Data Integrity Protection in Cloud Computing
Shashidhara, R1 Merin Thomas2
1PG Student 2Assistant Professor
1,2Department of Computer Science and Engineering
1,2DBIT, Bangalore

Abstract—In this paper, we aim to provide protection and security solutions based on many aspects of large integrated system in cloud. Cloud computing is the emerging technology to minimize the lots of users burden by providing different types of service to users. Here we also implement our system using ns2 which also show how our application can work in real world scenario. First we list some of the sources where we can download ns3. Secondly we discuss how to build our application and implementing in a real world database. It is high-speed data recovery scheme with minimal loss probability and using a forward error correction scheme to handle bursty loss. The proposed approach is highly efficient in recovering the singleton losses almost immediately and from bursty data losses.

Key words: Cloud Computing, Security Solutions, Data Integrity

I. INTRODUCTION
There are several different definitions of cloud computing, but all of them agree on how to provide services to users of the network. Cloud computing is an Internet-based development and use of computer technology.

Data integrity is defined as the accuracy and consistency of stored data, in absence of any alteration to the data between two updates of a file or record. Cloud services should ensure data integrity and provide trust to user privacy. Although outsourcing data into the cloud is economically attractive for the cost and complexity of long-term large-scale data storage, it’s lacking of offering strong assurance of data integrity and availability may impede its wide adoption by both enterprise and individual cloud users. Ns are an object oriented simulator, written in C++, with an OTcl interpreter as a frontend. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy in this document), and a similar class hierarchy within the OTcl interpreter (also called the interpreted hierarchy in this document). The two hierarchies are closely related to each other; from the user’s perspective, there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy. The root of this hierarchy is the class Tcl Object. Users create new simulator objects through the interpreter; these objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class hierarchy is automatically established through methods defined in the class Tcl Class. User instantiated objects are mirrored through methods defined in the class Tcl Object. There are other hierarchies in the C++ code and OTcl scripts; these other hierarchies are not mirrored in the manner of Tcl Object.

Class TclObject is the base class for most of the other classes in the interpreted and compiled hierarchies. Every object in the class TclObject is created by the user from within the interpreter. An equivalent shadow object is created in the compiled hierarchy. The two objects are closely associated with each other. The class TclClass, described in the next section, contains the mechanisms that perform this shadowing. In the rest of this document, we often refer to an object as a TclObject or by this, we refer to a particular object that is either in the class TclObject, or in a class that is derived from the class TclObject. If it is necessary, we will explicitly qualify whether that object is an object within the interpreter, or an object within the compiled code. In such cases, we will use the abbreviations “interpreted object”, and “compiled object” to distinguish the two. and within the compiled code respectively.

Unlike ns v1, the class TclObject subsumes the earlier functions of the NsObject class. It therefore stores the interface variable bindings (Section 3.4.2) that tie OTcl instance variables in the interpreted object to corresponding C++ member variables in the compiled object. The binding is stronger than in ns v1 in that any changes to the OTcl variables are trapped, and the current C++ and OTcl values are made consistent after each access through the interpreter. The consistency is done through the class InstVar (Section 3.8). Also unlike ns v1, objects in the class Tcl Object are no longer stored as a global link list. Instead, they are stored in a hash table in the class Tcl.

By convention in ns, the class Agent/SRM/Adaptive is a subclass of Agent/SRM, is a subclass of Agent, and is a subclass of TclObject. The corresponding compiled class hierarchy is the ASRMAgent, derived from SRMAgent, derived from Agent, derived from TclObject respectively. The first line of the above example shows how a TclObject is created (or destroyed) (Section 3.4.1); the next line configures a bound variable (Section 3.4.2); and finally, the last line illustrates the interpreted object invoking a C++ method as if they were an instance procedure malicious. In most cases, access to compiled member variables is restricted to compiled code, and access to interpreted member variables is likewise confined to access via interpreted code; however, it is possible to establish bi-directional bindings such that both the interpreted member variable and the compiled member variable access the same data, and changing the value of either variable changes the value of the corresponding paired variable to same value. The binding is established by the compiled constructor when that object is instantiated; it is automatically accessible by the interpreted object as an instance variable. Ns support five different data types: real, bandwidth valued variables, time valued variables, integers, and Booleans. The syntax of how these values can be specified in OTcl is different for each variable type.

Note that all of the functions above take two arguments, the name of an OTcl variable, and the address of the corresponding compiled member variable that is linked. While it is often the case that these bindings are established by the constructor of the object, it need not always be done in this manner. We will discuss such alternate methods when we describe the class InstVar in detail later. Each of
the variables that are bound is automatically initialized with default values when the object is created. The default values are specified as interpreted class variables. This initialization is done by the routing init-instvar {}, invoked by methods in the class Instvar, described later. Init-instvar {} checks the class of the interpreted object, and all of the parent class of that object, to find the first class in which the variable is defined. It uses the value of the variable in that class to initialize the object.

II. SYSTEM MODEL

Cloud Computing has been considered as the next generation architecture of IT Enterprise. Cloud computing moves the application software and databases to the centralized large data centers, where the management of the data and services may not be fully trustworthy. This new paradigm makes many new security challenges depicts Cloud Data Storage Model (10). A cloud computing is made up of several elements: Cloud users, cloud service providers, third party auditors (11). Each element plays a specific role in delivering functional cloud-based application. Although cloud computing certainly gives organizations with significant cost savings and operational efficiencies, it also brings new security risks and uncertainties. The increased attack surface in a Cloud environment allows for other vulnerabilities to be exploited, thereby increasing the organization’s risk (An Introduction to Securing a Cloud Environment). The risk is defined as a given threat that exploits vulnerabilities of an asset or group of assets and thereby cause harm to the organization (11). The increased attacks in cloud environment; virtual switches and hypervisor that are not present in the traditional data center, allows for other vulnerabilities to be exploited, thereby increasing the organization’s risk (9).

Virtualized servers will be less secure than the physical servers. However, all risks are not reduced by moving operations to a cloud environment. While some risks are reduced, other risks may increase. With the addition of virtual network switches, hypervisors and virtual images, the attack surface increases. A single host with multiple virtual machines may be attacked by one of the guest operating systems, or a guest operating system may be used to attack other guest operating systems.

Therefore, every new Agent/SRM/Adaptive object will have distance_ set to 12.0; last Sent_ is set to 8.345m from the setting of the class variable of the parent class; ctrlLimit_ is set to 1.44M using the class variable of the parent class twice removed; running is set to false; the instance variable distance_ is not initialized, because no class variable exists in any of the class hierarchy of the interpreted object. In such instance, init-instvar {} will invoke warn-instvar {}, to print out a warning about such a variable. The user can selectively override this procedure in their simulation scripts, to elide this warning. Note that the actual binding is done by instantiating objects in the class InstVar. Each object in the class InstVar binds one compiled member variable to one interpreted member variable. A TclObject stores a list of InstVar objects corresponding to each of its member variable that is bound in this fashion. The head of this list is stored in its member variable instvar_ of the TclObject. One last point to consider is that ns will guarantee that the actual values of the variable, both in the interpreted object and the compiled object, will be identical at all times. However, if there is methods and other variables of the compiled object that track the value of this variable, they must be explicitly invoked or changed whenever the value of this variable is changed. This usually requires additional primitives that the user should invoke. One way of providing such primitives in ns is through the command () method described in the next section.

III. CLOUD COMPUTING SECURITY ISSUE

Cloud Computing has been considered as the next generation architecture of IT Enterprise. Cloud computing moves the application software and databases to the centralized large data centers, where the management of the data and services may not be fully trustworthy. This new paradigm makes many new security challenges. Figure (1) depicts Cloud Data Storage Model (10). A cloud computing is made up of several elements; Cloud users, cloud service providers, third party auditors (11). Each element plays a specific role in delivering functional cloud-based application. SaaS is becoming an increasingly prevalent delivery model as underlying technologies that support Web services and service-oriented architecture (SOA) mature and new developmental approaches, such as Ajax, become popular. Meanwhile, broadband service has become increasingly available to support user access from more areas around the world.

SaaS is closely related to the ASP(application service provider) and on demand computing software delivery models. IDC identifies two slightly different delivery models for SaaS. The hosted application management (hosted AM) model is similar to ASP: a provider hosts commercially available software for customers and delivers it over the Web. In the software on demand model, the provider gives customers network-based access to a single copy of an application created specifically for SaaS distribution.

Platform as a Service allows users to create software applications using tools supplied by the provider. PaaS services can consist of preconfigured features that customers can subscribe to; they can choose to include the features that meet their requirements while discarding those that do not. Consequently, packages can vary from offering simple point-and-click frameworks where no client side hosting expertise is required to supplying the infrastructure options for advanced development.

The infrastructure and applications are managed for customers and support is available. Services are constantly updated, with existing features upgraded and additional features added. PaaS providers can assist developers from the conception of their original ideas to the creation of applications, and through to testing and deployment. This is all achieved in a managed mechanism.
IaaS is one of the three fundamental service models of cloud computing alongside Platform as a Service PaaS and Software as a Service SaaS. As with all cloud computing services it provides access to computing resource in a virtualised environment, “the Cloud”, across a public connection, usually the internet. In the case of IaaS the computing resource provided is specifically that of virtualised hardware, in other words, computing infrastructure. The definition includes such offerings as virtual server space, network connections, bandwidth, IP addresses and load balancers. Physically, the pool of hardware resource is pulled from a multitude of servers and networks usually distributed across numerous data centers, all of which the cloud provider is responsible for maintaining. The client, on the other hand, is given access to the virtualised components in order to build their own IT platforms.

In common with the other two forms of cloud hosting, IaaS can be utilised by enterprise customers to create cost effective and easily scalable IT solutions where the complexities and expenses of managing the underlying hardware are outsourced to the cloud provider. If the scale of a business customer’s operations fluctuate, or they are looking to expand, they can tap into the cloud resource as and when their needs it rather than purchase, install and integrate hardware themselves.

**IV. SECURITY RISKS**

Virtualized servers will be less secure than the physical servers. However, all risks are not reduced by moving operations to a cloud environment. While some risks are reduced, other risks may increase. With the addition of virtual network switches, hypervisors and virtual images, the attack surface increases. A single host with multiple virtual machines may be attacked by one of the guest operating systems, or a guest operating system may be used to attack other guest operating systems. The TclClass library exports two classes of TracedVar: TracedInt and TracedDouble. These classes can be used in place of the basic type int and double respectively. Both TracedInt and TracedDouble overload all the operators that can change the value of the variable such as assignment, increment, and decrement. These overloaded operators use the assign method to assign the new value to the variable and call the tracer if the new value is different from the old one. TracedInt and TracedDouble also implement their value methods that output the value of the variable into string. The width and precision of the output can be pre-specified.

This section describes the internals of the class InstVar. This class defines the methods and mechanisms to bind a C++ member variable in the compiled shadow object to a specified OTcl instance variable in the equivalent interpreted object. The binding is set up such that the value of the variable can be set or accessed either from within the interpreter, or from within the compiled code at all times. There are five instance variable classes: class InstVarReal, class InstVarTime, class InstVarBandwidth, class InstVarInt, and class InstVarBool, corresponding to bindings for real, time, bandwidth, integer, and boolean valued variables respectively. We now describe the mechanism by which instance variables are set up. We use the class InstVarReal to illustrate the concept. However, this mechanism is applicable to all five types of instance variables. When setting up an interpreted variable to access a member variable, the member functions of the class InstVar assume that they are executing in the appropriate method execution context; therefore, they do not query the interpreter to determine the context in which this variable must exist. In order to guarantee the correct method execution context, a variable must only be bound if its class is already established within the interpreter, and the interpreter is currently operating on an object in that class. Note that the former requires that when a method in a given class is going to make its variables accessible via the interpreter, there must be an associated. An implicit solution occurs whenever a new TclObject is created within the interpreter. This sets up the method execution context within the interpreter. When the compiled shadow object of the interpreted TclObject is created, the constructor for that compiled object can bind its member variables of that object to interpreted instance variables in the context of the newly created interpreted object. An explicit solution is to define a bind-variables operation within a command function, that can then be invoked via the cmd method. The correct method execution context is established in order to execute the cmd method. Likewise, the compiled code is now operating on the appropriate shadow object, and can therefore safely bind the required member variables. An instance variable is created by specifying the name of the interpreted variable, and the address of the member variable in the compiled object. The constructor for the base class InstVar creates an instance of the variable in the interpreter, and then sets up a trap routine to catch all accesses to the variable through the interpreter. Whenever the variable is read through the interpreter, the trap routine is invoked just prior to the occurrence of the read. The routine invokes the appropriate get function that returns the current value of the variable. This value is then used to set the value of the interpreted variable that is then read by the interpreter. Likewise, whenever the variable is set through the interpreter, the trap routine is invoked just after to the write is completed. The routine gets the current value set by the interpreter, and invokes the appropriate set function that sets the value of the compiled member to the current value set within the interpreter.

This object is used to classify a packet as a member of a particular flow. As their name indicates, hash classifiers use a hash table internally to assign packets to flows. These objects are used where flow-level information is required (e.g. in flow-specific queuing disciplines and statistics collection). Several “flow granularities” are available. In particular, packets may be assigned to flows based on flow ID, destination address, source/destination addresses, or the combination of source/destination addresses plus flow ID.

The classical method to handle this case is through class inheritance. For instance, if one wants a node that supports hierarchical routing, one simply derive a Node/Hier from the base node and override the classifier setup methods to insert hierarchical classifiers. This method works well when the new function blocks are independent and cannot be “arbitrarily” mixed. For instance, both hierarchical routing and ad hoc routing use their own set of classifiers. Inheritance would require that we have Node/Hier that
supports the former, and Node/Mobile for the latter. This becomes slightly problematic when one wants an ad hoc routing node that supports hierarchical routing. In this simple case one may use multiple inheritance to solve the problem, but this quickly becomes infeasible as the number of such function blocks increases.

V. CONCLUSION
Cloud computing has been considered as an emerging technology in future IT world in which we can store lots of data and we can also minimize user burden. In this paper we have come across many concepts of cloud for example SaaS, PaaS and IaaS. In this paper we list some of the security issue and its risk in cloud. Also we discuss about providing protection for data integrity for cloud.

ACKNOWLEDGEMENT
I would like to express my gratitude towards my mentor Mrs. Merin Thomas, Assistant Professor in Department Of Computer Science and Engineering at DBIT College, Bangalore for her kind co-operation and encouragement which helped me to understand the vast topic of Data Integrity Protection in Cloud Computing. I would like to thank my family as well for their continuous support and encouragement.

REFERENCE