Global Information Assurance Certification Paper

Scott Clark
Submitted February 26, 2004
Signal to Noise – Monitoring Your Infrastructure
Research Topics in Information Security
GIAC GSEC Practical Assignment version 1.4b (amended August 2002)

Abstract:

Managing a technology environment can be a tough assignment. If you only have a handful of system in your environment, then maybe you can get away with managing them manually, but as the number of systems, services, applications, and users per system administrator increase, then you will need to rely on intelligent, modular automation to help you manage the system. Under the paradigm "You cannot manage what you cannot measure", the first place to start is with measurement. This implies quantifying as much as you can about your environment, and measuring those metrics both instantaneously and historically.

This paper will discuss a framework for setting up a scalable / extensible monitoring infrastructure using opensource tools, starting out very simple and providing options to extend the sophistication. It will talk about the policy, the different components, the particulars of how to design, and things to look out for in your implementation of an environment monitoring system.

There are two strategic concepts that will be realized within this:

1. You can manage with certainty your technology environment without the need for draconian measure. You will have the information necessary to detect ANY changes that happen within your environment, and therefore do not need to *exert* control, because you will *have* control.
2. Aggregating ALL the information regarding the state of your infrastructure to be accessible by a single entity (response system) will allow to make the best possible decisions in response to potential issues detected. You will be able to cross-validate any reported issue for positive identification prior to any action.

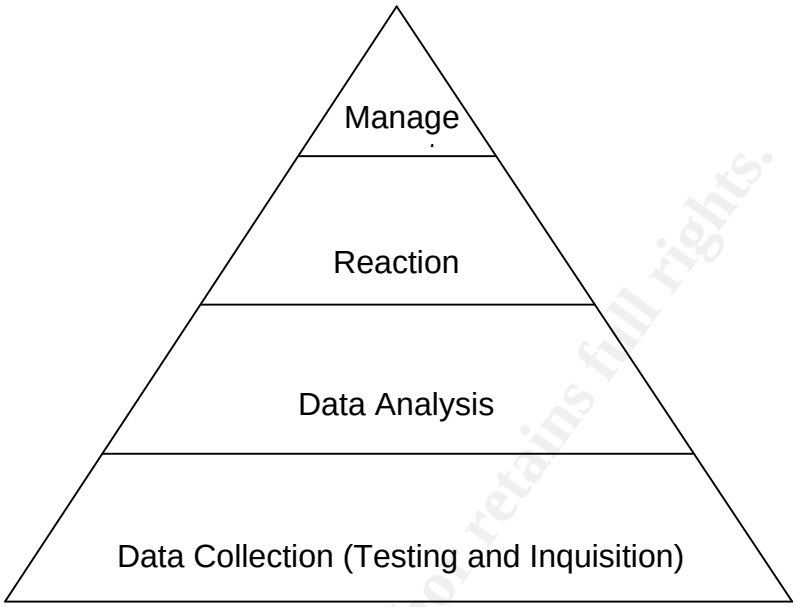
Understanding the Components of Infrastructure Monitoring:

In order to properly understand what we need in an infrastructure monitoring solution, we must first understand the component parts. The solution can be broken up into three layers, each providing service to the next: a data collection layer, a data analysis layer, and a response layer.

The data collection layer is usually a collection of simple inquiries of devices, services, or applications and the associated results. These results form the foundation for the next layer, which is data analysis. The data analysis layer must review the data, and based on threshold definitions, make decisions on whether or not to initiate an event. If an event is generated, that will then stimulate the response layer, which can range from generating a simple log entry all the way

up to taking automated responsive action, and can handle various types of events, managing them differently.

It is best to imagine the components as a hierarchical pyramid[1]:



Each layer acts as a supplier to the layer above (consumer layer), supplying data as input to the consumer layer, with the consumer layer providing data processing, imbuing the data with value, and transforming it into information prior to supplying that information as input to the next subsequent layer.

The final distinction is between instantaneous and historical monitoring. Instantaneously, you will need to know the health and status of your environment at any point in time in order to efficiently address issues that urgently need resolution (system down, service broken, etc.). Historically, you will be interested in tracking information over time, so you can look at trends to see if you will need to proactively address capacity issues (at the current utilization rate, you will be out of disk space in 1 month, network throughput to a remote site is going to hit a wall in 3 months given the current growth rate, etc.). Another advantage to historical trending would be to understand anomalous behavior (We had no problems with this system until 2 weeks ago, and it has been unstable for the last two weeks).

[1] Pyramid concept and management top block taken from “Building the New Enterprise: People, Process, and Technologies” by kern, Johnson, Galup, Horgan, Cappel (P21).

Designing the solution:

The Goal of the Design:

- Gain control of your environment, so that no change can happen within the environment that you don't know about (because your monitors caught those changes).
- Create a solution that is concurrently aware all the facets of your environment with which to make decisions.
- Create a solution that is easy to setup, manage, and use.
- Create a single place to look for information regarding health and status of your infrastructure.
- Leverage a modular approach to monitoring, creating a solution (framework) that can accommodate existing or envisioned monitoring tools for any monitoring you would need to manage your environment.

The goal of the design would be to create a framework that would emulate the military command and control model. It would leverage an army of data collectors to feed a single decision system in order to make the best possible decision with regard to a given situation having an understanding of ALL the information that is available, not just the single pointed stimulus that raised the issue as a potential problem (what a point tool would yield).

The framework should be built hierarchically, with the data collection layer reporting to the data analysis layer, that would in turn report to the response layer, which can then make decisions based on the input of ALL the data that has been collected. The intentional design of this solution is to give the decision point (response system) the ability to validate the issue from many different directions, significantly increasing the accuracy of the reported issue prior to any response or action.

The hierarchy would serve additionally to isolate the functions of data collection, data analysis, and response, allowing each one of those layers to be focused on that specific function. This emulates the OSI protocol standards model (3 layers, not 7) for the same benefits that OSI has observed[2]:

- Easier for humans to discuss and learn
- Allows integration of partial solutions (Facilitates Modular Engineering)
- Creates better environment for interoperability
- Reduces complexity, allowing easy programming changes and faster integration
- The layer below another layer provides a service to the higher layer.

[2] These concepts were referenced from CCNA Exam Certification Guide, CCNA Exam 640-607, P75-81, Wendal Odom

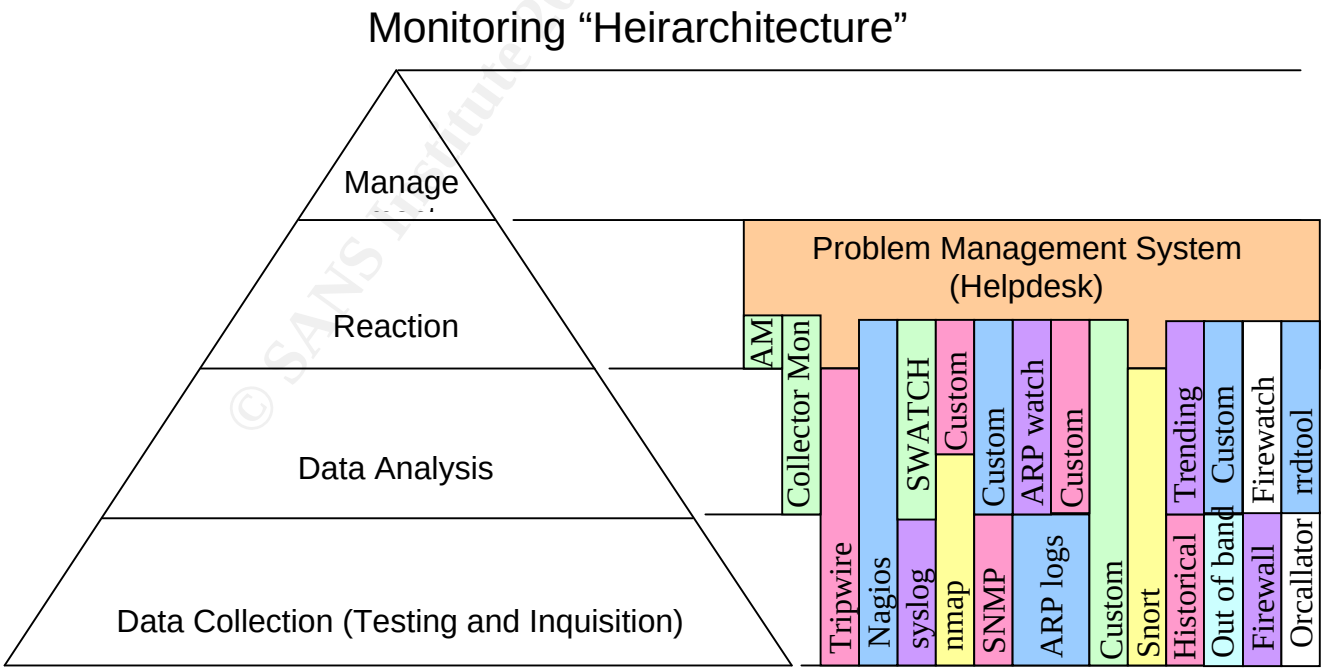
The framework would manage all communication and action, allowing you to glue together specialized, intelligent modules to perform the tasks at each layer. You would be able to leverage any work that you do across all monitoring tools, as compared to having to add intelligence to each package that you are using to monitor different aspects of your environment. Configuration of the response system would happen infrequently, and as new monitors are installed, all that would be needed is to define the new monitor within the existing framework, and configure the monitor itself. There would be no additional management required.

Finally, this model would create a single point to inquire for any issue within your environment. The response system would have knowledge of every portion of your environment that would be under investigation or declared defective.

Set up for Extensibility

This is a modular framework that would accommodate any type of monitor that you can think of or that has already been developed, and leverage the common parts of the framework to create a single solution for monitoring your environment in its entirety.

Because many of the current monitoring tools have their own methods for communicating events, attempting to leverage these tools collectively would create many point solutions within your environment as compared to creating a single, extensible monitoring solution. While having fault tolerance in your monitoring system is a good thing, having every system have its own communication process and channel would get out of hand quickly.



A helpdesk (Problem Management System) is the unifying technology for aggregating the monitoring of your infrastructure. This gives you ONE point of control for notification and action, allowing you to initiate an informed and appropriate response (properly filtered to respond according to criticality/urgency, having verified the issue with additional input).

This way, you can have as many systems as you would like to monitor your infrastructure, detecting potential issue, and have them interface to a single helpdesk package. This allows the helpdesk package to handle ticket (issue) management, status management, classification, categorization, escalation management, action management (communication would be handled in this step), logging, resolution tracking, knowledge base management, reporting, and overall management of how to react once a potential issue has been identified. Depending on the sophistication of the helpdesk package used, this allows for a varied and granular response to any number of events, including automation of additional inquisition for clarification of the potential issues, or even fully automated rectification of the event based on how thorough the understanding of the issue is. In this manner, you can focus the customization for your environment, and the majority of the effort on the response system (helpdesk). You can then leverage the plethora of existing opensource tools (without modification) that are available to provide the data collection and analysis layers, for monitoring your environment and detecting potential issue.[3]

Start Small:

Begin your adventures into sleepless purgatory by integrating a single monitor into the framework. This will allow you to work out the kinks of your framework and reduce the signal to noise ratio to something manageable for that one monitor. Once you have control of that monitor, then select one more monitor, and integrate that. Take it one tool at a time so that you do not inundate yourself with false-positive issues, and deteriorating the confidence in the monitoring solution. Once support staffs lose confidence in the solution, they tend to not pay attention to any notifications, assuming the it will be yet another false-positive. At that point, your monitoring solution, no matter how elegant, is useless.

You should also be ready with each monitor before you integrate it into your framework. Have an understanding of what is a “good” state and what is a “bad” state for what you are trying to monitor. Otherwise, you will end up, once again, inundating your sysadmins with false-positives and they will lose faith in the solution as a tool. All monitoring is relative to some reference data, so you should be able to define the reference data prior to initiating monitoring. For every monitor, there is an integration period where you are getting your infrastructure to a “healthy” state with respect to the monitor, as well as a period of time to get the monitor to a “stable” configuration. It is recommended to not turn on notification until the monitor has reached a “healthy” AND “stable” state.

[3] Concepts for using a helpdesk as an aggregation point came from a discussion with the original designers of OpenARGUS http://openargus.sourceforge.net

Gain Control / Retain Control:

As you bring online more monitors, you will eventually reach a state where you are monitoring all (or almost all) aspects of your environment. In the context that we are using the term, "control" revolves around configuration management, and the monitors associated with that facet of the solution. You will be monitoring all your systems for changes to key files (hostbased intrusion detection – tripwire[4], nmap[5] queries for new hostbased services/applications), monitoring your network for changes in configuration (arp table queries of network gear, nmap scans for new IP's, nmap queries for new network services, tripwire[6] on the running configurations for all network devices), monitoring firewall logs for intrusion attempts (firewatch[7]), monitoring incoming traffic for suspect traffic patterns (SNORT[8]), etc. Now, any change that your monitors detect is an indication that something is different. You will know from your monitors EXACTLY what is running on your network, and by tracking changes, you will gain an upper hand in the battle to "manage" your environment.

This leads us to a very important paradigm in security: "What you cannot monitor, prevent, and what you cannot prevent, monitor"[9]. Knowing precisely when someone adds a system or service to your network, and being able to tell where, when, and what changed (MAC address, port number, office/cube, what system is now running a Napster service, etc.) is critical to you being able to retain control of your environment once you have gained it.

The revelation we should take away from this is that we can now manage our environment by leveraging information and technology, and not through draconian measures, as is common in our industry.

[4] Tripwire – http://www.tripwire.com/products/servers or alternately, http://www.tripwire.org for the opensource version of Tripwire
[5] NMAP – http://www.insecure.org/nmap
[6] Tripwire – http://www.tripwire.com/products/network_devices
[7] Firewatch – http://www.bellcore.com/SECURITY/firewatch.html
[8] SNORT – http://www.snort.org
[9] This paradigm taken from many SANS, USENIX, and LISA conferences. Credit to the security community at large.

A few words about requirements and tool selection:

In defining the requirements of the monitoring system, there is no simple checklist that you can follow to determine the "right" solution or tell you what package to pick. Some things to consider when evaluating this would be:

- What is the budget for the monitoring solution? This will drive your primary decision point. The commercial tools have an initial cost factor that the opensource tools would not. There will be associated cost to configuration, support, maintenance, and customizations for both commercial and opensource solutions that will depend on the skillsets you have available to you.
- What is it you need to monitor? There is a large variety of tools available.[10]
- How sophisticated does the tool need to be? - Do you need to be monitoring memory utilizations for Oracle database servers, or is it sufficient to be able to ping the systems to know that they are connected to the network and have power to them? What targets do you have that require sophistication and how many targets require sophistication?
- How many total nodes are you monitoring? – This might drive the number of data collection systems you might need, or the location of the data collection systems.
- How many sites are to be monitored? – This would probably drive the distribution of your data collection systems, depending on the size of the site.
- Do you have access to the facilities at any remote sites that need to be monitored? – Would it be possible to install a data collection system onsite or do you have to monitor them remotely?
- Do you have access to the systems to be monitored? – Can you install client software or do you need to query the systems from the outside only?
- How many support personnel do you have that will be using the monitoring tool? – What type of interface do you need to put on the monitoring system?
- How sophisticated are the users of the tool? Are the support personnel capable of managing complex solutions, or do you need to make it intuitive and fully automated? Are the support resources capable of making customizations to the tool, or are do you need out-of-the-box solutions?
- What type of support do you need for the overall monitoring solution? Does your company need 7X24 access to commercial support hotlines for the monitoring solution, or can you get away with the opensource model.

Based on the requirements definition, select the tools that best reflect the situation for your environment. There might not be a single tool that encompasses all the facets that you will want to monitor, so your implementation

[10] An overview of many of the available tools and what they do is available at http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html

probably will be a hybrid of solutions, gluing many tools together to create a single aggregate monitoring "solution". There are commercial monitoring platforms that have aggregate monitoring frameworks for monitoring many different things within the same solution. Tools like HP/Openview[11], and IBM/Tivoli[12] are two such solutions that have been around for many years, and are fairly mature in their development. Opensource tools offer a cost effective alternative for environments that have the resources capable of executing the integration and customizations required. Two tools that apply for a good portion of our monitoring needs are SNIPS[13] and Nagios[14], with Tripwire and NMAP filling other significant holes.

Data Collection System

The first thing that needs to be addressed is to understand what is it that you want to monitor (and therefore what data needs to be collected). Some potential options are[15]:

- Fault Management
  - Monitor system availability (ping test and round trip time – SNIPS or Nagios)
  - Monitor application availability (port response test – custom)
  - Monitor for application integrity (functional test - custom)
  - Monitor network availability (ping test and round trip time – SNIPS or Nagios)
- Security Management
  - Monitor log files (syslog collection, firewall log collection, application log collection)
  - Monitor Network Traffic (SNORT, Arpwatch[16], custom)
- Configuration Management
  - Monitor system configuration (tripwire evaluation)
  - Monitor system configuration (nmap port scan of system for services running, SATAN[17])
  - Monitor network configuration (tripwire evaluation on network device's running config)
  - Monitor network configuration (nmap address space evaluation)
  - Monitor network configuration (network device arp table query)
- Performance Management
  - Monitor capacity points (client utility query – SNIPS, Nagios, SNMP[18], or custom)

[11] HP/Openview - http://www.openview.hp.com/
[12] IBM/Tivoli - http://www.tivoli.com
[13] SNIPS - formerly NOCOL - http://www.netplex-tech.com/snips/
[14] Nagios - formerly NetSaint – http://www.nagios.org/
[15] Categorizations taken from Network Monitoring Explained: Design and Application By: Dah Ming Chiu (p29-34)
[16] ArpWatch - http://www.securityfocus.com/tools/142
[17] SATAN - http://ciac.llnl.gov/ciac/ToolsUnixNetSec.html#Satan
[18] SNMP - http://sourceforge.net/projects/net-snmp

    - Monitor for capacity points (performance validation test – SNIPS, Nagios, Orcallator[19], or custom)
- Capacity Management
    - Monitor for predictive capacity limits (historical logs of data collections)

Once you have you have an understanding of what data is to be collected, there are several implications that need to be addressed:

- Providing instantaneous and historical data collection
- Placement of data collection system
    - Network latency created by data collector placement
    - Effective DoS created by overloading any one data collector system or network segment
    - Dependencies created within monitoring system by placement of data collector system
- Check for availability of the data collector from another point
- Data comparison integrity and security
- Data collection client software can impact client performance
- How to keep data collectors up to date with respect to what data to collect

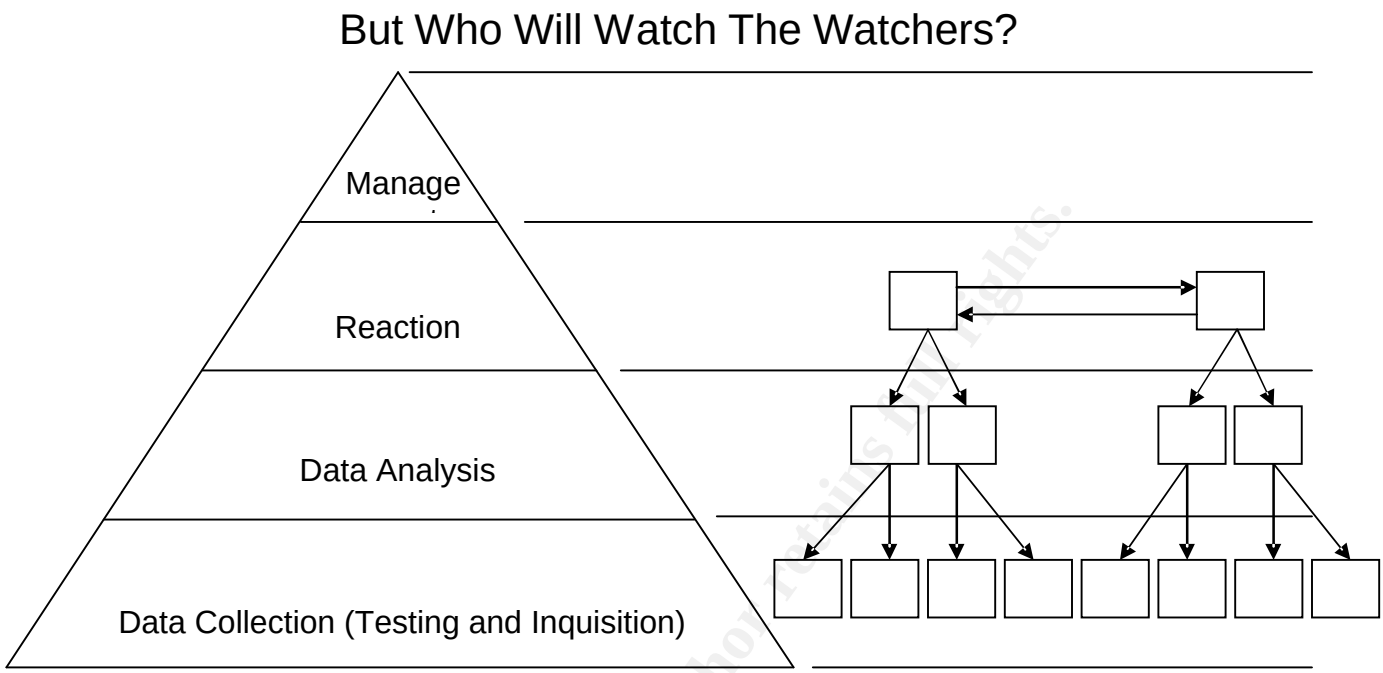
Your data collection system needs to be collecting data for the here-and-now, so that you will know the status of your network and any given time. But you will also need to track those results over time, so that you can go back and review historical information regarding your infrastructure, and assemble availability statistics (looking at all dependencies when you are doing your calculations), trending analysis (when will you need to increase capacity?), and reliability assessments (when did we start seeing trouble with the component in question?).

The placement of the data collection system also needs to be reviewed. If you have multiple subnets or multiple sites that you are monitoring, then you should consider placing a data collection system local to the subnet or site to minimize traffic across the routers and WAN. In considering this, look at how many systems it is that you will be monitoring across the link, the frequency needed of the monitor, the link speed, the link latency, and the type of monitors that will be used across subnets to help you determine if you will need to allocate a system for monitoring that would be local to the subnet. The issue you would be most concerned about with monitor location would be link loading – understanding how much traffic you will be putting across an interface collectively, and what that will do to that link.

If you have multiple data collection systems, another important point to remember is to have each of those data collection systems to be monitored by another system for availability. This is to make sure that you don't have an

[19] Orcallator – http://www.orcaware.com

outage in the devices that are supposed to tell you that other devices are not operating properly – no news is not good news in this case.



Each level of the solution has to monitor the previous level and all related dependencies for availability and function in order to properly manage notification. For example, if the network is down to the data collection system, you would only want the response system to inform you that the network is down, not inform you that every system on your network is down (data storm).

It is important to make the each facet of the monitoring systems as standalone as possible so that outages (what you are monitoring for) do not impact the monitoring function. Have small, localized monitoring systems that monitor the supplier service (as we talked about in layers) for availability. If the supplier is not available, raise the alarms indicating that the service provider (previous layer) is unavailable…

For data collection functions that need to share data with other applications, it is important to build resiliency into how the data collectors access the centralized data space. In the UNIX world, this would mean using NFS hard mounts with the INTR option set so that the systems can function in the event that the file server has a failure (it is understood that monitors dependant on the data space in question would not function in this case). One important reason to have data located in a central location would be for extensibility purposes. In the event that you need to use multiple data collectors for the same data type (number of systems being tested drives multiple monitors for throughput, or to avoiding a router hop, etc.), having the data centralized would allow you to collect data using multiple sensors, and still do the analysis with a single machine.

Another facet to understand is the access privileges that are required for each data collection system. This is especially important in the case of configuration management / data integrity assessment. When doing data integrity assessments, the reference data (data that is being compared against) would need to be mounted read-only, as well as the executables that are doing the comparison. This will ensure that your integrity check is accurate. You will also need to figure out a method of being able to commit changes to your reference files, given that you have a read-only mount to those files. The majority of these monitors will be security related, and therefore has more sensitivity.

Sometimes it will be better to only collect data from the outside of a system, this being on systems where performance is critical and close to a capacity point. The act of monitoring on the system itself might steal resources that need to be used for the function of the system, and we do not want the act of monitoring to directly impact the performance of the system (a very bad twist on the Heisenberg Principle).[20] This is left to your discretion for when to have client code and when to not have client code, but CPU intensive systems are probably better monitored from afar…

Another troublesome aspect is how to keep the configuration files for your monitors up-to-date. With your network and infrastructure changing regularly, how is it possible to keep the monitors watching for ALL the systems, applications, and services that you are running. The simple answer is "by monitoring" of course. Set up monitors to continuously scan your environment for new systems, applications, and services being introduced, and have the monitoring system flag new entrants. If they are legitimate, then you can have an option to add them to the monitoring configuration (and any other tracking system that you have). A good additional feature to a helpdesk system is to have an asset database attached. This is a good point to pull the initial information for your configuration files, and have your automation update both (the asset database and your configuration files) as you go forward. You can also use your monitoring system to validate other types of configuration files. You can verify NIS tables, DNS entries, LDAP tables (maybe your asset management system consists of LDAP tables), Active Directory tables, etc. You can monitor anything you can think of (an probably should!).

Definitions for Data Collectors

Data Collection Type	Description
Service / Application Availability	A request to the service or application to determine if the application or service is running. A good example of this would be a sample SQL call to verify that an Oracle

[20] The Heisenburg Uncertainty Principle, Werner Heisenberg

	database is running. Another example would be a named query to a BIND server to verify that DNS was responding.
System Availability	A request to the system to determine if the system is responding. A good example of this is ping.
Client Query	A request against a client process running on the system that is running specifically for data collection purposes. An example of this would be SNMP
Performance Query	A request to a system, application, or service to determine the performance of that system/service/application. Good examples of this would be round trip time (determining latency) or throughput tests.
Integrity Query	An inquiry of a dataset to determine if the dataset has been altered. A good example of this is an MD5 hash of a file set. This would be compared against a known good hash, and any difference would indicate a change in the files contained within the dataset. This is the premise that Tripwire has been built upon.
Data Collection	The collection of data for later analysis. Good examples of this are application log files, syslog service collection, network based IDS (SNORT, TCPDump, etc.).

Data Collection Schedule	Description
Continuous	This schedule corresponds to data collections that should be run continuously within the bounds of physical constraints and congestion avoidance. Always do the math on the connection that you are
Scheduled	This could be hourly, daily, weekly, monthly, or any other scheduled event.
On Demand	On-the-fly data collection for validation purposes, instantaneous assessment, etc.

Data Analysis System

As with the data collectors, making the analysis systems as standalone as possible is just as important (and will continue to be a theme throughout this design). So making any NFS mounts be interruptible is just as important here, so that the function of the system (data analysis) can continue in the event that the file server is the point of failure. Also, as with the data collection systems, the analysis systems benefit from using centralized storage by being able to add data analysis systems as load requires. The analysis processes need to be

broken up among the systems, but many systems can simultaneously participate in the analysis process, leveraging the data from a common point.

One function of the data analysis layer is to have the ability to communicate with the systems that are being analyzed out of band. One of the analysis processes will be to verify the system status through an alternate path (other than the network). The usual configuration for this is with system specially configured for serial port connectivity (for UNIX systems) that can usually aggregate up to 48 serial ports per system. Some solutions can be found from Aurora Technologies[21] as well as from Mirapath.[22] Out of band management combined with Remote Power Boots will yield a fully remote manageable solution with the exception of physical hardware replacement. Mirapath provides a very interesting remote power management solution that is daisy-chainable and Western Telematic[23] also provides a solid solution, which facilitates additional response options (allows you to power cycle systems even if the system is completely non-responsive, which can be verified through the out of band connection).

The reason to have multiple data analysis systems would be if the processing power required to perform the analysis required more processing power than a single system requires. This allows you to use older system with perhaps less processing power in tandem to accomplish the same result. This obviously implies that one of the capacity points that you will be monitoring instantaneously and historically would be CPU usage on the analysis system, but that is exactly what we are trying to design – a system for monitoring any and all aspects of the environment.

We will also need the ability to integrate any new analysis functions and interface all to the response layer, and the response layer will have all the intelligence with respect to what to do next, what the event means, if there are dependencies, and if this analysis result is related to an issue.

The concept of an event ID (probably most commonly referred to as a trouble ticket ID) needs to be tracked at this level, even though we have not yet discussed the response system. If an analysis has been performed, and the analysis system was not passed an event ID, that would imply that it is a new event, and have an ID of (-1) until it is passed to the response system to be assigned a unique ID. In the event that the response system needs additional analysis performed, it will make a request of the analysis layer, and pass the event ID to the analysis layer to update the status of the event with the result of the analysis (for example, given the system does not respond to pings, check the out-of-band status of the system).

---

[21] Aurora Control Tower http://www.auroratech.com/
[22] Mirapath Cyclades TS and Alterpath ACS http://www.mirapath.com/products/cycladesindex.htm
[23] Western Telematic http://www.wti.com

Definitions for Analysis Systems

Data Analysis Type	Description
Command Status Analysis	The results of a ping request, port connection, etc.
Expected Response Analysis	A sample database query will have an expected result. HTTP queries…
Threshold Analysis	Performance values will be greater than or equal to a threshold value. Resource capacity points, round trip times, throughput tests, snmp values
Integrity Analysis	Data integrity, tripwire will either be equal or not
Key phrase Analysis	SWATCH, Shadow/SNORT, application log analysis
Trend Analysis	Historical trending of all monitored events
Custom Analysis	As it says

Data Analysis Schedule	Description
Continuous	SWATCH, log analysis, etc.
On demand	Any of the analysis
Scheduled	Every 5 minutes, 15 minutes, hourly, daily, weekly

Event Response System

This is the heart of the design, to have a single system (per monitoring infrastructure) as the command and control center for all monitored events. This system would have all the information regarding dependencies, how to communicate, when to communicate, how to escalate, when to escalate, any automated actions that can be taken, etc. This gives you a single focal point for assessing issues within your environment.

Requirements of the response system:

- Track status
- Assign categorization
- Assign classification
- Escalation tracking
- Trigger action based on status change, escalation criteria, categorization criteria, or classification criteria (state change).
- Leverage the analysis system to transform data into information
- Leverage other response systems to transform data into information

- Checking for freshness of the data – would indicate that a data collector is probably down or not working properly.
- Coordinate ticket information with the analysis system for additional action relative to an issue that has been raised

Some opensource options for your helpdesk package are:
Tuxmonkey Issue Tracking system[24] and Request Tracker[25]

Programming Policy:

The response system will need to have an understanding of conditions for your environment (what are the circumstances that indicate a fault state for each system, service, application, or other monitored event). This will be comprised of a collection of measurements, with different combinations indicating different potential faults. You will then need to have responses defined for each variation of fault, based on how the particular fault impacts the operation of your environment. You will want to drive this based on the priority of the most urgent function associated (including anything that is dependant) with the monitored object. This will determine how fast you respond, how quickly you escalate, what form of response, who would be notified, and during what hours you respond for the identified issue.

This allows you to apply conditions to faults that are flagged by the lower layer monitoring systems (data collection and analysis layers), thus imbuing them with increased accuracy. In a normal monitoring infrastructure, the failure of a ping monitor to a host would result in a fault being flagged to that system, and generating a notification alpha-page to the system administrator. In our enhanced model, we will only generate the alpha-page to the system administrator if the ping test fails AND the out-of-band test fails as well. If the out-of-band test is successful, this will trigger additional tests to verify the function of the switch, individual switch port of the system in question, interface on the system through the out-of-band system, and continue until we have enough information to draw a conclusion as to the exact location of the fault. You can use automation to run tests and attempt to recover the fault. In the event that you unable to recover the fault in a predetermined period of time based on criticality (timer kicked off when the initial fault is received from the analysis layer), you would THEN generate an alpha-page to the appropriate administrator to notify them that there is an issue that is unresolved, and in need of immediate attention.

It would be at this level that you will be able to program your entire configuration into the system that can help determine more precisely where the REAL problems lie. You can define dependencies associated with monitored items, and if the monitored item is down, then check the objects that it is dependant on. If

[24] TuxMonkey Issue Tracking System http://www.tuxmonkey.com
[25] Request Tracker http://www.fsck.com

one or more of those objects are down, then you step in to the next layer in the dependency chart, and make your dependency check again. You can continue this until you identify the object farthest in on the dependency tree (closest to the base) that is in a fault state, and report on that object, and not on every object in the tree. This adds a tremendous amount of intelligence and improves the accuracy of your solution.

It is important to also build fault tolerance into your monitoring system. You should have monitors watching the monitors for availability, and sending notifications in the event that a fault is discovered. It is also interesting to have your system send out a regular (once a day during normal work hours) test message during non-intrusive hours so that you will know from the system itself that it is operating properly. If you do not get that regular message, you know to go look.

Definitions for Response System

Object Classification type	Description
Project	Tracked in weeks, not minutes
Non-Critical	Tracked during work hours, and is concerned with individual productivity. Tracked in hours, not minutes.
Important	This is an escalation process for systems that have import functions, but less devastating implications. Tracked in single digit hourly increments (1, 2, 4), and is fairly aggressive about escalating when no progress is being made.
Critical	The most aggressive escalation process. These are for absolutely critical systems/services that the company is taking financial impact from by being unavailable. A highly aggressive escalation schedule dealt with in ¼ hour increments (15 minutes).

Object Categorization Type	Description
System Group [1-99]	Helps identify who to notify
Service Group [1-99]	Helps identify who to notify
Application Group [1-99]	Helps identify who to notify
Network Device [1-99]	Helps identify who to notify

Event Status Type	Description
New	Fresh out of the analysis layer
Responded	An action has been taken, and the issues is in process of resolution.
Closed	All done!

Event Response Type	
Log (passive notification)	This can be local log files or can be collective logs that are centrally registered. This would require action (parsing the logs) on the part of the administrator to be notified of an issue.
Web page update (passive notification)	Updating a web page to indicate a fault. This would require action (checking the web page) on the part of the administrator to be notified of an issue
Email notification (passive notification)	Email notification can be either passive or active at the same time. An email to an administrators work email account would require the administrator to log in a check his/her email to see an event message, but with newer technology, this can also just as easily land on a Blackberry device or cell phone.
System audible / visual alarms (active notification)	Perhaps your site is fortunate enough to have 24X7 staff that is tasked with performing round the clock services (back-ups, help-center, fault monitoring, NOC, etc.). Sending audible signals to a system in the proximity of these individuals would notify them that there is an event in need of attention.
Alpha-numeric page (active notification)	This considered active, but is susceptible to device configuration management (having good batteries in the pager or cell phone.). Count the times you have heard *that* excuse ☺
Scripted Response	This is for when you have gained enough history with a problem, and can say with certainty what actions to take. For example, if when a system running a MS OS stops responding to pings, you can probably rest assured that sending the out-of-band management system a power cycle for the power port for that system would be a safe (and probably the recommended) response to the issue.

Reference:

[1] Kern, Johnson, Galup, Horgan, Cappel
Sun Microsystems, 1998
Pyramid concept and management top block
Building the New Enterprise: People, Process, and Technology (P 21)

[2] Odom, Wendal.
CCNA Exam Certification Guide
CCNA Exam 640-607, P75-81

[3] Bell, Chris; Burroughs, Eric
Concepts for helpdesk as an aggregation point
Original designers of OpenARGUS
http://openargus.sourceforge.net

[4] Tripwire
http://www.tripwire.com/products/servers or alternately,
http://www.tripwire.org for the opensource version of Tripwire

[5] NMAP
http://www.insecure.org/nmap

[6] Tripwire
http://www.tripwire.com/products/network_devices

[7] Firewatch
http://www.bellcore.com/SECURITY/firewatch.html

[8] SNORT
http://www.snort.org

[9] General Security Paradigm
"What you cannot prevent, monitor, and what you cannot monitor, prevent."
SANS, USENIX, SAGE conferences

[10] An overview of many of the available tools and what they do
http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html

[11] HP/Openview
http://www.openview.hp.com/about/index.html

[12] IBM/Tivoli
http://www-306.ibm.com/software/tivoli/features/oct2003

[13] SNIPS - formerly NOCOL

http://www.netplex-tech.com/snips/

[14] Nagios - formerly NetSaint
http://www.nagios.org/about.php

[15] Categorizations taken from Network Monitoring Explained: Design and Application By: Dah Ming Chiu (p29-34)

[16] ArpWatch
http://www.securityfocus.com/tools/142

[17] SATAN -
http://ciac.llnl.gov/ciac/ToolsUnixNetSec.html#Satan

[18] SNMP
http://sourceforge.net/projects/net-snmp

[19] Orcallator
http://www.orcaware.com/orca/docs/orca.html

[20] Werner Heisenberg
The Heisenburg Uncertainty Principle

[21] Aurora Control Tower
http://www.auroratech.com/consolemanagement/ct/

[22] Mirapath Cyclades TS and Alterpath ACS
http://www.mirapath.com/products/cycladesindex.htm

[23] Western Telematic
http://www.wti.com/power.htm

[24] TuxMonkey Issue Tracking System
http://issue-tracker.sourceforge.net/

[25] Request Tracker
http://bestpractical.com/rt/