process. The parent process can examine the core image and registers of another process and also change it. This mechanism is primarily used to perform system call tracing as well as breakpoint debugging.
* **Proc**: On Solaris, you can use /proc; /proc lets you specify the subset of system calls that you are interested in wrapping, which lets you achieve better performance at the cost of compatibility.
* **Ld\_preload**: LD\_PRELOAD is a dynamic linker flag which allows any other shared library to be loaded before any other library. If you set LD\_PRELOAD to the path of a shared object, that file will be loaded before any other library (including the C runtime, libc.so). So to run ls with your special malloc() implementation, do this:

$ LD\_PRELOAD=/path/to/my/malloc.so /bin/ls

* **seccomp:** secure computing mode (also called seccomp) is a facility that provides a mechanism for sandboxing in a Linux system. Version 2.6.12 of the Linux kernel which was released on 8th March, 2005, had seccomp merged into it. Seccomp puts a process into a “Secure” state wherein it allows the process to make only four system calls namely exit(), sigreturn(), read() and write() to already open file descriptors. If the process attempts to make any system calls apart from these, the kernel will terminate it using SIGKILL. Hence we can say that seccomp does not essentially virtualize the resources of the system, but isolates them from the process entirely.

**Sandbox systems:**

Misuse of system calls to compromise a system is a potential threat and to avoid it, sandboxing is used. In computer Security, Sandboxing is a mechanism that allows unknown or untrusted code into the system, and yet does not let it damage the system. A sandbox isolates the running program from the rest of the system by imposing restrictions on network resources and file system access, and keeps the host system safe. A sandbox system heavily restricts the program from inspecting the host or reading from the input device. Sandboxes can be used to analyse applications or test their legitimacy by allowing them to execute in a highly controlled environment so that they cannot cause any harm to the rest of the system. A sandbox can restrict the operations of a code by allowing it only as many permissions as required, without granting additional permissions that can be used to harm the system. Thus, sandboxing provides security by isolating a software so as to prevent malicious code from abusing the system. This can be done by specifying security policies for the applications or isolating each application in a separate virtual machine.

**Sandbox features:**

* Security Policy: Security policies provide a description of the system calls that can be made by and the other privileges given to an application. The sandbox system intercepts any system calls made by the application and compares it with the specified policies. If the application is allowed to make the system calls that it is trying to make, then it will be allowed to proceed. However, if a system call made violates the specified policy, then there can be a possibility of system intrusion. In such a case, the sandbox system denies the system calls as they might produce behavior that is unusual or harmful for the host system. These security policies play a very important role in the operation of the sandbox system and helps the host system to be protected from malware.
* System call interposition method: Tools in linux used for filtering system calls. This can be done using one of the methods from among seccomp/ptrace/proc/ld\_preload as described above.
* System calls check: sandbox system usually checks a few system calls which are more important to the goal of achieving security. A few tools interpose all system calls which results in overhead, consequently causing performance issues. A few tools interpose a few system calls causing less overhead, but is less strict, and hence may be less secure.
* File system changes: sandbox tools may either choose to allow read and write system calls which makes direct changes to the files , or it may disallow those system calls altogether. Some sandbox tools allow changes to virtual files, which may be reflected in the actual system if the user permits.
* Logging: Post-execution analysis of a targeted application might be useful sometimes to manually ensure that the sandbox worked correctly and no harmful system calls have been made by the targeted application. For this purpose, the sandbox system may offer a logging facility to users.

**Tools:**

Sandboxing using system call interposition has been popular for a long time now and thus a number of security tools based on system call interposition have been developed. In the following section we will review some of these tools by sharing their features.

* Janus: Janus is a sandboxing tool which mainly focuses on system call interposition. When a process makes a system call , janus puts it to sleep and confirms with the security policy of the process whether it can be allowed to proceed. After checking the security policy, janus can either allow or deny the system call. In case system call is denied, then the tool returns error message to traced process. This tool checks only those system calls that grant or manipulate file descriptors. It does not check system calls such as write and read as they operate on already open file descriptors. As a result , the overhead caused due to switching between the kernel and user space is reduced. Janus is implemented using proc interface in solaris os. It allows user to specify only one global security policy and the same configuration file is used for all processes. The main drawback of janus is that it does not allow multi threaded programs as it cannot handle race conditions. However, the newer version supports multi threading. Apart from multi threading, janus also does not have logging mechanism. So if the user wants to analyse the behaviour of a program after its execution, then the user cannot use janus.
* BlueBox: Bluebox uses ptrace interface to intercept system calls and impose the specified security rule. It generates a list of harmless system calls and skips these system calls at the time of interposition. It supports static security policies that need to be defined previously. It does not allow user to specify the policies dynamically. It supports interposition of system calls for filesystem resources. However apart from the file system related calls there are various system calls that affect the system like network.
* MapBox: MapBox is also based on system call interposition but its operation in different from the tools that we have seen previously. Like Janus, MapBox too uses the proc interface of Solaris OS for tracing and intercepting system calls. However, it uses a slightly different method for specifying security policies. Mapbox creates classes based on the behaviour and the functionality of program. It specifies a separate security policy for each class. When the program is running in the sandbox, it is assigned one or more behaviour classes and based on the policy file assigned to these classes, the program is permitted to access the resources. The major drawback of the MapBox is that multiple programs belonging to the same behaviour class will be given same access rights which is not reliable. When the security policy is violated MapBox allows the program to make the system call but ensures that the changes are made to a copy of the original file. Thus the host system is not affected and the running program is unaware of the where the changes are being made.
* Systrace: It not only has a detailed logging mechanism but also supports generation of different security policies for different applications. It allows the user to specify the policies dynamically via an interface. The system calls in this tool are partly checked in the kernel and partly in use space. If the system call is known to be harmful or harmless for sure, then it is resolved at the kernel level itself, else it is passed to the user space. Here, the user can dynamically specify whether to allow or deny a system call if it is not already mentioned in the policy. Drawbacks of this tool are as follows :

1. It cannot work with the latest kernel.

2. It requires a training session.

3. It has some usability issues.

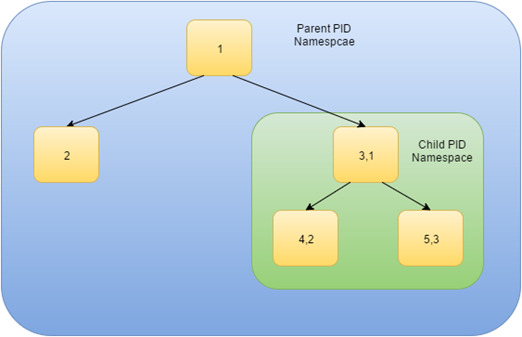
* Ostia: This tool was built in order to eliminate the drawbacks of Janus. Ostia has a delegating architecture wherein only a small number of system calls that might be dangerous, are delegated. If the system call made is one that needs to be delegated, then the system call is converted to a message and sent to the tracer which acts as an agent between the system and targeted application. The tracer process then checks the policy to see if the call is supported by it. If yes, then the tracer makes the system call and returns the outcome to the running process. The use of the agent maintains atomicity in the system and thus avoids race conditions. However, as a separate agent is created for each thread in a program, Ostia too, does not support multi-threading. Ostia does not have support for interactive security policy generation.
* MBox: MBox introduces a file system layer between the host file system layer and the operating system. The process running in the sandbox affects only the virtual file system layer and does not make any changes to the actual file system. Once the program has finished executing, the user can examine the virtual layer to see what changes have been made and decide which of these changes must be reflected on the host file system. The insertion of the virtual file system layer is done by allocating the sandbox a separate file on the host system. Any changes made by the running program are recorded here. This is possible because the sandbox rewrites the arguments of the system call to direct the effects of the system call to the required folder. In this tool when a sensitive program is called , it will point to a new page called read only memory. This memory is private and can not be modified.
* SElinux : This tool is found in all modern tools like Ubuntu, red hat enterprise. All the new linux systems are configured with the default security policies of SElinux. The user can enable or disable the tool . The drawback of this system is that it does not allow the user to trace application activity and logging mechanism for post-execution analysis.
* AppArmor: This is similar to SElinux. This tool focus on the file paths and also helps the user to create ACLs to protect file and network resources. This tool gives an additional feature wherein the defilements of the program are recorded. These records helps to create new security policy. The drawback of Apparmor is similar to Selinux that it does not provide a proper graphical interface. Moreover, it is difficult to define security policies.

# New Technologies:

Several new features have been introduced in the linux Kernel recently. These features make it easier to confine running processes. Features which are of interest to our system are:

* **Seccomp-bpf**: An extension to seccomp, seccomp-bpf is implemented using Berkeley Packet Filter rules. It allows the use of configurable policies for filtering system calls. These filters can allow or deny specific system calls. Seccomp-bpf is available since Linux version 3.5. Google Chrome/Chromium web browsers on Chrome OS and Linux as well as OpenSSH and vsftpd use seccomp-bpf.
* **Linux Namespaces**: Namespaces are an elegant way to confine processes. “**chroot**” in Linux is a way to change the apparent root directory for a process and its children. This way, the process can only access certain parts of the file system. Similarly, namespaces allow other aspects of the Operating System, like process tree, networking interfaces, mount points, inter-process communication resources and more, to be modified as well.

o **Process namespaces:** Linux kernel maintains a single process tree. Given that a process has the appropriate permissions, it can kill any other process. Process namespaces allow a process to view a completely isolated process tree so that it cannot kill the processes not created by itself even if it has the required permissions.



o **Network Namespaces:** A network namespace allows processes to view a different set of networking interfaces. Even a different loopback interface.

o **Mount Namespaces**: It allows processes to see a complete independent set of mount points. If the child process tries to change mount points, or the root directory, it will only be seen by the child process and will not be affected in the original mount point structure.

o **User Namespaces**: user namespaces allow a process to have root access within the namespace, without having access to processes not belonging in that namespace.

We propose a system that incorporates the following features:

* Seccomp-BPF: It is much faster than ptrace, causing less overheads. This enables us to run processes without causing significant overheads. The downside is that it is available on the recent kernels.
* GUI: A casual user would prefer a graphical user interface which simplifies using the system. The user should feel at ease when defining policies, and enforcing restrictions.
* Namespaces: For most basic purposes, it is essential to incorporate process namespaces and user namespaces. Process namespaces would help isolate two processes running in the system. User namespaces on the other hand, enables the child process to behave as the privileged user, but only within the namespace.
* Per app security policy: This enables us to have more flexibility. We can choose restrictions for each individual apps. This will allow us to be more specific and impose restrictions separately for every new app as per our needs. This is much better than having a global security policy or different static policies for different classes of applications.
* Logging: All the restrictions imposed on a program and the modifications made by it should be logged in a file so that it can be used for post-execution analysis. By evaluating the logged records of the processes, we can see which resources each application tried to access. From this we can judge which applications can be harmful by determining which applications tried to access resources that it need not have accessed.
* Virtual File system: Much like what Mbox implements, it is beneficial to let a process have access to a virtual file system. All changes made to files can be shown to the user, and committed to the actual system only if the user wants to. This can also help us install software programs virtually and test them before installing to the actual system.

Review of the already existing sandboxing mechanisms showed us that there were a lot of drawbacks in each tool and that there was no ideal mechanism available to be used on a local Linux based system. In the system that we propose, we try to eliminate those drawbacks by maintaining optimal performance. Our tool not only supports unique security policies, a detailed logging mechanism and an interactive and user-friendly GUI, but also introduces a virtual file system layer. It allows the user to examine all the modifications caused by a running program and gives the user the choice to decide which of these changes must be committed. It reduces overheads without compromising on performance and also supports process isolation.

1. **SYSTEM ANALYSIS**
   1. **Functional Requirements**

REQ 1:- Security Policy

The system must support specification of static security policies and also dynamically ask the user for input when a specific rule is missing in the policy. Security policies provide a description of the system calls that can be made by and the other privileges given to an application. The sandbox system intercepts any system calls made by the application and compares it with the specified policies. If the application is allowed to make the system calls that it is trying to make, then it will be allowed to proceed.

REQ 2:- System call interposition

The tool must implement a mechanism to interpose system calls at the time of execution of a program. Tools in linux used for filtering system calls. This can be done using one of the methods from among seccomp/ptrace/proc/ld\_preload.

REQ 3:- System calls check

The system must only check sensitive system calls and skip harmless system calls so as to reduce overhead. For this purpose, there must be a way of differentiating between malicious and benign system calls.

REQ 4:- File System changes

Sandbox tools may either choose to allow read and write system calls which makes direct changes to the files , or it may disallow those system calls altogether. Our sandbox tools must allow changes to virtual files, which may be reflected in the actual system if the user permits.

REQ 5:- Logging

Post-execution analysis of a targeted application might be useful sometimes to manually ensure that the sandbox worked correctly and no harmful system calls have been made by the targeted application. For this purpose, our sandbox system may offer a logging facility to users.

* 1. **Non-Functional Requirements**

The non-functional requirements of the application are as shown below:

**3.2.1 Reliability**

The sandbox should work as per the specified security policy by implementing all the rules appropriately.

**3.2.2 User-friendliness**

The application is designed for easy usage by any person. The primary objective of this application is enabling an individual to specify security policy rules without much technical expertise. The user however, should be literate in the English language in order to interact with the system effectively.

**3.2.3 Performance**

The system must interpose only the necessary system calls so as to reduce overhead. It reduces overheads without compromising on performance and also supports process isolation. In the system that we propose, we try to eliminate those drawbacks by maintaining optimal performance.

**3.2.4 Flexibility**

The system makes changes to virtual files and hence provides the user flexibility to choose which changes to commit. Logging mechanism can be used for post execution analysis and the observations can be used to make the system more secure. If a certain rule is not specified in the policy, the user is allowed to dynamically specify it.

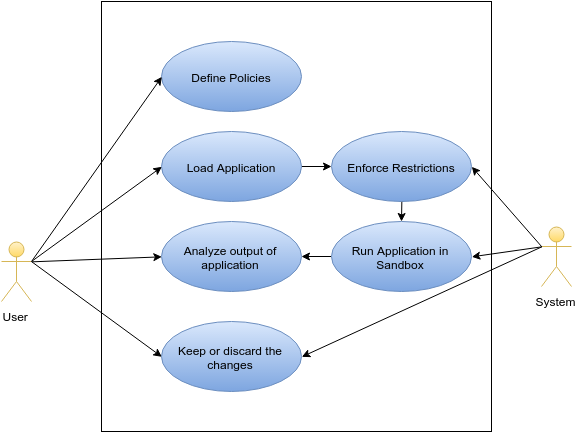
* 1. **Specific Requirements**

**3.3.1 Software Requirements**

* Linux with kernel version post 3.5
* Gcc, g++
* Library libseccomp
* WXwidgets
* Python
* WXpython
* WXFormBuilder

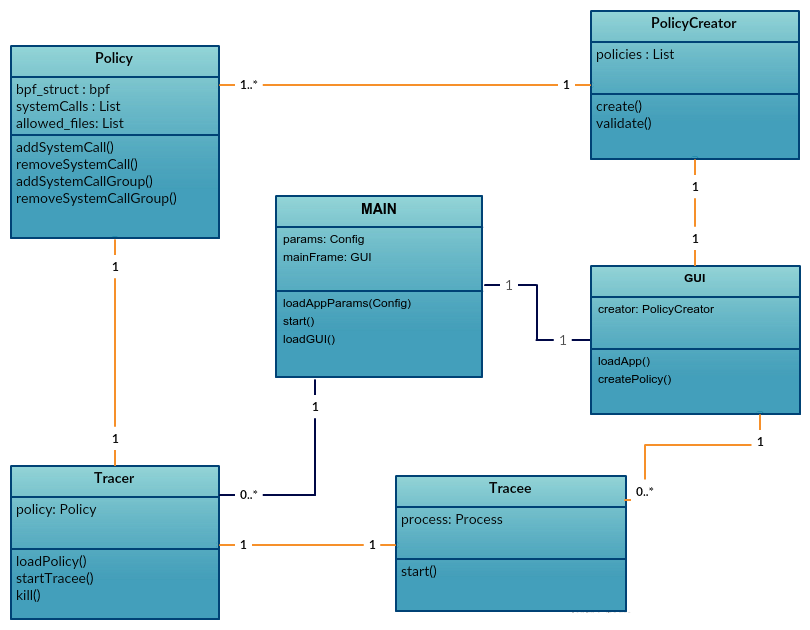
**3.3.2 Hardware Requirements**

* Intel processor or AMD processor with minimum 1 GHz clock speed.
* Sufficient amount of RAM
* Monitor
  1. Use-Case Diagrams and description



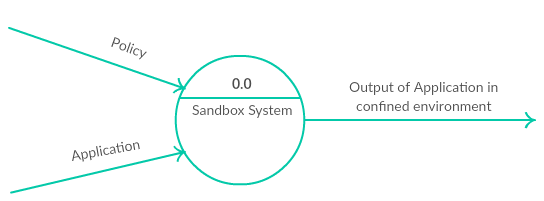
A user can define security policies for the programs to be executed in a sandbox. The user can also load the desired application for execution. Once an application is loaded for execution, the system enforces the necessary restrictions as per the corresponding security policy. The application is then run in the sandbox with the restrictions imposed. Once the application has completed its execution, the user can analyze the output of the application i.e user can inspect the changes made to the virtual file system. On the basis of this, the user can now decide which changes he wants to retain and which ones he wants to discard them.

1. **ANALYSIS MODELING**
   1. **Class Diagrams**



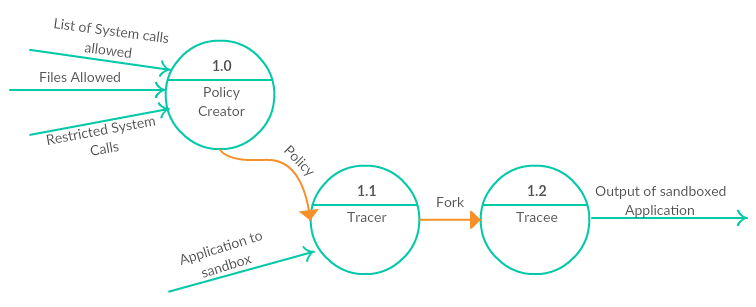
We have a main class which is responsible for starting the GUI and loading the initial application parameters for configuration. The main class can load multiple tracers which in turn loads the security policies. The tracer class is responsible for starting the tracee by using the ‘fork’ system call. Each tracer forks a single tracee. The GUI class creates a PolicyCreator object which simplifies the creation of the policy file to be used by the tracer. A single GUI class creates a single PolicyCreator object. However, a single PolicyCreator object can be used to create multiple policies. The policy class stores various parameters like the list of system calls allowed and restricted, and also a list of allowed files.

* 1. **Functional Modeling**



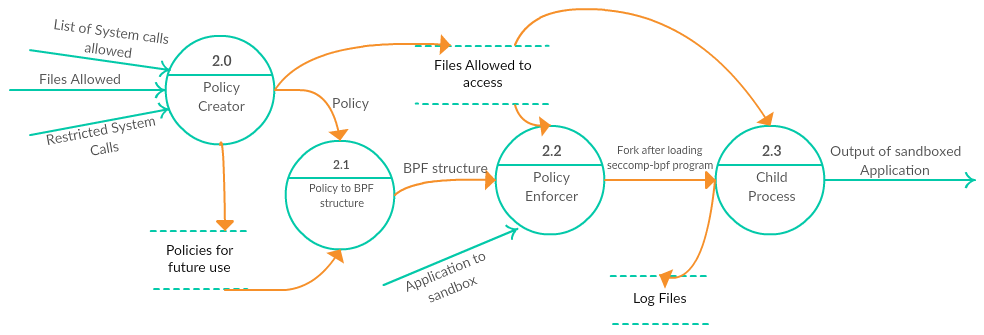
Level 0 DFD

Our sandbox system takes two inputs- an application that needs to be run in a sandbox and a security policy that needs to be enforced on the corresponding application. The result of execution of the application in our sandbox will give the output of the application as it would be in the confined environment.



Level 1 DFD

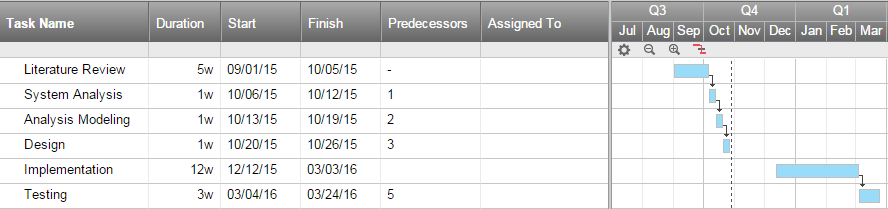
Our sandbox system can be thought of as containing a policy creator, a tracer and a tracee. The policy creator takes all the rules of the policy as input. This includes the list of system calls and files that are to be allowed and a list of system calls that need to be restricted. The tracer process takes the policy and the application to be sandboxed as input and in turn forks a tracee, i.e. the sandboxed process. Execution of the tracee gives the output of the sandboxed application.



Level 2 DFD

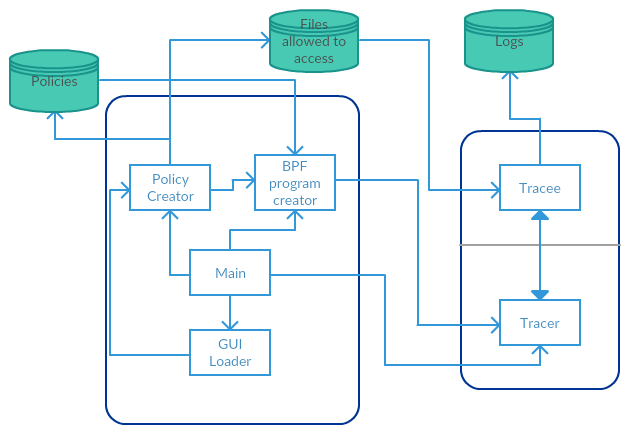
The policy creator creates a policy which is stored for future use. The files allowed to be accessed as per the policy are also stored. The created policy is then converted to a BPF structure which is further passed on to the policy enforcer. The policy enforcer also refers to the files that can be accessed in order to enforce the policy appropriately. After loading the seccomp-bpf program, the policy enforcer forks a child process that performs the actual execution of the application in a sandbox. At this stage, the log files are also updated.

* 1. **TimeLine Chart**



The above figure shows the Gantt Chart of the project It represents the tasks involved in the project along with the approximate time duration required to complete the task, Along with this, an expected start and finish date is associated with each task. Predecessors are used to represent any dependencies present among the tasks. The project is expected to be completed by the end of March 2016.

1. **DESIGN**
   1. **Architectural Design**



The system is divided into two main subsystems. One subsystem consists of the tracer and the tracee while the other consists of the main class, GUI creator, policy creator and the BPF program creator. The system also consists of policies, a list of files allowed to access and logs. The main component starts the GUI which in turn loads the policy creator. The policy creator refers to the policies to pass the needed policy characteristics to the BPF program creator. This component in turn creates the desired BPF program for the policy and passes this on to the tracer. The tracer forks a tracee and enforces the security policy when the tracee is executing. The log files are also updated as the tracee executes.

// Conclusion & references editing done. No future scope done yet.

1. **CONCLUSIONS & FUTURE SCOPE**

The Internet has changed drastically the way we live, moving interactions between people at a virtual level in several contexts spanning from the professional life to social relationships. The IoT has the potential to add a new dimension to this process by enabling communications with and among smart objects, thus leading to the vision of ‘‘anytime, anywhere, anymedia, anything” communications. To this purpose, we observe that the IoT should be considered as part of the overall Internet of the future, which is likely to be dramatically different from the Internet we use today. In fact, it is clear that the current Internet paradigm, which supports and has been built around host-tohost communications, is now a limiting factor for the current use of the Internet. It has become clear that Internet is mostly used for the publishing and retrieving of information (regardless of the host where such information is published or retrieved from) and therefore, information should be the focus of communication and networking solutions. This leads to the concept of data-centric networks, which has been investigated only recently. According to such a concept, data and the related queries are self-addressable and self-routable. In this perspective, the current trend, which we are assigning an IPv6 address to each IoT element so as to make it possible to reach them from any other node of the network, looks more suitable for the traditional Internet paradigm. Therefore, it is possible that the Internet evolution will require a change in the above trend. Another interesting paradigm which is emerging in the Internet of the Future context is the so called Web Squared, which is an evolution of the Web 2.0. It is aimed at integrating web and sensing technologies together so as to enrich the content provided to users. This is obtained by taking into account the information about the user context collected by the sensors (microphone, cameras, GPS, etc.) deployed in the user terminals. In this perspective, observe that Web Squared can be considered as one of the applications running over the IoT, like the Web is today an (important) application running over the Internet. In this paper, we have surveyed the most important aspects of the IoT with emphasis on what is being done and what are the issues that require further research. Indeed, current technologies make the IoT concept feasible but do not fit well with the scalability and efficiency requirements they will face. We believe that, given the interest shown by industries in the IoT applications, in the next years addressing such issues will be a powerful driving factor for networking and communication research in both industrial and academic laboratories.

1. **REFERENCES / BIBLIOGRAPHY**

[1] D. Giusto, A. Iera, G. Morabito, L. Atzori (Eds.), The Internet of Things, Springer, 2010. ISBN: 978-1-4419-1673-0.

[2] National Intelligence Council, Disruptive Civil Technologies – Six Technologies with Potential Impacts on US Interests Out to 2025 – Conference Report CR 2008-07, April 2008, .

[3] Cook, D. and Youngblood, G. 2008. Algorithms for smart spaces. Handbook of Smart. (2008), 783-801.

[4] Ricquebourg, V. et al. 2006. The Smart Home Concept : our immediate future. 2006 1ST IEEE International Conference on E-Learning in Industrial Electronics. (Dec. 2006), 23-28.

[5] Deleawe, S. et al. 2010. Predicting Air Quality in Smart Environments. Journal of ambient intelligence and smart environments. 2, 2 (Jan. 2010), 145-152. [6] Van Laerhoven, K. and Gellersen, H.-W. 2002. Multi sensor context awareness. Department of Computing, Lancaster University, Lancaster, UK

[6] K. Sakamura, Challenges in the age of ubiquitous computing: a case study of T-engine – an open development platform for embedded systems, in: Proceedings of ICSE’06, Shanghai, China, May 2006.

[7] M. Presser, A. Gluhak, The Internet of Things: Connecting the Real World with the Digital World, EURESCOM mess@ge – The Magazine for Telecom Insiders, vol. 2, 2009.

[8] M. Botterman, for the European Commission Information Society and Media Directorate General, Networked Enterprise & RFID Unit – D4, Internet of Things: An Early Reality of the Future Internet, Report of the Internet of Things Workshop, Prague, Czech Republic, May 2009. [9] B. Sterling, Shaping Things – Mediawork Pamphlets, The MIT Press, 2005.