Cloud systems, as individual components of larger aggregated systems, need to be continuously monitored for security and reliability, and if determined to be out of compliance for any reason, those elements need to be triaged. To do this, component system responses must be evaluated against some known standard to evaluate deviation. A simple example is evaluating a request/response cycle for elapsed response times, where if that response time exceeds a specific threshold, an alarm is triggered, or the slow system is replaced by a failover element.

This consists of two distinct activities --- an *evaluation* stage, followed by a *mitigation* stage. Furthermore, either stage can be on or offline; online evaluation occurs in soft or hard real-time, while offline evaluation occurs without any real-time restrictions. Returning to our response time example, online evaluation and decision would check the response time to a request following the request's response and immediately activate failover systems if the response time is greater than a threshold. Offline evaluation could occur through a logging system, where a sensor tracks information with respect to response time and logs that information for later review; that later review would evaluate that response time and act accordingly.

Furthermore, evaluation activities can either be standalone, whereby the cloud system as a whole makes decisions with respect to the dependability of single elements based on individual measurements, or aggregated, in which some kind of trust authority makes decisions with respect to system inclusion based on more than one measured response and potentially including outside context.

Problem

In a nutshell, the problem with these kinds of distributed cloud systems is a wholesale lack of control. Systems that used to be guaranteed to be at least physically secure (within realistic bounds) can no longer be believed to be secure at all. Furthermore, virtually, these systems can be spoofed, compromised, or otherwise rendered useless at best and dangerous at worst. As these systems are more frequently used in DoD computing infrastructures, system managers need some way to automatically monitor trustworthiness and control failover and virtual replacement.

Utility computing, and cloud computing as its realization, are not passing trends. They are here to stay, and computation-intensive agencies and organizations need to evolve to be able to better mange new, emerging risks.

Objective

In order to mitigate these new risks, organizations must be able to evaluate system trustworthiness and mitigate any exposed problems. In order to reach this goal, agencies and companies need to be able to monitor and control arbitrarily complex infrastructures under time constraints that preclude all but the most high-level manual intervention.

Developed systems must handle various types of cloud services, ranging from common software-as-a-service models (SaaS) to both infrastructure and platform-as-a-service (IaaS, PaaS) styles of computation. Future aggregated computing systems can count on needing to integrate best-of-breed commercial software service offerings as well as requiring platform and infrastructure services like queuing or managed virtual machine instances, respectively. As these systems are integrated, they must also be managed and controlled in order to prevent either system compromise or recover from malicious intrusion scenarios.

The objectives of this initial work include incorporating current and developing new approaches to evaluating trust in information delivered from services, designing the architecture of a prototypical

system to evaluate trust using the proposed methodologies, and finally developing a proof-of-concept system demonstrating the new capabilities.

Approach

We have two central cases we must address – an evaluation case, where we determine the trust associated with a give source, and the mitigation case, where we handle a detected loss of trust.

The evaluation case involves first *sensing* the current state of a system. Then, involved components must come to an *evaluation* of recorded sensory input. This evaluation, as well as the initial sensing step, can occur at various levels of analytical depth involving a wide range of information. Finally, if the evaluation determines that the source is no longer worthy of trust, the system as a whole automatically transitions to *mitigation.*

Mitigation implies responses ranging from switching to a hot spare, initializing new instances, restarting, reinstalling old instances, or simply removing the capability from the system. All require some amount of pre-planning; for example, excising a service from a cloud system may be acceptable if the service is ancillary, like a calendar service or perhaps a personalization service. If that service supplies, say, satellite imagery in a satellite imagery processing system on the other hand, simple removal of that functionality will generally not be well received by any user community, no matter how jaded.

Sensing

The power of deployable sensors will vary greatly depending on the type of cloud service used. In general, the more extensive the cloud environment, the more flexibility sensor designers have. IaaS environments would be the most flexible, while SaaS environments the least, leading to a hierarchy of sensor power.

IaaS services supply basic infrastructural computing needs, most commonly virtual machine (VM) instances configured with various levels of possible performance (i.e. allocated RAM, disk space, processor clock settings, and so on). In this kind of environment, sensors can be installed in virtually unlimited ways. In fact, they need not even be located only on virtual systems providing specific services – in fact, specific VMs could be dedicated to providing sensor information on infrastructures as a whole.

PaaS systems are more limited. Example systems like Heroku give users the ability to deploy applications or application-centric services onto managed infrastructure. Google and Redhat offer similar services. In this environment, sensors need to be embedded within deployed systems in order to collect meaningful information.

Finally, SaaS sensors have the smallest number of options. SaaS services usually offer some kind of computational service with a programming interface. System developers simply do not have access to the internals of these services. As a result they are relegated to only being able to monitor system state via the externally exposed interface and any data it might accept or return.

This gives us a clear hierarchy of sensor power – IaaS Sensors, followed by PaaS sensors, finally trailed by SaaS sensors. Establishing the bounds of of optimal use of these kinds of hierarchical sensors within a cloud environment has thus far been unaddressed.

Evaluation

Like sensors, evaluators also have varying degrees of possible system impact, being able to evaluate more contextual information the farther logically away that evaluator is from a given sensor.

In this kind of an application, level one evaluators would be installed with a sensor, providing the ability to quickly determine trust levels based on simple non-derived data elements. For example, a level one evaluator in a PaaS environment could send a message suggesting mitigation by routing requests to a hot spare when it detects a database unavailability error. It could in fact be required to operate under specific hard real-time constraints as well.

Level two evaluators would have a larger context, perhaps accessing a centralized trust store as well as the input from a multitude of installed sensors. At the very least, a level two evaluator would have more than a single sensor's data to work with, and would likely need to operate under more stringent performance constraints than level three evaluators, but less than level one.

Finally, level three evaluators would be able to not only access a centralized trust store, but also all sensors in the network, external context, and historical data. In this context, level three evaluators have the most time to make trust decisions, can implement the most sophisticated evaluation algorithms, and can operate under the loosest performance constraints.

Trust algorithms are hosted within evaluators. The way that evaluators interact to come to a joint decision with regard to cloud elements is an open research issue. Furthermore, the concept of a centralized trust store enables more efficient resource use in monitoring systems. Systems with low historical trust values can have more sensing and evaluation resources directed to monitor them, while systems with a higher degree of trust can be monitored correspondingly less.

Mitigation

Possible mitigation algorithms range from switching to alternative providers to going without the service completely. These options all incorporate some ratio of efficiency to cost, where the minimization of service interruption correlates to increased system maintenance costs.

Arguably the most expensive option supplying the least interruption is a fully operational but unused hot spare. The next most costly would be redundant services scaled across heterogeneous providers. While both provide fast switchover in the case of one node's compromise, they do so at a fairly high cost – the cost of maintaining the redundant services. They also however provide an additional benefit; the additional service gives evaluators the ability to compare responses and determine some kind of geometric distance between them, identifying outliers as potential problem nodes.

The least expensive option is to simply deactivate the service and do without, if possible.

Active Research Areas

Areas requiring active research within this proposal include:

- *Optimal Evaluator Configuration* – Evaluators can be deployed in a variety of locations within a given monitoring system and they can interact in many different ways, ranging from client-server or pub-sub configurations to peer-to-peer monitoring. Optimal configurations will likely vary widely depending on specific system configurations.
- *Evaluator Algorithms* – Evaluators can implement internal algorithms for determining trustworthiness ranging from simple response time metrics to more complex statistical analysis of historical data filtered with current contextual information. The cost-benefit ratios of these various approaches will need to be more clearly understood.
- *Optimal Sensor Configuration* – Like evaluators, sensors will have optimal configurations depending on system needs, and these relationships must be more clearly understood.
- *Trust Store Integration* – Finally, the configuration and impact of a trust store on computational performance and accuracy must be explored.

Work Breakdown Structure

The project is currently scheduled over the course of one year:

1. *Infrastructure Configuration and Validation* (Two Weeks): During the first two weeks of engagement, the team will assemble and validate the configuration management, issue tracking, continuous integration, and continuous delivery systems. We will ensure that stakeholders have easy assess to developed artifacts and code, as well as any supported status dashboards showing general project health such as unit and behavioral tests status and general build status. At the end of this period, we will have also assembled our development environment, saved the various components in our configuration management system, and have rolled that environment out do development staff. We will also document key information on a team wiki as well.
2. *Establishing Initial System Architecture Vision* (One month): During this stage, the team will develop an initial validated system architecture in the Department of Defense Architecture Framework (DoDAF) style. Important to note however is that the architecture will not be completely specified after this period. We will however have identified, modeled and developed viewpoints of key system aspects at a level of detail appropriate for development to start. We will continue to re-validate and refactor architectural decisions to ensure they still apply and best fit requirements. We will also conduct initial requirements analysis and use case identification as part of business architecture development during this period. We will validate our identified component hierarchies (e.g. sensor and evaluator) as well.
3. *System Architecture Validation* (Two weeks): During this period of time the development team will develop a throwaway system validating initial architectural decisions to ensure soundness. This prototype is specifically identified as a throwaway as it will not actually implement any sensor or evaluation algorithms of consequence, but will demonstrate that such algorithms could be housed at certain system locations. It will also demonstrate system data flow. Certain elements of this system may be reused in the future even though the prototype as a whole is to be discarded after development.
4. *System Development* (Nine months): Over this period, the team will incorporate, develop, and document the system as a whole, including specific algorithmic elements. We will begin assembling papers and technical reports as well, as appropriate. We will develop the system and various artifacts using short iterative development cycles, delivering completed functionality as soon as it is built via our continuous delivery system.
5. *Project Close (*One month): At the end of the project, the team will deliver all developed artifacts to stakeholders, including but not limited to developed source code and executables, development environment components, and architectural records and documentation.

Team Structure

We currently envision roles on the project including Principal Investigators, a system architect, team infrastructure specialist, and developers, spread over a team of four engineers (not necessarily full time, nor all engaged throughout system research and development). *Principal Investigators* will lead and coordinate the effort, the *system architect* will be responsible for maintaining and documentation the system architecture, ensuring that it reflects the actual system state, the *infrastructure specialist* will maintain all team infrastructure (including but not limited to version control systems, development and modeling software, and so on). Developers will be tasked only with system development work. Personnel in the first three roles will also contribute to any development work as required.