LiveOps: Systems Management as a Service

Chad Verbowski – Microsoft Research
Juhan Lee and Xiaogang Liu – Microsoft MSN
Roussi Roussev – Florida Institute of Technology
Yi-Min Wang – Microsoft Research

ABSTRACT

Existing Management Systems do not detect the most time-consuming and technically difficult anomalies administrators encounter. Oppenheimer [25] found that 33% of outages were caused by human error and that 76% of the time taken to resolve an outage was taken by humans determining what change was needed. Defining anomaly detection rules is challenging and often cannot be shared across organizations. It requires a deep combined knowledge of the software, workload, system configuration, and tuning parameters specific to the workload and overall distributed application topology.

We present LiveOps, a scalable systems and security management service based on auditing the interactions between applications and the persistent state they use [33]. This approach simplifies identifying security vulnerabilities, performs compliance auditing, enables forensic investigations, detects patching problems, optimizes troubleshooting, and detects malware/ intrusions. The service enables knowledge sharing across organizations and administrative boundaries and allows for seamless integration between analysis results from disparate management products that build on it. Our configuration-free agent collects all read and write access to registry entries, files, binaries, and process creation. The agents streaming lossless compression creates log files of only 20 MB per day containing an average of 45 million events. The scalable LiveOps back-end service can analyze 1000 machine days of logs in 30 minutes. LiveOps agents have been deployed on 1149 machines from home systems to corporate desktops, including 381 production MSN servers across 11 sites.

Introduction

MSN System administrators spend a third of their time managing system configuration despite the use of state-of-the-art systems management solutions. This not only reduces the number of systems that a single administrator can effectively manage, but is a significant liability in overall system reliability.

Several examples of configuration errors impacting system reliability are described in [33], where at one MSN site, 70% of persistent errors were found to be persistent state (PS) related, and 28% of support calls at a large software company's help desk were configuration related. Furthermore, a lack of understanding about the impact configuration changes have on critical applications can delay the deployment of critical security patches [2, 28, 34], and has been found to be responsible for 76% of the time taken to resolve datacenter outages [25].

Existing systems management solutions are difficult to implement, expensive to deploy, and are largely ineffective at addressing the most costly systems and security management problems. Furthermore, for scalability and performance reasons they do not track all changes, or the interactions between applications, users and PS. Administrators must manually apply their experience and application knowledge to filter the subset of PS changes recorded by these systems to identify only the PS changes impacting their applications.

Defining anomaly detection rules based on application generated events [31, 40] is challenging and often cannot be shared across organizations. It requires a deep combined knowledge of the software, workload, system configuration, and tuning parameters that are specific to the workload and overall distributed application topology.

Software developers and systems management solution vendors are restricted to creating rules that are based on the available application events, and that are locally tunable or independent of workload, system configuration, and distributed application topology. Administrators are left with the challenge of determining the appropriate tuning parameters for existing rules and are required to identify and codify the bulk of rules needed to manage their environments. Sharing rules between organizations is further complicated by variations in tuning parameters and rule definition languages.

We present LiveOps, a scalable systems and security management service based on auditing the interactions between applications and persistent state they use

[33]. This approach simplifies identifying security vulnerabilities, performs compliance auditing, enables forensic investigations, detects patching problems, optimizes troubleshooting, and detects malware/intrusions.

The LiveOps architecture consists of agents running on the managed systems that report logs to a back-end service which processes the data and provides extensible interface for generating of web reports, alerts, or integrating with other management products. Our configuration-free agent collects all read and write access to registry entries, files, binary loads, and process creation. The agent's streaming lossless compression creates log files of only 20 MB per day containing an average of 45 million events. The scalable LiveOps back-end service can analyze 1000 machine days of logs in 30 minutes

The key contribution of LiveOps is scalable and complete configuration monitoring, comprehensive anomaly detection through cross machine and cross time baseline analysis in an organization, and globally across all machines reporting to the service, enabling self-tuning of rules with respect to workload and topology. Secondly LiveOps' ability to manage 4000+ monitored machines via a single back-end server enables it to be run as a service supporting multiple organizations.

As a service, LiveOps provides immediate benefit to all subscribers when new rules are developed. Using a service rather than a locally installed management system eliminates the cost of maintaining a systems management solution, enables IT departments to draw on the expertise of other administrators, and potentially alerts the original developers of a problematic application. When rare and difficult issues are encountered, the service provides the specific management details that enable collaboration with experts that can help resolve the issue. Furthermore, after an issue is resolved the relevant centralized data can be annotated to benefit other users experiencing a similar problem.

We present our experiences and analysis of the alerts and reports collected from running LiveOps in MSN. LiveOps agents have been deployed on 1149 machines, from home systems to corporate desktops, and including 381 production MSN servers across 11 sites. In the next section, we describe the motivation for creating LiveOps and the related work comparing it with existing management approaches. We subsequently describe the LiveOps architecture and present the management scenarios that LiveOps addresses, provides sample reports, and a summary of results obtained from analyzing the reports from MSN production servers. We then present a data analysis of the PS used in a production environment, and an analysis of the feasibility of classifying them. Finally, we conclude.

Motivation and Related Work

The creation of LiveOps was motivated by the discovery that infrequently occurring unique configuration issues have a large reliability impact, require the most administrator time, and are the most technically challenging to resolve. Traditional management products [21] are largely ineffective at managing these problems because they rely on the monitored applications to log events [31, 40] when they are malfunctioning.

It is unrealistic to expect developers of the application to have correctly anticipated all possible problems that may occur with the application internally – from integrating with other applications, OS interactions, and interacting with distributed systems such as databases, firewalls, web servers, and network related services. Events logged by applications are best used for automating responses to frequently occurring problems that are well understood, but seldom provide sufficient insight for administrators to enable quick resolution of issues unanticipated by the original application developers. The following example of a problem at a large MSN site illustrates how understanding the impact of individual PS changes on an application enables quick problem resolution:

> An administrator was assigned to resolve an intermittent web page issue with a large online site. During the course of solving the problem, a critical configuration file was inadvertently deleted. After the intermittency issue was resolved, the site appeared fixed and the issue was closed.
>
> After 18 hours it was discovered that the site was experiencing a partial outage. Investigation of the outage involved teams of engineers, and lasted for 27 more hours, before the originally modified configuration file was discovered as the root cause. With a record of what PS had been changed on each machine, the investigators could have quickly and easily identified the configuration file as a root cause candidate.

The next example illustrates how failing to verify the complete set of files and settings modified after a patch installation can prevent diagnosis of configuration problems:

> A patch was deployed on servers across a large MSN site. Later it was found that the site was experiencing a partial outage after the installation, where some users experienced poor performance or received an error message after refreshing their web page two out of five times.
>
> Since the site load balances incoming requests over a large pool of servers, it was difficult to determine which servers were causing the problem. It took two engineering teams approximately 72 hours to determine that the root cause was that one of the servers had only received partial settings during the recent patch install.

The LiveOps approach is to manage configuration from the OS perspective of interactions between running processes and PS. This differs from asserting correctness constraints on subsets configuration for the system as a whole, as done with CFEngine [5], because using interactions enables consideration of the subset of PS that are used by each application. Overall this reduces the volume of PS to consider because less than 15% of non-temporary registry and files are typical in daily use [32].

We believe this is more effective than the traditional approach of analyzing application logs for the following reasons:

1. It is a non-participatory model, meaning that all changes made by applications are tracked regardless of the APIs they use or logs they create.
2. Only a few well-defined event types from a single source are required for monitoring all applications, as opposed to multiple event logs containing application specific events.
3. If the specific root cause of a problem is not obvious the information provided reduces potential root cause candidates, making problems easier to solve.
4. When the root cause of the problem is found, future incarnations of the problem can be avoided through knowing which process was used by whom at which specific time to make the breaking change.

Another approach to configuration management is to eliminate the need to identify the root cause of problems by following a non-traditional approach of designing applications with the expectation that they will fail [4], as opposed to attempting to eliminate all possible errors. This is achieved by making them quick to install and restart, and by creating software probes that remotely monitor the application to detect failures and restart the application when failures are detected. While this approach minimizes the time spent by operators investigating problems, the lack of root cause understanding means that future occurrences of the problem cannot be avoided.

The implicit assumptions are that problems will not simultaneously affect large numbers of machines, and that problems affecting a machine will be relatively infrequent. These may not be valid assumptions if DDOS attacks are made on the infrastructure. Furthermore, this approach requires that all failures be detectable by the probes.

LiveOps provides a critical missing component of the software configuration management cycle (shown in Figure 1) by detecting all system changes, verifying approved requests, and alerting upon unknown modifications. Closing this loop is becoming increasingly important in order to identify unwanted changes, malware for example, and to fulfill auditing requirements [27, 29].

Managing servers and desktops through a management service has several advantages over individual companies or departments maintaining a local management infrastructure. These are:

1. Current and historic information about the managed systems is available and accessible despite local outages and system failures, and that it provides a quick and easy method of sharing detailed information with support professionals and other problem domain experts without local network or system access.
2. The service enables correlation of system activities across large numbers of managed hosts to identify known good and known bad values.
3. The service enables application and OS developers to receive details on how their software is being configured and used which can help identify problems, and drive future product enhancements.
4. The service provides a centralized, globally available, and integrated system for application developers, administrators, and support professionals to add knowledge on specific PS, regarding optimal values, and new analysis rules to identify and resolve problems so future instances will not require detailed investigation.
5. The service becomes a management platform for new reports so problem analysis techniques can be added to the service's back-end servers without requiring software or configuration updates to the agents running on managed systems.

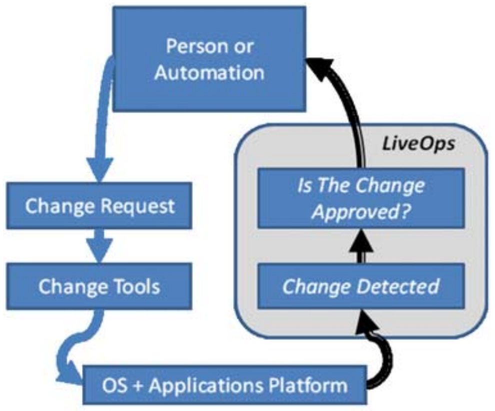


Figure 1: Typical Change Management Process showing how LiveOps closes the loop.

Architecture

In this section, we present an overview of the LiveOps system architecture. Figure 2 describes the system architecture and data flow of:

1. Our low-level server or desktop agent that logs all registry, file, binary load, and process creation interactions, compresses the trace events into log files, and uploads them for analysis [33]
2. The LiveOps back-end service that processes and archives the uploaded log files
3. An extensible web interface that supports retrieving reports, programmatic data access, and integrating with existing notification mechanisms.

Our implementation does not require any changes to the core operating system or any applications specific changes or configuration

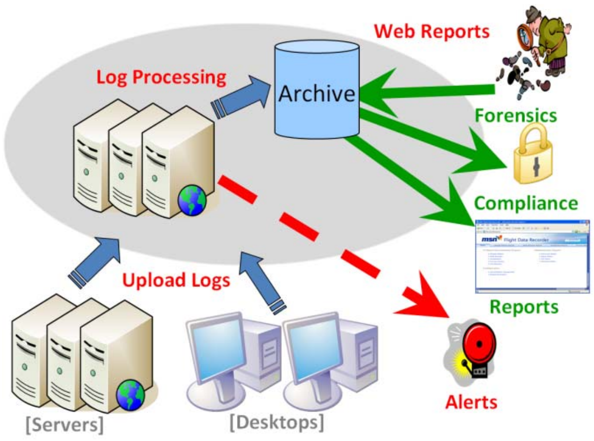


Figure 2: LiveOps System Architecture and data flow.

Figure 3 describes the processing flow of the back-end LiveOps servers, detailing the analysis performed by the daemons on newly received logs. Anomaly detection rules, such as checking for policy violations or correlating an observed change with a planned change work order, are written in standard C/C++/C# code and compiled into a dynamically loadable Query Module (QM). At startup the daemons load the QMs, passing each QM a reference to the newly uploaded logs.

Internally, a QM may integrate with external IT data sources, such as change management databases (CMDB) or change policy definitions. The QM then detects anomalies such as identifying patterns within the logs, correlating log activities with CMDB work items, or using change policy definitions to identify policy violations. All QMs write alerts to a common API that stores them in the Alert Database. An Alert Notification daemon reads the alerts and sends notifications through Instant Messenger, email, and RSS feeds. A QM may also maintain an internal database, such as the baseline database used to store a history of application interaction with PS. The baseline QM uses its database to detect deviations and log them as alerts. Once log files have been processed by all QMs they are moved into a central archive. Administrators can create ad-hoc queries against all archived data using the web service that exposes the LiveOps query API. This web service is used by the LiveOps web server to generate HTML reports, and can also be used by other products to integrate with LiveOps data.

In addition to providing access to the information, LiveOps also provides contextual information about each alert and the settings it describes. Three stages of annotations:

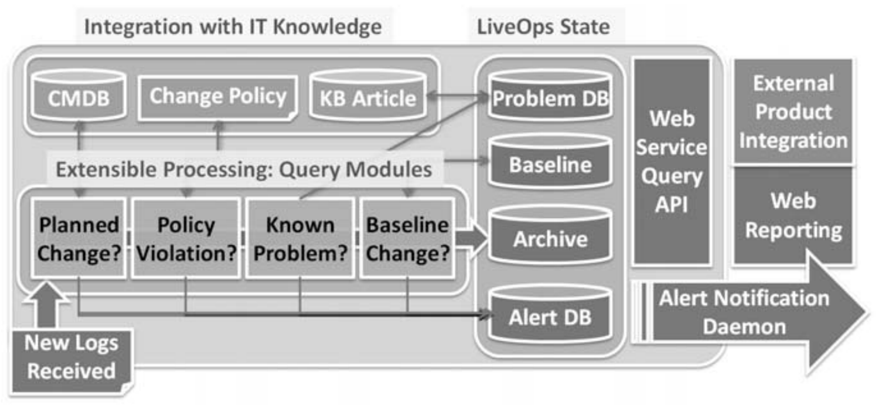


Figure 3: LiveOps analysis framework.

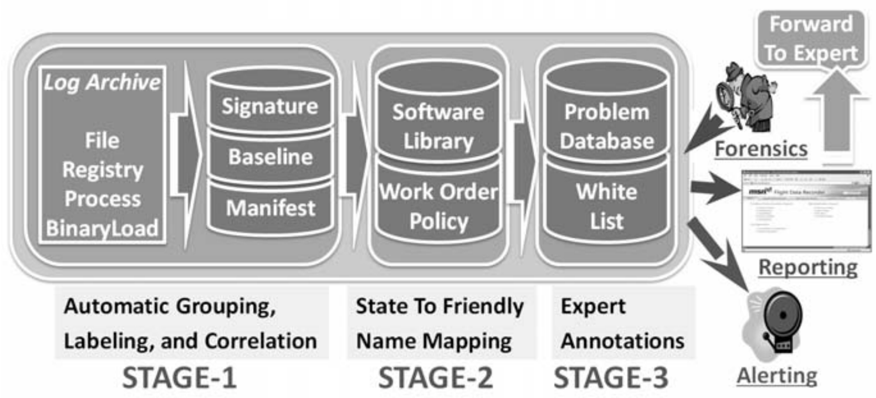


Figure 4: LiveOps annotation of information.

1. ***Automatic***
2. ***Pre-defined context mapping***
3. ***Expert***, as described in Figure 4, are applied to the alerts and individual file and settings as they are read from the web interface

For example, let's consider the annotations added for an alert that describes a new application installation on a monitored machine. The first level provides statistical information such as how many other machines have this application installed, and the most common versions of files and values of settings associated with it. The second level of annotation compares the binary hash of the installed files with a database that indexes company name, application name, and version information based on binary hashes. The final level of annotation adds any comments or rating information provided by other users of the system.

If this installation was known to be spyware or malware, the final annotations could include support articles that describe how to deal with the installation. The quality of this information grows as more machines report their data to the service, and as more users provide comments and ratings on the alerts and individual file and settings entries. It also provides valuable insight into the application vendors about how consumers configure and use their products, along with insight into the real-world problems their products have.

LiveOps Scenarios

LiveOps was created within the management philosophy that administrators understand the PS of a running system, all changes made to the PS, and the scope of PS interactions for correctly running processes. This is required in all administrative management scenarios which are described in the following subsections. Each section contains a description of the LiveOps reports, how they are generated, and a summary of results from running LiveOps on production servers at MSN.

The section on "Understanding Changes" describes how LiveOps improves the change management processes used by most IT organizations by showing how it can be used to:

1. Verify that changes are correctly made and to discover unauthorized changes
2. Identify unapproved processes
3. Discover the impact and frequency of changes impacting critical systems and applications
4. Identify sensitive information that is being copied to removable devices and network locations
5. Analyze and account for the PS that are often left behind by software installation and removal programs

The section on "Managing Abnormal System Activity" shows how LiveOps can significantly reduce troubleshooting time by:

1. Detecting known problems
2. Providing contextual information about what PS processes and users were accessing
3. Identifying processes that have become stale from changes to PS

"Enforcing Best Practices" describes how LiveOps can be used to improve the best practices followed by administrators by identifying:

1. When users are logging in to systems, potentially unnecessarily
2. Daemons running under local user or domain user accounts instead of the system account

Scenario: Understanding Changes

The typical change management process followed by many IT organizations is:

1. Identify change to be made and create a work order
2. Verify correctness of change in test environment with test workloads
3. Deploy the change to subset of servers and monitor for problems caused by real-world workloads
4. Broadly deploy the change and close the work order

The three main challenges with implementing this process are:

1. Defining the specific changes expected
2. Auditing the changes as they are made to the systems
3. Identifying the impact of the change to understand the applications that must be tested and verified

Without LiveOps 'Change Management' reports defined in this section, scheduled changes are often not understood and therefore their impact is not well known. For example, work orders may define a change as applying a SQL Server Service Pack 1 update, rather than listing the specific files and settings being modified. Consequently, understanding if the change has been correctly and completely applied to all required servers is often based on unreliable and incomplete methodologies, such as:

1. Sampling each affected server for one of the known files in the change
2. Relying on the administrator notes about what changes were made and when
3. By examining logs that may not fully describe the change

Often times the software installation for patches and applications does not maintain an accurate record of the PS being created or changed on the system, and consequently their uninstall programs can leave PS behind, or even corrupt the remaining applications on the system. In a later section, we present a case study of software installation and removal that demonstrates the significance of this problem.

Furthermore, current auditing techniques can only detect changes where existing changes are recorded by the affected applications and system logs, or those which are manually reported by administrators. For example, if an administrator makes multiple configuration changes in an attempt to troubleshoot a problem and forgets to roll back one of them, this change will not be recorded and may cause a future problem on the server.

In addition to auditing changes as part of the change management process, there is a growing need to audit all interactions on systems. Several regulatory bodies require maintaining strict audit logs of changes made to systems [27, 29]. Also, new consumer protection laws enforced in Japan and being drafted in other countries, now make businesses liable for Personally Identifiable Information (PII) that is leaked or stolen from their companies [13, 14]. In addition, the growing trend to outsource business functions that relate to Intellectual Property (IP), such as the source code used to develop software, provides a need for auditing IP and PII for when it is accessed, modified, or copied.

To improve the change management process and facilitate regulatory auditing, LiveOps provides reports that describe critical changes; document unauthorized applications and change impact analysis; and list sensitive data copied to network and removable devices. The remainder of this section describes these reports in more detail and summarizes our results from generating them in MSN datacenters.

Report: Critical Changes

LiveOps defines critical changes as:

1. Unexpected program execution
2. Modifications made to PS, used by the operating system and line of business (LOB) applications

The remainder of this section defines the alerts used to generate critical change reports, and describes the distribution of alerts detected for a one-month sample of LiveOps data.

Datacenter reviewers of the report can manually associate the processes needing approval with work order systems. LiveOps can be used to verify configuration changes, potentially in an automated way, by correlating planned changes with the process name, user name, and time of the changes made. LiveOps marks alerts for processes requiring approval that run during 'Lockdown' periods by integrating with the lockdown schedule maintained externally to LiveOps by operations managers.

Changes to PS used by the OS and LOB applications must be authorized and controlled to avoid reliability and availability problems. Changes made to management applications on these servers are also important because they control the insight administrators have into the performance, reliability, and change management behavior of the server. Incorrect changes to management applications may reduce or eliminate visibility, and may expose the system to attackers.

While it is important to understand the critical changes affecting systems, not all writes to PS are interesting to administrators. To identify the critical changes from all writes to PS the following nine prioritized classifications are used to label each entry:

- **Problem**: Indicates a known problem – results from the existence or removal of this PS.
- **Install**: PS added as part of an installation or upgrade.
- **Setting**: Changes made to configuration PS.
- **Content**: Web pages, images, and user data.
- **Management Change**: Installation, patching, or configuration changes made to the management applications running on the system.
- **Unauthorized**: Installation of prohibited applications, or configuration changes to prohibited values.
- **User Activity**: PS modified as a result of users logging in or running window applications.
- **Noise**: Temporary or cached PS.
- **Unknown**: Unclassified PS.

The classification process involves associating each match of a substring contained in a classification rule to the PS name contained in each change event. Matches to classification substrings with higher priority take precedence over those with lower priority. For

	State Grouping		Process Grouping of State Changes			
State Classification	All	Distinct	Daily Instances	Daily Distinct	Monthly Distinct	Average Per Machine Daily Distinct
Problem	0	0	0	0	0	0
Install	104,149	16,947	810	155	69	4
Configuration	176,300	3,340	399	86	22	3
Content	16,261,721	1,593,100	9,513	50	1	3
Management Change	57,145	864	234	79	11	2
Unauthorized	4,206	634	14	9	3	2
User Activity	1,221	189	96	33	4	3
Noise	104,109	1,727	909	66	14	2
Unknown	39,715	2,613	534	60	13	3
TOTAL	16,748,566	1,619,414	12,509	538	137	22

Table 1: Critical changes for one month of production server logs across 34 machines showing individual changes and changes grouped by process.

example, a PS name matching both 'User Activity' and 'Install' classification substrings will be labeled as 'Install.' From examining 28 days of change activity from 34 systems, we identified 1 to 20 substring rules for each classification. Administrators can update the classification rules as needed, however, Table 1 shows that the initial rules cover all except 0.2% (39 k/16.7 M) of the PS observed.

Table 1 shows a summary of the changes observed across one month of traces from 34 production servers. It shows that 16.7 M individual changes were made to 1.6 M distinct PS entries, meaning that on average each PS was changed 10 times. Considering an overall average of 300 k PS entries per machine, this means that only 16% (1.6 M distinct state grouped changes / (300 k PS * 34 machines)) of the PS was modified during this period.

Although 1.6 M changes are too many for an administrator to review we can significantly reduce the items to review if we assume that each process instance is updating a set of related PS. For example, an installation program will add thousands of Registry entries and hundreds of files during the installation of a single application.

Table 1 shows that there is an O(10^2) reduction in changes if daily change process instances are considered instead of distinct state. This can be further reduced to O(10^3) by considering only changes made by daily distinct processes, however, comparing monthly distinct with daily distinct we reduce the number of changes to consider by only half. For example, in MSN it is common for the same patch installer, KB-123.exe, to be run on all 34 systems, which can be presented as one distinct kb-123.exe entry, providing the administrator with the ability to 'drill in' to see the affected machines and individual PS changes. If we consider the average daily distinct changes for each machine, rather than the overall distinct changes, we find that there are only O(10^1).

The LiveOps lockdown reports show configuration changes such as installations, patching, changes to LOB, OS, or management applications during periods when changes are prohibited on systems. Lockdown reports from five properties were analyzed for nine lockdown ranges over a seven month period. Figure 5 shows that all properties had lockdown violations during at least one period; two properties had violations in eight of nine lockdown periods. Violations ranged from minor issues such as running diagnostics on systems, to more severe changes like installing service packs and new applications. We expect identifying and attributing lockdown violations to specific administrators will be a strong deterrent.

Report: Unauthorized Applications

Only approved processes should be running on datacenter systems. Similar to systems such as TripWire [16], LiveOps uses a predetermined list of approved and unapproved applications to identify unauthorized processes on servers. The list contains the following details for each of the approximately 200 distinct process names observed on production servers, and 1300 distinct process names observed across all systems including desktop and home machines:

- **Approved** – Marks the process as approved (or not) for running on production systems, or requires a work order before running it.
- **Type** – Indicates if this process is a command line tool, daemon, or interactive window application. Reports will use this classification to determine if daemons are not running under local system credentials, if window applications are being remotely launched, or if local logins are made when administrative activities could be done remotely.
- **Category** – Indicates the intended use of the process
  - ***LOB***: Created specifically for a business need, such as the infrastructure for a web site.
  - ***OS***: Processes that run as part of the OS as required components, such as lsass.exe.
  - ***Desktop***: A corporate desktop application like an email client. Applications that are used in server environments but designed for home or desktop use may not meet the stringent security requirements of a server environment.
  - ***Dev***: A developer tool such as a compiler.
  - ***Home***: For home entertainment, such as a game.
  - ***Mgmt***: Management products and tools used to manage systems like agents or event log tools that ship with the OS. Instances of diagnostic tool processes can indicate server problems may exist, and instances of configuration tools indicate that prior authorization should be granted.
- **Function** – Describes what this process can potentially do on the system. This classification is used by reports that highlight users running

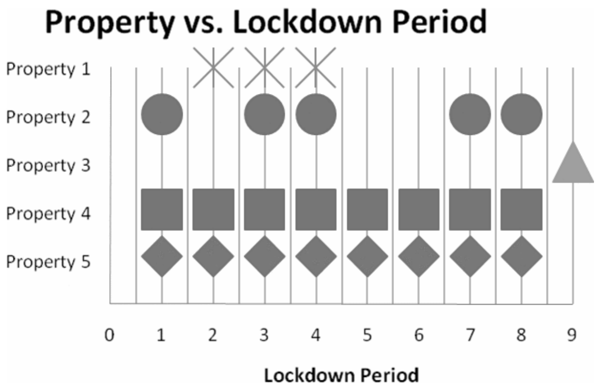


Figure 5: A marker for each property with at least one violation during a Lockdown period.

processes that could make changes to the system's PS.
  - ○ ***Setup***: Can be used to install or remove applications.
  - ○ ***Patch***: Updates existing applications.
  - ○ ***Config***: Can be used to modify the configuration of the system.
  - ○ ***Diag***: Used for retrieving diagnostic information from the system. Examples are ping and traceroute.
  - ○ ***Viewer***: Displays read-only data, from files or other sources. Tools like 'winver.exe' fall into this category.
  - ○ ***WriteData***: Indicates the process can create data such as log files like the LiveOps agent, or can modify existing data like notepad.exe.
  - ○ "***Fun***": have no business value, and used only for entertainment. Examples are games and DVD playing applications.
  - ○ ***System***: Required for operation of the system. Some examples are lsass.exe and csrss.exe.
  - ○ ***Runtime***: The process hosts other applications. Examples are scripting environments such as Perl, or surrogates like dllhost.exe, svchost.exe and java.exe.
  - ○ ***Malware***: Malicious or unwanted software that should never be run.
- **Product** – Name of the product.
- **Manufacturer** – Details about the Manufacturer.
- **Description** – Description of the product describing its key functionality.

Processes seen for the first time are by default not approved, and therefore show up in the report for classification by administrators. Additionally, processes that are known to be used for diagnostics or implementing configuration changes to the system are alerted on, and marked as needing approval.

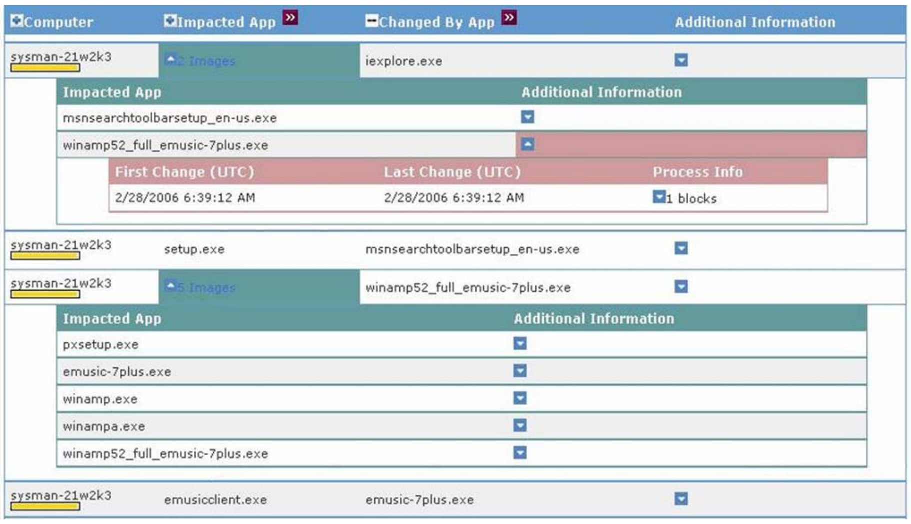
Analyzing a sample report from 126 production servers over a one month period we found 37 systems (29%) ran an overall total of 18 distinct unauthorized processes. The breakdown of these is: three were associated with automatic updates to a runtime environment, seven were desktop applications and data manipulation tools, and eight could not be identified by administrators or security experts in the organization. 76 systems (60%) ran 17 distinct processes that required approval because they can be used for diagnosis or configuration changes.

Report: Change Impact Analysis

Each time a new binary is used on a system this alert shows the application that installed the binary file, and the user context that installed it. LiveOps catches binary installations regardless of whether the installing application participates with RPM, MSI, or PackageAdd. Figure 6 shows the daily alerts generated by iexplorer.exe (web browser) which was used twice to download and install applications. The first time it downloaded and installed msnsearchtoolbarsetup_en-us.exe (MSN Search Toolbar) and the second time it downloaded and installed winamp52_full_emusic-7plus.exe (Winamp Media Player). It shows that Winamp created several binaries on the system, including emusic-7plus.exe (EMusic), which later installed even more binaries.

This report is useful to administrators that may not have expected EMusic to be installed, and might at a later time wonder where, when, and how the EMusic binaries got on their machine. Figure 7 presents a 'drill in' of the detailed process tree at the time of the installations. It clearly shows that the web browser launched the Winamp installation, which in turn launched the EMusic installation.

Identifying the impact that changing a PS entry has on a system's running applications is useful for test verification planning, and for troubleshooting. Knowing the specific PS affected by a change and the



Computer	Impacted App	Changed By App	Additional Information
sysman-21w2k3	2 Images	iexplore.exe	

Impacted App	Additional Information
msnsearchtoolbarsetup_en-us.exe	
winamp52_full_emusic-7plus.exe	

First Change (UTC)	Last Change (UTC)	Process Info
2/28/2006 6:39:12 AM	2/28/2006 6:39:12 AM	1 blocks

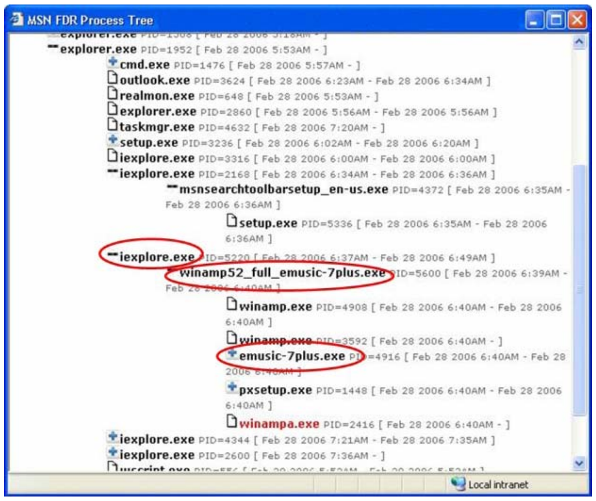
Computer	Impacted App	Changed By App	Additional Information
sysman-21w2k3	setup.exe	msnsearchtoolbarsetup_en-us.exe	
sysman-21w2k3	5 Images	winamp52_full_emusic-7plus.exe	

Impacted App	Additional Information
pxsetup.exe	
emusic-7plus.exe	
winamp.exe	
winampa.exe	
winamp52_full_emusic-7plus.exe	

Computer	Impacted App	Changed By App	Additional Information
sysman-21w2k3	emusicclient.exe	emusic-7plus.exe	

Figure 6: LiveOps detecting a web browser (iexplorer.exe) downloading and installing two new applications.

affected applications enables administrators to prioritize testing the specific features of the applications controlled by the updated PS. Administrators and support professionals can also significantly reduce the scope of potential root cause PS by knowing which recently modified PS is used by the broken application.



```
MSN FDR Process Tree
    explorer.exe PID=1952 [ Feb 28 2006 5:53AM - ]
        cmd.exe PID=1476 [ Feb 28 2006 5:57AM - ]
        outlook.exe PID=3624 [ Feb 28 2006 6:23AM - Feb 28 2006 6:34AM ]
        realmon.exe PID=648 [ Feb 28 2006 5:53AM - ]
        explorer.exe PID=2860 [ Feb 28 2006 5:56AM - Feb 28 2006 5:56AM ]
        taskmgr.exe PID=4632 [ Feb 28 2006 7:20AM - ]
        setup.exe PID=3236 [ Feb 28 2006 6:02AM - Feb 28 2006 6:20AM ]
        iexplore.exe PID=3316 [ Feb 28 2006 6:00AM - Feb 28 2006 6:00AM ]
        iexplore.exe PID=2168 [ Feb 28 2006 6:34AM - Feb 28 2006 6:36AM ]
            msnsearchtoolbarsetup_en-us.exe PID=4372 [ Feb 28 2006 6:35AM -
            Feb 28 2006 6:36AM ]
                setup.exe PID=5336 [ Feb 28 2006 6:35AM - Feb 28 2006
                6:36AM ]
        iexplore.exe PID=5220 [ Feb 28 2006 6:37AM - Feb 28 2006 6:49AM ]
            winamp52_full_emusic-7plus.exe PID=5600 [ Feb 28 2006 6:39AM -
            Feb 28 2006 6:40AM ]
                winamp.exe PID=4908 [ Feb 28 2006 6:40AM - Feb 28 2006
                6:40AM ]
                winamp.exe PID=3592 [ Feb 28 2006 6:40AM - ]
                emusic-7plus.exe PID=4916 [ Feb 28 2006 6:40AM - Feb 28
                2006 6:40AM ]
                pxsetup.exe PID=1448 [ Feb 28 2006 6:40AM - Feb 28 2006
                6:40AM ]
                winampa.exe PID=2416 [ Feb 28 2006 6:40AM - ]
        iexplore.exe PID=4344 [ Feb 28 2006 7:21AM - Feb 28 2006 7:35AM ]
        iexplore.exe PID=2600 [ Feb 28 2006 7:36AM - ]
Local intranet
```

Figure 7: Drill in of the process tree at the time of the installation.

LiveOps identifies the dependencies of each process running on a system by tracking the PS read and modified by each observed process. This list of dependencies, called a manifest, can then be intersected with each change on the system to identify the impacted processes [8, 30]. If the contents of patches or applications are known before they are installed, their impact on existing applications can be predicted by intersecting their contents with the manifests for each observed process. The Application Compatibility Toolkit version 5.0 [19] contains a feature called the Update Impact Analyzer, which uses portions of the LiveOps technology to provide this functionality, enabling administrators to prioritize testing and migration of applications and patches.

Table 2 presents a summary of daily changes affecting the five categories of processes observed across 34 production servers over a one month period. Surprisingly we see that each day there are several changes made that impact LOB and OS processes, and that each machine on average only receives about half (90 of 163) of the distinct global installations impacting OS applications. This means that the 34 machines are not all receiving the same set of installation changes, whereas the two install changes affecting the desktop category of processes is uniformly applied to all systems. Overall, we see a trend of uniformly applied PS changes to desktop, developer, and home applications. However, less than half of the LOB, management application configuration, and OS configuration changes are consistently applied to all machines. Although we expect systems from all five MSN sites to run the same management processes, and configure them the same, these results show that there are in fact differences.

Report: Sensitive Data

Protecting a company's intellectual property (IP) is of paramount importance because IP is often a critical asset responsible for its competitive advantages. Examples are source code, strategic business plans, and product specifications. Similarly, many customers provide personally identifiable information (PII) to corporations as a part of transacting business. Examples include credit card and social security numbers, contact information, and medical records. Personal Information Protection Laws enacted in 2003 in Japan [13, 14] make companies liable for up to 300,000 yen or six months jail time, for each person affected by PII leaks, which, coupled with an increase in malware and hacking [26] means that corporations need to ensure data is protected.

Further demonstrating the significance of this problem, the CSI FBI report [11] asserts that 59% of corporate security abuse is caused by employees [15]. All anonymous security experts we surveyed on this point thought the number may in fact be closer to 80-90%. Protecting IP and PII from internal employees is challenging because many of them will have legitimate business needs to access the data, and administrators will have implicit access to the data by virtue of managing and backing up information on the server. This means network isolation, firewalls, and

		Categories of Distinct Impacted Processes				
Average Daily PS Change Type		Desktop	Developer	LOB	Management	OS
Across all Machines:	Install	2	182	56	168	25
Per Machine:	Install	2	124	23	95	5
Across all Machines:	Configuration	0	2	4	27	9
Per Machine:	Configuration	0	2	2	10	1
Across all Machines:	Content	0	8	37,063	46	9
Per Machine:	Content	0	8	18,112	22	6
Across all Machines:	Management	0	0	1	59	5
Per Machine:	Management	0	0	1	59	6

Table 2: Impact analysis showing the average daily global and per machine changes impacting each of the six categories of processes observed running across the 34 production servers during a one month period.

	User Setup	Script	Auto Update	Self Update	Developer
Home	7%	7%	51%	35%	0%
Desktop	5%	8%	58%	22%	7%
Lab	2%	5%	61%	32%	0%
Server	1%	17%	38%	44%	0%

Table 3: Distribution of installations by program type for all observed installations.

access control lists (ACLs) will not be effective at solving this problem.

LiveOps addresses this issue by providing an audit report on all files copied to remote network shares and removable devices, for each machine running its agent. The files being copied are then categorized according to their content using a set of configurable classification filters. An alert is then generated for each process instance that contains the user context of the process, the process name, and a list of the sensitive files being made remote.

From examining 383 machine days of logs from 35 datacenter servers and corporate desktops we found: 36 instances of internal documents, six instances of source code, and three instances of applications being copied to network and removable devices. The reports also identified 100's of Corporate IT tools copying logs off of these machines.

Case Study: Tracking Software Ownership

Intuitively, we may think that managing the PS of a system should be easy because executable files and configurations are infrequently created or are modified by well-known software installers. We analyze below our traces to find that installations actually occur quite frequently. Additionally, while most software installers provide manifests, listing the PS entries owned by the installed application to facilitate its clean un-installation, these manifests are often incomplete or incorrect [12]. In the very next section we analyze PS entries across 70 machines and find that many entries cannot be accounted for via the machines' static manifests. To better understand the origins of these orphaned PS entries, we describe point experiments with installing and uninstalling three popular applications.

How Often Is Software Installed?

To identify a software installation we examined our PS traces to identify the creation or modification of files that were later loaded by a process as an executable file. We found that on average 20% of all machine days had at least one installation. However this varied significantly across each environment. 15% of Home and Lab machine days and 30% of desktop machine days had at least one install. Server environments had a wide variance in the frequency of software installations, ranging from 7%-80% of machine days having at least one install. This reflects the variation in change management policy for each Internet service.

While we might have thought that centralized administration of corporate desktops, or Windows auto-update service might cause synchronized updates, this does not appear to be the case. Overall, we observed that most software installations, even in the server environment, occur in an unpredictable manner.

Table 3 describes the distribution of observed installations across install types. We see that processes which update themselves (Self Update), predominantly anti-virus applications, account for a significant portion of installs. Also, enterprise software distribution applications and Windows auto-update account for the majority of software installs in most environments. As we expect, servers have a large portion of scripted installs from administrators manually rolling out upgrades and in-house applications. Installations caused by users running install programs are infrequent. It is interesting to see that our analysis was able to distinguish binary files created and used on developer machines as 'installed' by the developer tools by tracking the processes that created each binary.

Static Software Ownership Manifests

Unfortunately, a statically declared manifest is not always complete. Today's manifests are not always correctly specified, nor do they account for PS created post-installation, such as user preference settings, log files, etc. This means that during software upgrades or removal, entries can become orphaned on the machine. Furthermore, installation or removal of software can fail, or be interrupted which often leaves registry entries and files in an inconsistent PS. Over time the orphaned files and registry entries accumulate on a machine causing a buildup of unused entries that can lead to system problems.[1] Because of this phenomenon, common advice is to reinstall your OS and all applications occasionally to return the system to a known state.

To quantify the significance of this problem, we wrote a tool to extract PS ownership manifests from within the Windows OS. The tool identifies the components installed on the system by analyzing the OS installation configuration files, enumerating the list of programs that have registered with the Windows 'Add/Remove programs' component, the Windows Installer database (WI), the OS configuration for launching applications when a file of a given extension is run, and manifests of patch contents as described in the Microsoft Security Baseline Analyzer

[1]Examples can be found at http://support.microsoft.com/ via the article IDs: 898582, 816598, 239291, 810932, 181008.

		Manifest	Implicit	Data	Temp	Unknown
File	Desktop	18.5%	21.0%	20.6%	8.3%	31.6%
	Server	4.7%	52.6%	2.4%	3.4%	36.9%
	Lab	13.2%	5.8%	9.5%	1.4%	70.1%
Registry	Desktop	28.2%	32.2%	N/A	N/A	39.6%
	Server	10.5%	36.4%	N/A	N/A	53.1%
	Lab	30.3%	31.7%	N/A	N/A	38.0%

Table 4: Average file and registry entries that are specified in manifests, implicitly in manifests, user data, or temp entries. We do not have heuristics to recognize data and temporary registry entries.

tool [20]. We then enumerate all files and registry entries on a machine and compare them with these manifests to identify unaccounted for entries. We associate entries that are descendents of entries that are referenced in the manifests as implicitly associated with the manifest as well. Finally, we filter the remaining list to exclude well known user data files by their extension, and remove entries from well known temporary folders. We define the remaining subset as leaked entries on the system. Table 4 contains the results of running this tool across eight desktops, 20 Server, and 42 lab machines. It shows that 31-70% of files and 38-53% of registry entries could not be accounted for.

To further understand the prevalence of software leaks, we measured the leakage of three popular commercial applications. By running our data collector while installing the application, using it for a short period, and then uninstalling the application; we were able to measure the net increase of file and registry entries on the machine.

The first application was the game 'Doom3,' which left nine files and 418 registry entries. The second was the common corporate desktop application suite Microsoft Office, which left no files, but 1490 registry entries. Additionally, it left 129 registry entries for each user that logged into the system and used the program while it was installed. The third example was the enterprise database application Microsoft SQL Server Yukon edition, which leaked 57 files and six registry entries.

Scenario: Managing Abnormal System Activity

Administrators are often required to reactively manage systems that do not operate as expected because of:

1. Hardware problems
2. PS problems such as incorrect configuration or mismatched binary versions
3. Programming logic related issues such as memory corruptions and crashes.

The process for debugging hardware and programming logic related problems is relatively straightforward because there are many tools to aid in the analysis and the correct behavior is known.

For example, solutions often exist for hardware diagnostics such as correctness tests performed in software [41], and often devices like hard-drives are capable of reporting when they are about to fail [1, 17]. Similarly many tools exist for software debugging such as code analysis tools that are run during the software development process [22], specialized software debuggers [10, 18] and crash analysis software programs for diagnosing application problems when they happen [3, 39].

Configuration related problems on the other hand, are very difficult to debug because tools do not typically exist to analyze the 200,000 settings [36], and 100,000 files [7, 32] that exist on a typical system. Furthermore, the knowledge of which files and settings the application under investigation depends on is typically manually learned by operators as they gain experience with applications. The task of debugging application configuration is further complicated by the complexity caused by frequent updates to configuration, and the customization of system environmental variables and settings.

Several strategies have been proposed for minimizing the potential for configuration problems at the expense of either application functionality or availability. One approach is to statically link libraries with executables to reduce the overall number of executables on the system and thereby minimize the potential for mismatched binary versions. However, doing this makes it more difficult to patch libraries when security vulnerabilities are discovered because all applications using the library need to be rebuilt and reinstalled.

To improve the troubleshooting and forensic investigation of problems, LiveOps provides reports that describe stale PS which identifies applications that are using stale versions of PS compared to the version that exists on disk, the ability to generate adhoc forensic reports for specific time ranges that can be filtered by processes, users, and files and setting interactions, and instances of known problems. The remainder of this section describes these reports in more detail and summarizes our results from generating them in MSN datacenters.

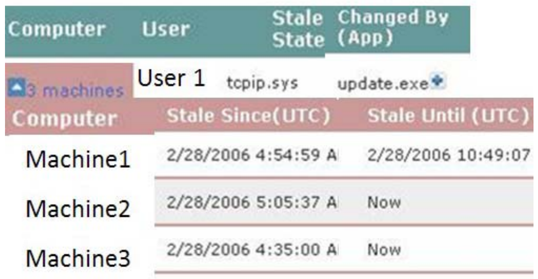
Report: Stale Processes

When applications read PS into memory there is the potential for the persisted version of the PS to be updated by another process, causing it to use an old 'stale' copy of the PS. An example of this is applying a security patch to a critical file such as the tcpip.sys network driver in Windows, which updates the file on disk.

If the system is not rebooted after the patch has been applied, the old copy of the file is used in memory and therefore the system is still vulnerable to the security exploit. Similarly if an application reads its configuration settings at startup and does not monitor for changes that happen while it is running, any new changes made while the application is running will not take effect. During installations and upgrades it may be expected that some PS will becomes stale while upgrades are made, however, the length of time applications are stale should be small. We define any process that has not re-read updated PS within five minutes as a stale process.

LiveOps has the unique ability to detect when binary, file, or registry PS read by a process has become stale due to external changes. A report on stale processes is created by correlating all writes to PS with the PS loaded by currently running processes. The LiveOps report contains an entry for each stale process, and enables the administrator to 'drill in' to each entry to examine the individual stale PS, showing the time it was modified and the process and user context that made the change.

Figure 8 presents a section of a LiveOps stale report that shows a specific user running Windows Update (update.exe) at 2/28/2006 on three production servers. It shows the exact time that the tcpip.sys driver was modified on each of the machines, and the time when the new version of the driver was reloaded into memory. We can see that the systems were updated between 4:35 am and 5:05 am, but only the first one reloaded the driver six hours later at 10:49 am. This means that the other two machines were still exposed to the security vulnerability caused by the old tcpip.sys binary more than 24 hours later when this report was viewed. Without this LiveOps report the server could have been vulnerable for several days because reboots of servers happen infrequently.



Computer	User	Stale State	Changed By (App)
3 machines	User 1	tcpip.sys	update.exe

Computer	Stale Since(UTC)	Stale Until (UTC)
Machine1	2/28/2006 4:54:59 A	2/28/2006 10:49:07
Machine2	2/28/2006 5:05:37 A	Now
Machine3	2/28/2006 4:35:00 A	Now

Figure 8: LiveOps detected that the updated tcpip.sys binary was not reloaded on the bottom two machines, therefore they are still vulnerable.

We examined several examples of stale processes found from a sample set of 34 machines over a one month period. Several stale OS, Management, and LOB processes were stale for more than 20 hours from software binary updates and management configuration changes. In cases where administrators are troubleshooting problems with these applications, knowing that they are stale could significantly reduce troubleshooting time, because examination of the newly changed state could mislead administrators to believe that the new values were being used.

The observed stale state ranged from:

1. Cached domain server information which could affect the performance of authentication operations
2. Management configuration which could affect the availability of monitoring information or the ability of administrators to connect and diagnose the system
3. Updated environment variables that could affect LOB applications that required specific path settings to locate needed binaries and data files
4. Web site content and configuration which may be expected to have an immediate affect once modified on the system, or could break other websites due to partially available new content

Query Interface: Forensics and Troubleshooting

The Forensic query interface is useful when administrators want to know who, when or how changes are being made [23, 37, 38]. For example, if malware is detected on the system by anti-malware tools, the forensics interface can be used to identify when it arrived and which user context and process was used to create the malicious files to identify the root cause of the issue and prevent future occurrences.

LiveOps has successfully identified performance problems in several applications that were unnecessarily reading registry entries hundreds of times per second. One instance was found in an LOB application running on a web server where the registry entry \HKLM\SOFTWARE\Microsoft\Cryptography\Defaults\Provider\Microsoft Strong Cryptographic Provider, and all of its values were read 240 times per second. Another case was found in a commercial management product agent that was deployed on a production server, where it was continuously reading all registry entries and values under the \HKLM\System\CurrentControlSet \Services\ key to detect changes in the parameters of daemons installed on the system. The developers of both products were notified and told that they could make their applications vastly more efficient by using the RegNotifyChangeKeyValue function to register for registry changes rather than polling for them.

LiveOps can also be used in an ad-hoc manner when troubleshooting to identify if access violations reading files or registry entries are causing applications to fail, or even for investigating intrusions from hackers where the investigator wants to know which processes were used during the attack, and potentially which sensitive data files were compromised.

Report: Detecting Known Problems

The root cause of configuration problems identified by administrators, support professionals or technically adept end-users, can be used to define assertions about the correctness of configuration on any system. The LiveOps known problem report is generated by applying codified configuration assertions, called rules, to the registry entry names and values collected from each managed system. The report contains a row for each machine and configuration entry that matches a rule. Administrators can review the rule description that indicates why the entry is suspicious, and view the data contained in the problematic entry.

While several registry scanning products exist, LiveOps is different in that it does not need to run on each managed system to scan the entries because it can instead scan the logs that are centrally uploaded. LiveOps results are also more relevant because the logs contain the registry entries and values that are currently being used by the applications so there are less false positives from entries that are not actually used. Also, user perceived application problems can be correlated with PS usage because the logs contain the times an application used the problematic registry entry. The logs will also contain the exact time, process, and user context that changed the registry entry to the problematic value.

The LiveOps known problem rule set contains approximately 100 assertions created from reviewing the Strider troubleshooter cases [36], PC fragility cases [9], and from critical registry entries identified by MSN administrators such as the setting that controls the operating system paging file. This system can be further enhanced to take advantage of PeerPressure [35] comparisons to identify problematic configurations based on the configuration of similar machines.

Reviewing the known problem report for a sample one month period from 350 desktop, home, and server machines revealed several issues. Most interesting were the 35 servers that had their page file setting configured to null, which caused the system to crash when physical memory was exhausted. Reviewing the problem incidents from the time when these changes were made revealed that several of these servers had in fact crashed from this incident. The reports showed that it was the svchost.exe process configured to support remote registry calls that was used to make the problematic changes, probably from an incorrectly written management script.

Another interesting problem was found on a laptop where the rule for the GSM audio codec configuration detected a missing value. This codec is part of the default Windows installation for playing .wav files. Attempting to play a .wav file on the machine confirmed that it was an actual problem. Resetting the value resolved the issue.

Several of the remaining detected problems related to rules regarding user preference changes that end users may unintentionally make and not know how to undo. For these entries, LiveOps marks the alerts as warnings rather than errors, and if a user from a managed system complains about an application problem these entries helps quickly resolve related issues.

Scenario: Enforcing Best Practices

Datacenter managers typically define best practices for managing systems to reduce the potential for security vulnerabilities, and to minimize the potential for introducing reliability and availability issues. However, it is a never-ending challenge to audit whether these practices are being followed. As an organization grows the number of its datacenters around the world, it becomes more difficult to educate the administrators on the best practices. This also means that an increased number of administrators will be interacting with the systems, thereby increasing the potential for best practice violations. With more people interacting with systems it is hard to identify which administrators may have inadvertently introduced a problem, and therefore makes it difficult to re-educate them on the best practices to avoid problem reoccurrences.

LiveOps reports have been created to analyze best practices for minimizing exposure to potential security vulnerabilities, and for minimizing potential system instabilities. The initial best practices reports are: Login report which details the users that have been logging in to each server; and Daemons running with local or domain credentials.

Report: Logins

Avoiding unnecessary logins to production servers reduces the potential for hackers to obtain credentials that can be used to connect to other systems in a network. Logging in to a server causes a primary user context to be created, which can be used to connect to remote systems one hop away. If, instead tools are run remotely on the system, a secondary security context that is only valid on the remote machine is used. Furthermore, a best practice for avoiding the potential for problems is to minimize the amount of changes and interactions on managed systems.

A sample of LiveOps login reports for 34 production systems over a one month period shows several examples of logins. Most notably, nine systems were found to be running screen savers 28 times, by detecting instances of the scrnsave.src process. This implies that primary credentials were left on the systems for extended periods of time. It also showed that each system was logged into at least twice, with one system having 11 logins.

Report: Non-System Daemons

Windows systems joined to an active directory have machine accounts associated with them, which can be used as the user context for running daemons. Machine accounts have the same properties as user accounts and can be added to access control lists (ACLs) if needed. It is best practice to run all daemons using the machine account [6] because this will typically not have permissions on remote machines, and will not require storing user credentials (login names and passwords) locally to run the daemons. Locally stored credentials can potentially be obtained by hackers that infiltrate the system and be used to connect to other systems.

The LiveOps non-system daemon report for 34 machines over a one month period identified several daemons that were not running with the machine account security context. Six distinct processes were identified across all systems, five were management product agents, and one was an LOB process. Each of the 34 systems had violations for at least one daemon running, and a few machines had up to 4. The majority of these daemons were management agents.

Feasibility of Labeling All Changes

The ability to understand all changes made across large numbers of systems is possible only if the rate of new PS is small enough for humans to reasonably comprehend. To evaluate this we determined the first time LiveOps observed process names, PS entries across all managed systems. Using this information we determined:

1. The steady state daily rate of new items observed over an extended period
2. The learning period, which is the number of days taken to observe the majority of items

The learning period determines how long LiveOps must be run before it will reliably generate reports without false positives caused by observing PS for the first time. The steady-state determines how many new PS items must be classified on an ongoing basis for LiveOps to accurately label and understand all observable PS. If labeling is not periodically maintained LiveOps will still accurately and correctly generate all reports, however over time there may be some unknown entries for some reports.

To help prevent this LiveOps provides contextual information that makes it easy to label new PS, and can integrate with other sources descriptive PS information as described earlier. This includes contextual information about the processes using the PS, integration with existing software libraries [24] and work order systems, as well as correlation with previously labeled LiveOps PS. Additionally, automatic inheritance based labeling strategies can be employed such as labeling binaries used by only one process with the process's label.

Figure 9 shows the distribution of newly seen processes from all monitored systems over 39 days since the start of LiveOps data collection. We can see that the learning period is one day, at which time 1000 distinct processes were observed. The steady-state for new processes is typically 0, unless software updates or installations occur. During software updates many processes are created from files created with random temporary file names that do the installation work. There are two solutions to automatically labeling these processes:

1. Using the checksum and size of the process .exe will effectively canonicalize the names of the temporary files into distinct entries across all machines
2. Substring rules can be created to eliminate process names created within known temporary folders.

Figure 10 presents the distribution of newly seen binaries, that is any .dll, .exe, or other executable file that is loaded by any of the monitored machines. Distinct entries are determined by using the checksum stored in the executable header and the size of the binary file. These are the same properties used by kernel debuggers to identify the correct version of symbols to load when debugging a process. We can see that the learning period lasts for one day, when 1400 binaries are first observed, then reaches steady-state where new binaries are only observed if new installations or patches are applied.

The growth of new files and registry entries files since the start of LiveOps data collection is shown in Figure 11, and Figure 12 respectively. For files and registry entries, each day there are a large number of newly generated temporary and cached files with random names. These entries are filtered out in daily counts using simple substring rules that look for known temporary paths in the names. We can see from both graphs that the daily newly observed entries are

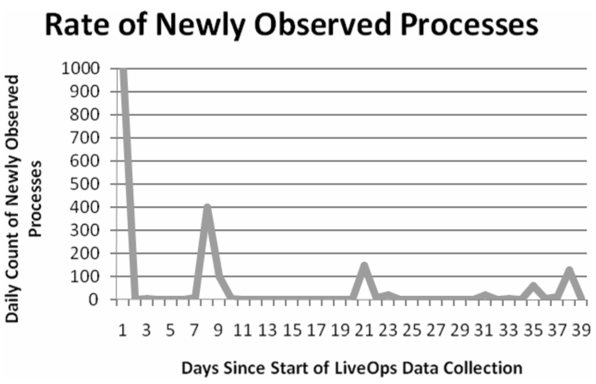


Figure 9: Growth of new processes since the start of LiveOps data collection.

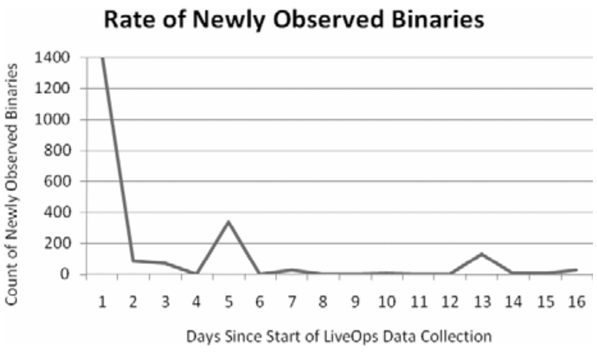


Figure 10: Growth of new binaries since the start of LiveOps data collection.

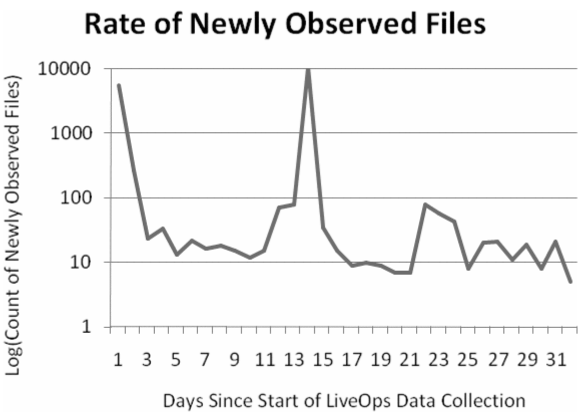


Figure 11: Growth of new files since the start of LiveOps data collection, filtering temp entries.

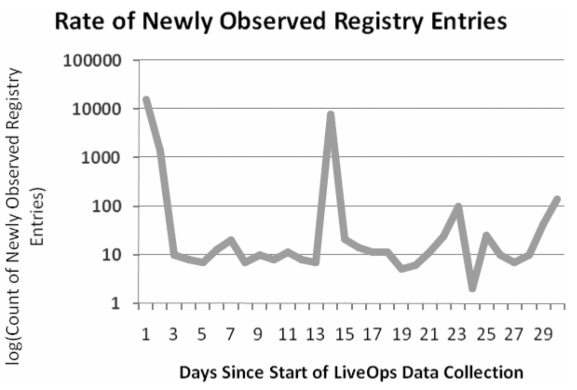


Figure 12: Growth of new registry entries since the start of LiveOps data collection, filtering temp entries.

O(10^1) across all systems, unless new installations or configurations are made, such as those that happened on the 14th and 23rd day. This is a reasonable amount for humans to manually evaluate.

Conclusion

We built LiveOps, a scalable systems management service and comprehensive low-overhead agents, that identify security vulnerabilities, perform compliance auditing, enable forensic investigations, detect patching problems, optimize troubleshooting, and detect malware/intrusions. The service provides a platform for knowledge sharing across organizations and administrative boundaries and allows for seamless integration between analysis results from disparate management products that build on it.

Our analysis of reports from deploying LiveOps in MSN demonstrates how it fulfills a critical need in the configuration management cycle by detecting all system changes, enabling verification of approved requests, and alerting upon unknown modifications. Closing this loop is becoming increasingly important for all datacenters for identifying unwanted changes, and to fulfill auditing requirements. These results illustrate the benefits and feasibility of managing configuration from the OS perspective of interactions between running processes and PS.

LiveOps empowers administrators with new visibility into the relationships between applications and the PS they use. This enables them to more knowledgably, efficiently, and accurately manage their systems and the applications that run on them.

Author Biographies

Chad Verbowski is an Architect in Microsoft Research. His early academic research on network management translated into a job designing network and systems management infrastructure for MFS Datanet. After surviving the WorldCOM takeover Chad worked at Cisco before joining a management focused software startup company as employee number 5. He eventually arrived at Microsoft where he worked on headless support in Windows 2000, then ran the core development team for the first release of Microsoft Operations Manager before finding his niche at Microsoft Research. At MSR Chad cofounded the Cybersecurity and Systems Management research group, where he focuses on his area of interest: reducing complexity in software.

Juhan Lee is an MSN Architect focused on improving reliability, scalability, and improving operation of MSN's Internet Service Platforms by innovating next generation MS products and research technology into MSN data centers. Juhan joined Microsoft in February 1996 to lead Windows 2000 Datacenter development, Systems core components of Management Server (SMS), and Microsoft Operations Manager 2000. Prior to Microsoft Juhan worked on IBM's CICS/DB2 and OS/2 product lines in Research Triangle Park, and directed distributed middleware and Lotus workflow engine R&D at UNUM Corporation that was ultimately licensed by large software corporations. Juhan enjoys all things electronics and works on home electronics on his free time. He studied Electrical Engineering at North Carolina State University, interned at Nortel and IBM as both Power Engineer and Software Developer where he chose software career by joining IBM in 1988.

Xiaogang Liu is a Software Development Engineer who joined Microsoft in 2005 and has been working on automatic diagnosis and discovery of system anomalies since then. Before that, he was the dev lead of an automatic computer test and configuration project, which has been deployed to two of the top three PC vendors in China and increased the output vastly. His first project in software industry was a Chinese-English dictionary on the Windows platform, where he's responsible for word/phrase lookup under mouse pointer initially and later the main UI. He likes reading books and playing badminton in his spare time.

Yi-Min Wang is a Principal Researcher at Microsoft Research, Redmond, where he manages the Cybersecurity and Systems Management Group and leads the Strider project. Yi-Min received his B.S.

degree from National Taiwan University in 1986. He received his Ph.D. in Electrical and Computer Engineering from University of Illinois at Urbana-Champaign in 1993, worked at AT&T Bell Labs from 1993 to 1997, and joined Microsoft in 1998. His research interests include security, systems management, dependability, home networking, and distributed systems.

Roussi Roussev is finishing his Ph.D. as a student at Florida Institute of Technology. His research interests include distributed systems, security, dependability and program verification.

Bibliography

[1] Allen, B., "Monitoring Hard Disks with SMART," *LINUX Journal*, 2004, http://www.linuxjournal.com/article/6983 .

[2] Arbaugh, W., et al., "Windows of Vulnerability: A Case Study Analysis," *IEEE Computer*, Vol. 33, Num. 12.

[3] Brodie, M., et al., "Quickly Finding Known Software Problems via Automated Symptom Matching," *ICAC*, Seattle, WA, 2005.

[4] Brown, A., et al., "Accepting Failure: A Case for Recovery-Oriented Computing (ROC)," *HPTPS*, Asilomar, CA, 2001.

[5] Burgess, M., et al., "Cfengine: a site configuration engine," *USENIX Computing Systems*, Vol. 8, Num. 3, 1995.

[6] Chen, S., et al., "A Black-Box Tracing Technique to Identify Causes of Least-Privilege Incompatibilities," *NDSS*, San Diego, CA, 2005.

[7] Doceur, J., et al., "A Large-Scale Study of File-System Contents," *SIGMETRICS*, Atlanta, GA, 1999.

[8] Dunagan, J., et al., "Towards a Self-Managing Software Patching Process Using Black-box Persistent-state Manifests," *ICAC*, New York, NY, 2004.

[9] Ganapathi, A., et al., "Why PCs Are Fragile and What We Can Do About It: A Study of Windows Registry Problems," *ICAC*, Florence, Italy, 2004.

[10] *The GNU Project Debugger*, http://www.gnu.org/software/gdb/gdb.html .

[11] Gordon, L., et al., *CSI/FBI Computer Crime and Security Survey*, 2004, go.sci.com .

[12] Hart, J. and J. D'Amelia, "An Analysis of RPM Validation Drift," *Proceedings of the 16th USENIX Conference on Systems Admininistration*, Berkeley, CA, 2002.

[13] *Japan's Personal Information Protection Act*, 2003 Law Num. 57, Japan, 2003.

[14] *Japan's Personal Information Protection Act*, http://www.privacyinternational.org/survey/phr2003/countries/japan.htm .

[15] Jaques, R., *Internal Hackers Pose The Greatest Threat*, 23 Jun 2005, http://vunet.com .

[16] Kim, G. H., "The Design and Implementation of Tripwire: A File System Integrity Checker," *ACM Conference on Computer and Communications Security*, Fairfax, VA, 1994.

[17] McLean, P., *Information Technology – AT Attachment-3 Interface (ATA-3), X3T13 2008D Revision 7b*, 1997.

[18] *Microsoft Debugging Tools*, http://www.microsoft.com/whdc/devtools/debugging .

[19] *Microsoft Application Compatibility, Toolkit*, http://www.microsoft.com/technet/prodtechnol/windows/appcompatibility .

[20] *Microsoft Baseline Security Analyzer*, http://www.microsoft.com/technet/security/tools/mbsahome.mspx .

[21] *Microsoft Management Products*, http://www.microsoft.com/management .

[22] *Microsoft Program Analysis Projects*, http://www.microsoft.com/windows/cse/pa .

[23] Moshchuk, A., et al., "Crawler-based Study of Spyware in the Web," *NDSS*, San Diego, CA, 2006.

[24] *National Software Reference Library*, http://www.nsrl.nist.gov .

[25] Oppenheimer, D., et al., "Why do Internet services fail, and what can be done about it?" *USITS*, Seattle, WA, 2003.

[26] Oswald, E., "Study: Adware Increasing Exponentially," *BetaNews*, September 11, 2006.

[27] *Payment Card Industry (PCI) Data Security Standard, v1.1*, Sep., 2006, https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf .

[28] Rescorla, E., "Security Holes... Who Cares?" *USENIX Security*, Washington, DC, 2003.

[29] *Sarbanes-Oxley Act of 2002*, http://fl1.findlaw.com/news.findlaw.com/hdocs/docs/gwbush/sarbanesoxley072302.pdf .

[30] Sun, Y., et al, "Global analysis of dynamic library dependencies," *LISA*, San Diego, CA, 2001.

[31] *Syslog*, http://en.wikipedia.org/wiki/Syslog .

[32] Verbowski, C., et al., "Analyzing Persistent State Interactions to Improve State Management," *SIGMETRICS*, Saint Malo, France, 2006.

[33] Verbowski, C., et al., "Flight Data Recorder: Monitoring Persistent-State Interactions to Improve Systems Management," *OSDI*, Seattle, WA, 2006.

[34] Wang, H., et al., "Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits," *ACM SIGCOMM*, Portland, OR, 2004.

[35] Wang, H., et al., "Automatic Misconfiguration Troubleshooting with PeerPressure," *OSDI*, San Francisco, CA, 2004.

[36] Wang, Y.-M., et al., "STRIDER: A Black-box, State-based Approach to Change and Configuration

Management and Support," *LISA*, San Diego, CA, 2003.

[37] Wang, Y.-M., et al., "Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management," *LISA*, Atlanta, GA, 2004.

[38] Wang, Y.-M., et al., "Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites that Exploit Browser Vulnerabilities," *NDSS*, San Diego, CA, 2006.

[39] *Windows Error Reporting*, http://www.microsoft.com/whdcmaintain/WERHelp.mspx .

[40] *Windows Event Log*, http://windowssdk.msdn.microsoft.com/en-us/library/ms732118.aspx .

[41] *Windows Memory Diagnostic*, http://oca.microsoft.com/en/windiag.asp .