

BAE SYSTEMS

SAGE™

Standard Automated Guard Environment

Technical Overview

ABSTRACT

The BAE Systems Information Technology Standard Automated Guard Environment (SAGE™) is a development environment for building Trusted Guard applications to run on BAE - IT's XTS-400™ Linux-compatible trusted computing system. SAGE Guards are designed to securely, automatically, and efficiently monitor a carefully controlled flow of data between systems or networks with different security characteristics (e.g., different classification levels, different "needs to know"). While the filters that perform the content checking and apply the Guard's security policy are written by the developer, SAGE includes a set of standard routines that perform the Guard's transaction handling, regrading (reclassification/relabeling), audit logging, and other generic Guard functions. SAGE also includes customizable Guard configuration and status monitoring tools that use the X-Windows graphical user interface. SAGE has been used successfully in several major US government and friendly government Guard applications. References are available.

SAGE was developed and documented using trusted software principles to ease the burden of accreditation. Several SAGE Guards have already been accredited, and future accreditation efforts may benefit from the previous certification work. In connection with the Security Test and Evaluation (ST&E) and accreditation of SAGE-developed Guards, BAE - IT will provide relevant SAGE design documents under Non-Disclosure Agreement to the government's Designated Accrediting Authority.

XTS-400™ Standard Automated Guard Environment Technical Overview

Michel W. Focke
XTS Product Manager
BAE Systems Information Technology
2525 Network Place
Herndon, VA 20171 U.S.A.
TEL: (703)563-8087
FAX: (703)563-8013
mike.focke@baesystems.com

TABLE OF CONTENTS

1	INTRODUCTION	1
2	TRUSTED GUARD DEFINED.....	3
3	SAGE CAPABILITIES.....	5
3.1	SAGE ARCHITECTURE	5
3.1.1	SAGE Application Layer.....	6
3.1.2	Platform Abstraction Layer.....	8
3.2	SAGE GUARD PROCESSING.....	8
3.2.1	Transaction Initialization.....	9
3.2.2	Input Processing.....	11
3.2.3	Preprocessing.....	11
3.2.4	Content Validation.....	11
3.2.5	Post-processing.....	11
3.2.6	Output Processing.....	11
3.2.7	Transaction Termination	11
3.2.8	Auxiliary Processes.....	12
3.2.9	Human Reviewer Support.....	12
3.3	SAGE GUARD DEVELOPMENT.....	12
4	PHILOSOPHY OF PROTECTION	13
4.1	LEAST PRIVILEGE	13
4.2	MANDATORY SECURITY POLICY.....	13
4.3	MANDATORY INTEGRITY POLICY.....	13
4.4	DISCRETIONARY ACCESS CONTROLS	13
4.5	TRUST FEATURES OF FRAMEWORK.....	13
4.6	NON-EVALUATED COMPONENTS	14
4.7	ADDITIONAL ASSURANCE FROM THE USE OF THE STOP OPERATING SYSTEM	14
5	SAGE DELIVERABLES	15
5.1	SAGE DEVELOPER'S TOOLKIT	15
5.1.1	Application Template Guard.....	15
5.1.2	Documentation.....	15
6	5.2 GUARD DEVELOPMENT SERVICES	18
7	WHY WOULD YOU USE THE XTS-400, STOP AND SAGE TO DEVELOP YOUR GUARD?.....	20

LIST OF FIGURES AND TABLES

Figure 1. SAGE Architecture.....	4
Figure 2. SAGE Data and Transaction Flow.....	8

Copyright © 2003 by BAE Systems Information Technology, LLC
SAGE, XTS-400, and STOP are trademarks of BAE Systems Information Technology, LLC
FORTEZZA is a registered trademark of the U.S. National Security Agency.

1 INTRODUCTION

The BAE Systems Information Technology XTS-400™ Standard Automated Guard Environment (SAGE™) is a set of design concepts, interface definitions, executable code, and accreditation documentation. SAGE is a development environment for building Guards; SAGE is not an executable Guard itself but is used to develop executable Guards which run on BAE - IT's XTS-400™ trusted computing system.

The general objective of a SAGE Guard is to securely, automatically, and efficiently allow a restricted flow of data between two systems or networks with different security characteristics. While the security policy can be customized by the Guard developer, SAGE has been designed to accurately enforce that policy, and to protect data from unauthorized disclosure or modification while the data resides on the XTS-400 system. SAGE has been developed and documented using trusted software principles to ease the burden of accreditation. Several SAGE guards have already been accredited and future accreditation efforts may benefit from the previous work.

SAGE supports a one-way flow of data from a single source or multiple sources to a single destination or multiple destinations, possibly all with different access constraints. Multiple software Guards of different types can run simultaneously on the same hardware platform, resulting in a functionally n-way data flow with different security policies governing the traffic on each path of data transfer. SAGE includes extensive and extensible Guard event logging with a configurable ability to send event records to the underlying XTS-400 audit file. Guard configuration and status monitoring tools are provided that utilize the X-Windows interface.

2 TRUSTED GUARD DEFINED

In the simplest terms, a Trusted Guard is a software application that automates the job now performed by human review-and-release officers. The Guard enforces some one or more security policy rules that enable it to determine whether a piece of information should be allowed to be released from the enclave (system, domain, or network) to another enclave at a different (lower or higher) sensitivity level (e.g., a different classification or need-to-know). In this way, the Guard replaces the “air gap” between the two enclaves. The Guard, in short, is designed to ensure that no information flows out of an enclave that is not allowed to flow out of that enclave.

Some Guards also enforce admittance policies governing what information may flow into an enclave (or domain), in essence protecting that enclave from the outside world. Like a firewall, this type of Guard is designed to protect the “inside” network from “outside” networks beyond its control. Unlike a firewall, a Guard achieves this protection by ensuring that only carefully vetted pieces of information flow into the protected enclave/domain.

At a superficial level, Trusted Guards seem to share characteristics with firewalls. The “inside/outside” paradigm applies for both types of systems. Both systems are meant to control the connection between two networks that would otherwise probably not be allowed to be connected. But this is where the similarity ends. While a firewall controls connections, either by opening or not opening a connection based on the source and destination IP address/ port number (in some cases authenticated, in others not) and the communications protocol being used, a Guard controls the content sent over a particular connection.

A SAGE Guard achieves this control in the context of the trusted and highly evaluated STOP operating system, which is much less likely to contain any operating system level security flaw and is far less susceptible to attack than the typical firewall platform.

The Guard’s communications paradigm is virtually always connectionless. A Guard does not simply open up a pipe and scan bits going over that pipe. Instead, it enables and controls the transaction-oriented flow of individual pieces of information - e-mail messages, data files, protocol data units, or other discrete units of data. The transaction-oriented, connectionless nature of the

Guard helps minimize potential covert timing channels that may be opened when mediating the connection between two enclaves at different sensitivity levels.

Unlike a firewall, a Guard is usually expected to enforce a more complex and granular content-oriented set of security policy rules governing the releasability or admissibility of discrete pieces of information. A Guard may parse the header and body of data files or e-mail messages, scanning each data element in the header and body to ensure its conformance with a programmed set of rules that define the characteristics that make that type of data element permissible to release (or admit). These data element characteristics may have to do with identification, authentication, and authorization/permissions of the sender and/or recipient of the information (based, for example, on a digital signature and X.509 certificate). The Guard may check for the correct encryption of the information, or the format of the content.

For example, the Guard may parse each data field in a fixed format data file to ensure that the data in every field conforms with the specific formatting rule governing that field (e.g., “data in field #3 must be alphanumeric”), whether any numbers it contains fall within certain high/ low thresholds, or whether any text strings it contains include “dirty words” (e.g., sensitive “code words”); dirty word scans can also be performed on free text files such as e-mail messages.

In this way, a Guard can enforce a complex combination of security policy enforcement processes that may include scanning the header and content of every file, message, attachment or protocol data unit (PDU) sent from or to the inside enclave to ensure that this piece of information satisfies all criteria by which the Guard may be permit it to flow to/from the outside enclave.

The trusted operating system used by the Guard is another aspect that sets it apart from a firewall. While both systems implement the idea of “inside” and “outside”, with the “inside” being inherently more sensitive than the “outside”, the Guard uses a multilevel secure (MLS) trusted operating system (STOP), which enforces a mandatory access control policy. This means that the notion of the higher and lower sensitivity of the “inside” vs. the “outside” is enforced in fact by the Guard’s operating system, and not just in theory (as on a firewall).

But even on a Guard, the “inside” need not be hierarchically more sensitive than the “outside”—it may simply be of a different sensitivity based on need to know. The difference is that in the Guard, with its MLS trusted operating system, the separation of need to know can be enforced by a mandatory access policy as well. At a practical level, this means that the separation of enclaves (whether based on hierarchical or non-hierarchical differences in sensitivity) is enforced with a higher degree of assurance than on most firewalls, which have no mandatory access controls.

What SAGE adds to the assurance equation is the underlying XTS-400 STOP operating system. The evaluation of STOP by NSA (as of this writing (April 2003), STOP’s Common Criteria EAL 5+ evaluation has been in progress for several months. The thoroughness of the evaluation can be seen from the 14 month length of time that the complete evaluation of even a repeatedly evaluated operating system like STOP takes. While Windows and Trusted Solaris were evaluated at EAL-4+, STOP is choosing to be evaluated against a significantly more demanding set of functional requirements and at the EAL-5+ level. EAL 5 and higher evaluations require NSA to do additional penetration testing, thus significantly differentiating EAL-5 operating systems from EAL-4 operating systems which do not receive this higher level of penetration testing. Prior versions of the STOP operating system were repeatedly evaluated by the National Computer Security Center at B3 under the now-obsolete TCSEC evaluation program).

The XTS-400 and its STOP trusted operating system contribute to the trustworthiness of SAGE-based Guards by providing a certified high-assurance operating environment that enforces mandatory separation of Guard-connected enclaves (except where transfer of data between those enclaves is strictly controlled by the Guard). Use of a high assurance evaluated operating system to support the SAGE Guard has been proven to ease the accreditation of such Guards. And the XTS-400’s unique integrity and process isolation mechanisms protect SAGE Guard executables and isolate system objects in a way that no other available operating system can.

3 SAGE CAPABILITIES

3.1 SAGE Architecture

SAGE is a client/server, transaction-oriented infrastructure. To minimize the application coding required of the developer, it provides the common

elements most Guards require. SAGE is designed to ease the application coding effort and allow the code or configuration to be easily modified to support alternate or additional requirements. Figure 1 illustrates the architecture of SAGE.

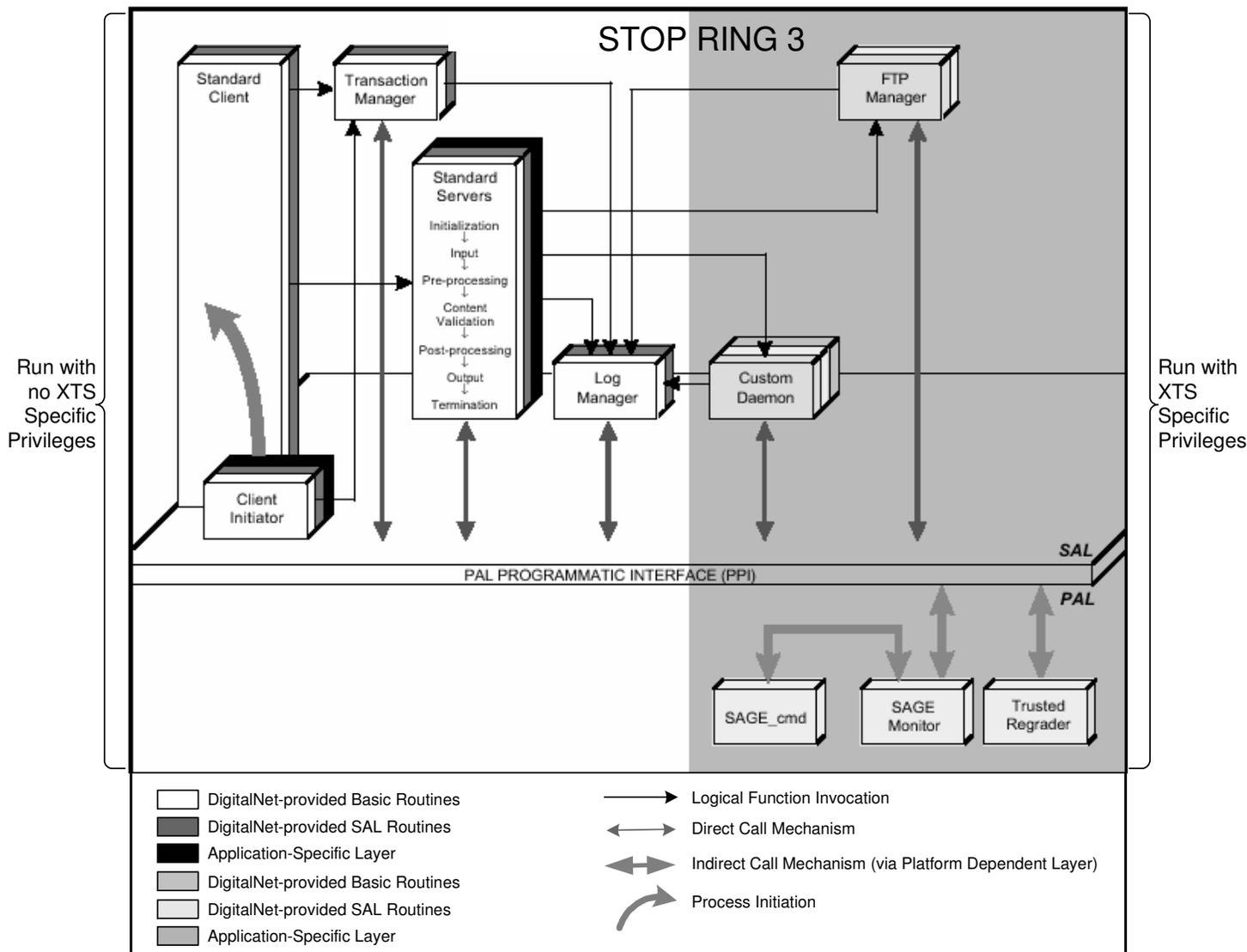


Figure 1. SAGE Architecture

SAGE comprises a series of portable modules that enable it to be easily customized to meet specific Guard requirements with minimal change to the overall design. At the core of this standard Guard framework is a transaction control and execution environment designed specifically to automate the enforcement of security policies associated with Trusted Guard applications. In addition, BAE - IT's developers determined that many common Guard functions could be implemented by standard code that would change very little from one Guard to the next. These common Guard functions include:

- transaction control and management;
- initialization of data for each transaction;
- input and output functions for each data path;
- preprocessing of data for each transaction;
- content validation of data for each transaction before transfer through the Guard;
- post-processing of data for each transaction;
- configuration of the Guard software through an X-Windows-based tool;
- auditing of Guard events and viewing audit logs through an X-Windows-based tool;
- status monitoring of Guard processes through an X-Windows-based tool.

The customizable elements of the common Guard framework are the security policy, along with any application-unique mechanisms required to enforce that policy. The Guard also supports controlled circumvention of system security policy but strictly limits such “violations” to processes safely isolated in the underlying trusted operating system. This isolation of security “violations” ensures that the Guard application can run without privileges, easing its accreditation.

The Guard framework is modular, with discrete functions communicating among themselves via system calls and APIs, enabling the easy integration of commercial-off-the-shelf (COTS) and/or government-off-the-shelf (GOTS) software packages to implement capabilities required by the specific Guard application. For example, the pre- and post-processing functions of the Guard framework could be easily extended to support other capabilities, such as digital signature and encryption algorithms or message handling protocols. Similarly, the context validation function could be extended to use content-analysis software such as COTS natural language processor for parsing text

messages, or an expert system for defining and enforcing complex security policies. Because the STOP operating system is Linux™ compatible, many libraries and applications are available for use with STOP and SAGE to ease Guard development time and reduce development costs.

In addition, the Guard framework provides a full set of extensible application programmatic interfaces (APIs) to support services in the underlying trusted operating system (STOP) of the computing platform, enabling the implementation of various communications protocols, etc., as needed by the specific Guard application. By making it easy to link in COTS and GOTS software packages, the Guard framework enables the rapid implementation of sophisticated Guard applications with a minimum of custom development required. Thus, a SAGE-based Guard will comprise bound units provided by BAE - IT combined with application-specific components provided by the Guard developer/integrator.

A SAGE-based Guard consists of two major software layers: the Platform Abstraction Layer (PAL) and the SAGE Application Layer (SAL). The PAL encapsulates platform dependencies, including privileged software, to moderate communication between SAL processes at different security levels.

3.1.1 SAGE Application Layer

The SAGE Application Layer (SAL) provides the SAGE-based Guard with its platform-independent application functionality. The SAL is implemented as totally unprivileged ANSI C code that is fully portable across multiple platforms. The lack of privileges required by the SAL code not only eases accreditation of SAGE-based applications; it contributes to the SAGE Guard's development. The SAL code can be programmed and tested on another system (perhaps a Linux system) and, because the Guard will not have to be reprogrammed at the application layer to “violate” the security policy of the STOP operating system, the executable can be copied into the XTS and executed. Instead, SAGE is designed to isolate privileged functions to the Platform Abstraction Layer, and to provide APIs that enable the Guard to exploit the underlying privileged functions to perform security functions, such as controlled policy violations, by making an API call to a facility in the underlying platform that is already privileged to perform such violations. SAGE ensures that when such “violations” are requested by the Guard, necessary flow checks are performed, and the violation logged and audited.

The SAL Standard Client and Standard Servers communicate indirectly with each other by passing messages through the Platform Abstraction Layer (PAL). Because it is unprivileged, the SAL cannot violate the Discretionary or Mandatory Access policies of the underlying operating system. Any security policy violations required by SAL Guard processing are implemented via APIs from the SAL to the PAL, and within the PAL by the privileged Monitor and Regrader processes. The SAL includes the components described below.

3.1.1.1 Client Initiator

The Client Initiator determines when a new transaction is ready to be processed, then causes a Standard Client to be initiated to process that transaction. For each Guard application, the Client Initiator is customized to include an application-specific mechanism to “trigger” its decision that a transaction is ready to be processed.

In the general, steady-state condition of the Guard, a “trigger” event occurs. The Client Initiator recognizes the event and requests a transaction ID from the Transaction Manager. The Client Initiator then starts the transaction on its way through the Guard’s Standard Server processes (i.e., Initialization, Input, Preprocessing, Content Validation, Post-processing, Output, and Termination).

3.1.1.2 Standard Client

The Standard Client invokes the Standard Servers listed in the system configuration file and controls the flow of a transaction through those Standard Server processes.

3.1.1.3 Standard Servers

The Standard Servers are stateless daemons invoked by the Standard Client. The Standard Servers call various SAGE Common Support Routines, PPI routines, and application-specific routines to perform their specific functions. The Standard Servers include the Transaction Initialization, Input, Pre-processing, Content Validation, Post-processing, Output, and Termination Servers. Each Standard Server is modular and fully extensible, allowing developers to easily “plug in” application-unique functions. For example, the Content Validation Server may be extended to call a standard COTS natural language processor to parse complex messages, thus enabling the Server to determine whether those messages meet or violate the defined security policy being enforced.

Similarly, the Pre- and Post-Processing Servers may be extended to call an encryption mechanism to decrypt and re-encrypt data before and after Content Validation.

As the communication mechanism is implemented in the Platform Abstraction Layer (PAL), the Input and Output Servers call the PAL to transport data to and from the Guard.

3.1.1.4 Custom Daemon

Custom daemons can be written by the developer and configured into the Guard, when needed, to implement trusted (privileged) functionality in the SAL. These daemons are not part of the Standard Server transaction processing flow. Instead, they provide services useful for a specific Guard application, e.g., encryption. The Standard Servers and/or other Custom Daemons can call “services” which have been “registered” by the Custom Daemon. Multiple custom daemons can be run at the same mandatory level. Custom Daemon functions should be strictly controlled and limited in complexity to minimize impact on the accreditation of the resulting Guard.

3.1.1.5 Log Manager

The Log Manager provides a common audit collection service to the Standard Servers and the Standard Client. The SAGE Log Manager can be configured to collect a security administrator-defined subset of security events in the SAGE log file, then to pass those events to the PAL for writing to the system audit file, where SAGE-based Guard audit events are combined with operating system-level audit events to provide full auditing of pertinent security-relevant activities. Thus, SAGE enables the easy addition of Guard-specific log events to the system log file.

3.1.1.6 Transaction Manager

The Transaction Manager is the SAGE server responsible for maintaining transaction state information and the inter-process data space used by the customized application components (i.e., the Standard Servers). The Transaction Manager supports the stateless operation of the Standard Servers by providing transaction identification, and by monitoring and managing the state of every transaction as it is processed through the Guard.

3.1.1.7 FTP Manager

The FTP (File Transfer Protocol) Manager provides a way for files to be imported and/or exported to the Guard. The FTP Manager transfers files via the FTP protocol implemented directly in the SAGE library and in FTP Custom Daemons implemented as part of the SAGE infrastructure. The FTP Manager registers its FTP services (e.g., put, get, delete, etc.) upon Guard start-up. SAGE processes request the use of these FTP

services. The SAGE Monitor can mediate access to these FTP services, thus verifying (through security level and privileges) that only appropriate processes may request these services. The FTP Manager can not be customized.

3.1.1.8 Status Monitor

The Status Monitor is an X-Windows-based display program for observing Guard functions in progress. The Status Monitor reads log files generated by the Guard and displays them in an X-Window interface while the Guard is running, to enable administrator tracking of the progress of Guard transactions and troubleshooting.

3.1.1.9 Guard Configuration Tools

The Guard Configuration Tools are X-Windows-based utilities for defining the Servers to be called by the Standard Client, and the events to be logged by the Log Manager. Whereas a Guard developer can directly edit the configuration files, these tools provide the only means by which the Guard administrator can modify an installed Guard's configuration before start-up.

3.1.2 Platform Abstraction Layer

The Platform Abstraction Layer (PAL) includes a library of software functions that provide a standard programmatic interface to platform-specific operating system, communications, and security services. The peculiarities of the platform are hidden beneath the PAL Programmatic Interface, which isolates the SAL from platform-specific services, while providing it with a standard interface to request those services.

Within a SAGE Guard, only the PAL's Platform-Dependent Layer (PDL) functions are privileged to violate the underlying operating system's mandatory and discretionary security policies. PPI and PDL services include:

- Guard control: Guard start-up, termination, and status monitoring, invoked and terminated by a trusted command;
- Service Dispatch: inter-process communication mechanism that allows SAL processes to request services from other SAL and/or PAL processes;
- Capability Enforcement: restricts PAL services available to particular SAL processes;
- PAL Configuration: performed at Guard start up; may include defining such things as the list of capabilities for each Standard Server;

- Library of platform-dependent interface routines: used by both PAL and SAL processes.

The PDL includes functions for low-level transaction routing and process management. These functions are performed via privileged processes (daemons), including:

- SAGE Monitor: responsible for initiating start-up and shutdown of Guard processes, and for routing and relaying messages between authenticated Standard Clients and authenticated Standard Servers. The SAGE Monitor ensures that the Server(s) communicates only with a Client at the same classification level unless an explicit regrade is requested via the Regrader.
- Guard start-up/shutdown command (SAGE_cmd).
- Trusted Regrader: acts in response to messages from the Standard Server enforcement modules which request it to perform any violations of operating system security policy needed to enforce the Guard's security policy (for example, reclassification of data). The SAGE Regrader preserves both the mandatory and discretionary labels on relabeled files.

3.2 SAGE Guard Processing

The following paragraphs describe the steps involved in a SAGE guard's processing of a transaction. This transaction flow is illustrated in Figure 2. Though they are not shown in Figure 2, a Custom Daemon and FTP Manager can be utilized in the SAGE Guard transaction flow. The Custom Daemon framework feature provides the ability for Guard developers to implement privileged or non-privileged daemons within the SAGE environment to implement miscellaneous functionality not normally associated with the standard SAGE servers. The FTP Manager framework feature provides a way for guard developers to import and/ or export files to the XTS-400 for processing by a SAGE guard via FTP network connections.

SAGE Guards are based on a client/server model with platform-specific communication mediation. All communication occurs via messages sent between the Monitor in the PAL and the Standard Client and Standard Servers in the SAL. No direct communication occurs between the Standard Client and the Standard Servers.

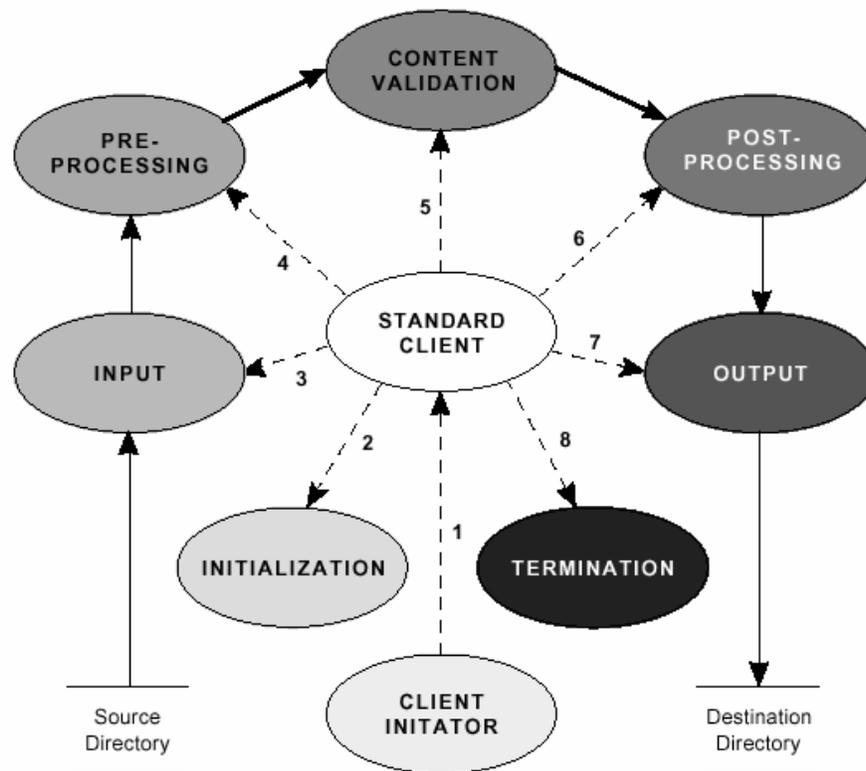


Figure 2. SAGE Data and Transaction Flow

A transaction is processed through the Guard in steps scheduled and managed by the Standard Client. To facilitate serialization of Server processes, the Transaction Manager stores each transaction ID and information about the transaction, including its current Mandatory or Discretionary Access level plus a fixed “scratch pad” buffer space to be used by the transaction. The ID associated with a transaction remains unique, even if the Guard is stopped and restarted. As some SAGE Servers take longer than others to initialize, SAGE will delay processing of transactions until all Servers are initialized. Each SAGE Server is restricted to retrieving transaction information only for the transaction currently being processed.

3.2.1 Transaction Initialization

The first step in SAGE Guard processing, Transaction Initialization, allows for application-specific initialization processes to be performed on a per-transaction basis. Before initialization, the Guard “idles” in a steady state awaiting a trigger event. The Client Initiator recognizes this event and requests a transaction ID from the Transaction Manager via the

PAL-level SAGE Monitor. The Client Initiator then requests the launch of a Standard Client process, and passes the transaction ID to that Standard Client. The mechanism the Client Initiator uses to recognize the trigger event depends on the requirements of the specific Guard application. The event might be the appearance of a file in a directory, or some other event. The logic for Client Initiator trigger recognition is implemented in the application-specific portion of the Client Initiator.

From this point, the Standard Client takes control of the flow of Server invocations required to process the transaction. The Servers to be invoked are listed in the Guard configuration file. If any Server returns a failure, the Standard Client logs the failure and calls the Termination Server (if it is configured) to unregister the transaction and perform application-unique housekeeping. If all Servers succeed, the Standard Client logs the success, passes the transaction out of the Guard to the destination point, and calls the Termination Server to unregister the transaction and perform housekeeping.

3.2.2 Input Processing

The second step in SAGE Guard processing is performed by the Input Server, which brings data into the Guard for security policy enforcement. Input processing gets the data from the input mechanism for further processing by the other Standard Servers. While they reside in the Guard, data are available to the Standard Server processes that will implement subsequent Guard processing only when those processes call application-specific logic. The Input Server ensures that data transferred from the originating system are stored in a high-integrity interim directory on the XTS-400. While they reside in this directory, the data are available to the processes that implement subsequent Guard processing steps when those processes call application-specific logic. How data are handled as they transit the Guard in other SAGE implementations is an application-specific detail. SAGE Input Processing provides interfaces to both SMTP and FTP import/export mechanisms.

3.2.3 Preprocessing

The third step in SAGE Guard processing, Preprocessing, provides a point in the transaction flow during which the Guard can perform transaction format processing, such as confirmation that it is working with well-formed records. Some reformatting of the data may occur during Preprocessing. For example, the Preprocessing Server could be customized to:

- parse packet headers to validate header structure;
- extract identification information from headers;
- compute digital signatures or recompute checksums;
- decrypt encrypted data;
- extract identifiers, sensitivity labels, date/time stamps, and other header data for maintenance by the Transaction Manager to ensure its correct reapplication to the data after Content Validation;
- generate messages caused by failed preprocessing;
- log security events according to defined audit collection criteria.

3.2.4 Content Validation

The fourth step in SAGE Guard processing, Content Validation, is the heart of the SAGE Guard implementation. Content Validation provides a point in the transaction flow in which the Guard can execute customized, application-specific logic for functions such as checking of data content against application-specific security policy rules, and enforcing those policy rules in onward processing of data by the other Standard Servers. Content Validation can be customized to validate data content against a customer-supplied security policy, generate error, or reject messages caused by failed validations, and to log security events according to pre-defined criteria.

3.2.5 Post-processing

The fifth step in SAGE Guard processing, Post-processing, provides a point in the data flow in which the Guard can perform outbound transaction formatting or other adjustments to meet application-unique requirements of the export mechanism, such as affixing a digital signature and/ or applying encryption.

3.2.6 Output Processing

The sixth step in SAGE Guard processing, Output Processing, provides the point in the transaction flow for transferring the data from the Guard to the export mechanism, e.g., from the SAGE host directory to a target directory on the destination system.

3.2.7 Transaction Termination

The seventh and last step in SAGE Guard processing, Transaction Termination, provides a point in the transaction flow in which the Guard can terminate processing for each transaction. Termination processing might be customized to perform the following:

- retrieve from the Transaction Manager the identifier, sensitivity label, time/date stamp, etc., extracted by Preprocessing and reapply this information to the data;
- delete data from the source directory upon transaction success or failure;
- generate an acknowledgment or non-acknowledgment message to send back to originating system;
- audit security events according to defined audit collection criteria.

SAGE allows the developer to design a Guard security policy and to write Guard application code to enforce that policy. The SAGE infrastructure supports the enforcement of that policy and protects transactions while they are on the Guard system from unauthorized disclosure, modification, and destruction.

A SAGE Guard contains application segments that run within or in conjunction with the SAGE Guard framework. As a trusted operating system, the XTS-400's STOP operating system protects itself from application software and provides process isolation and Mandatory Access Control (MAC) and Discretionary Access Control (DAC) policy enforcement with a very high level of assurance.

3.2.8 Auxiliary Processes

The Custom Daemons and FTP Manager (which is a single-purpose Custom Daemon) are available to developers to augment the services provided by the seven SAGE Standard Servers. Both of these auxiliary processes operate outside of the SAGE Guard transaction flow, but provide services to the Standard Servers that implement that flow.

At any point in SAGE processing, a Custom Daemon may be implemented to assist one of the Standard Servers in its processing, for example, to provide access to a trusted function in the operating system, such as encryption. Similarly, the FTP Manager may be used in conjunction with the Input and Output Servers to handle file transfers into and out of the SAGE Guard.

3.2.9 Human Reviewer Support

After performing its transaction processing function(s), any SAGE Standard Server may request that a transaction be placed "on hold" for rerouting to a queue that can be processed by a human reviewer, (e.g., for manual handling of rejected transactions or other exceptions). SAGE services are provided that enable the Guard administrator to query or update the status of transactions queued for human review. In addition, an external message logging interface is provided for non-Guard processes, such as human reviewer interface programs.

3.3 SAGE Guard Development

SAGE provides a well-structured framework within which programmers can build Trusted Guard applications more quickly and easily than developing those applications "from scratch". The Standard Server Framework provides the entry and exit points to and from each Standard Server, as well as transaction management logic for handling of each transaction

from initialization to termination. The Standard Server Framework simplifies the definition of security policy. Depending on the policy, the programmer will determine which Standard Servers should be invoked, and what each server should do.

In some cases, defining what a Standard Server should do will require integrating custom-built or third-party source or object code modules with the Standard Servers, or by making a call-level interface and runtime linking from the Standard Servers to system-level services. Thus, numerous existing software functions, such as packet header parsing, file content format checking, "dirty word" scanning, virus checking, and other security-relevant functions, can be easily linked into the appropriate SAGE Standard Server.

***EXAMPLE:** A Guard with a security policy that requires the receipt and decryption of encrypted files, followed by the re-encryption and forwarding of those files out of the Guard could be rapidly implemented in one of two ways:*

- 1) *by compiling the encryption driver source code libraries with the SAGE Preprocessing and Post-Processing Servers; or*
- 2) *by writing calls from Pre-processing and Post-processing to a library of encryption handling object code, with each call including arguments providing the encryption key and address space to which resulting data should be returned.*

At compile time, the encryption libraries would be linked into SAGE, to become part of the resulting Guard executable.

SAGE's support servers can also be integrated with third-party software through source/object library integration, call-level interfaces to system services, or data-level integration. For example, output from SAGE's Log Server (i.e., the SAGE log files) can be "fed", through data-level integration, into the STOP operating system audit collection mechanism to provide a single, integrated audit trail for Guard application- and system-level events. If desired, these log files could be provided, through data-level integration, to a third-party audit review/reporting tool.

4 PHILOSOPHY OF PROTECTION

4.1 Least Privilege

One Guard or system process on the XTS-400 cannot modify the attributes or data or another process unless the two processes explicitly share an object (file, device, semaphore, or socket). The STOP operating system's Mandatory Access Control (MAC) and Discretionary Access Control (DAC) enforcement can be overridden by processes that possess special privileges. However, in SAGE Guards, Custom Daemons (including the FTP Manager) are the only applications allowed to run with the necessary privileges to perform these security overrides. A program thus privileged can only be installed by the system administrator.

All other Guard applications run as subroutines within a SAGE framework process and possess no special privileges. A few special processes within the SAGE framework do possess special privileges (e.g., to perform a regrade), but they never call/use application segments; they use only the minimum set of privileges required to perform their function. Those privileges are disabled for all code segments except those that need them. Furthermore, the STOP operating system will not allow privileged processes to be stored at low integrity levels. In these ways, SAGE adheres to the principle of "least privilege".

4.2 Mandatory Security Policy

The mandatory security policy enforced by the STOP operating system prevents unauthorized disclosure of classified transactions in the Guard. Network devices are set to the security level of data flowing across the network. Each transaction is maintained at the same security level as the network from which it came or to which it is going. Only processes running at or above the level of the transaction can read the transaction. Furthermore, a Guard process that interfaces to one network device cannot directly interface to another network device. Each network device is assigned a different security level. An unprivileged process cannot open any device at a different security level.

4.3 Mandatory Integrity Policy

SAGE uses the STOP operating system's mandatory integrity policy to isolate Guard processes and data from non-Guard processes and data. The administrators are directed to restrict untrusted users to integrity levels lower than those at which the Guard processes run. Therefore, because of the

operating system-enforced integrity policy, untrusted users and the processes which run on their behalf cannot modify Guard data, access the network devices, or send messages to Guard processes. In this way, the SAGE Guard uses the STOP's mandatory integrity to prevent access violations that are overt or covert, intentional or inadvertent. Network traffic cannot bypass the Guard because, according to Guard administration procedures, only the Guard software will be configured at the integrity level able to access the network devices.

Included in this integrity isolation is protection against viruses and Trojan horses. The integrity levels also prevent filters from reading data created by non-Guard software. The integrity levels also prevent Guard processes from reading data created by non-Guard software. The mandatory integrity policy also controls use of non-network devices needed by a Guard (e.g., FORTEZZA® card readers), which are set at a higher integrity level and thus made inaccessible to non-Guard software.

4.4 Discretionary Access Controls

Discretionary controls are used to protect a Guard application's structures and executing processes from other Guard applications running on the same XTS-400, and from executing import and export mechanisms. A Guard's directories, files, and processes are owned by a UserID created specifically for that Guard. The permissions on these objects deny write access, except by their owner. These permissions prevent all non-Guard, unprivileged processes from modifying or deleting Guard files or sending messages to Guard processes. Discretionary controls are also used to maintain the privacy of input transactions (e.g., mail messages). The Guard UserID is assigned ownership of the input transaction data file; this file is assigned NULL (i.e., no) "group" and "others" permissions to ensure that no unprivileged processes outside the Guard can read the transaction. Any other file to which portions of the transaction data are copied will also disallow access, except to Guard processes.

4.5 Trust Features of Framework

SAGE Guard filter capabilities are limited by the capabilities provided by the SAGE framework. Unless a filter has been configured to possess certain capabilities, some of the framework services, such as transaction regrade, will not be available to it. This method further limits the damage that could be done by an erroneous application segment. By

default, the SAGE processes possess the minimum set of capabilities needed to get their jobs done and no more: another example of the adherence to the principle of least privilege. Except when a privileged Custom Daemon process is used to implement a Guard filter, the SAGE framework ensures that each filter in the Guard executes in the correct order. Framework code starts each filter, reports filter completion, and keeps track of progress through the Guard. The application pieces can not prevent the starting of a filter and can not modify the filter sequence.

Though several transactions may be simultaneously processed by a Guard, the SAGE framework will keep these transactions isolated to minimize the chance of the status or data of different transactions becoming mixed up. A new control process is started for each transaction; this control process is unaware of the existence of the other transactions, nor does any other process know to which filter any other transaction has progressed. The process framework for each Guard process (i.e., “filter”) handles only a single transaction at a time, and retains no memory of previous transactions. Application segments behave in the same way. If the Guard administrator enables the appropriate log events, the SAGE Guard will log every circumvention of the system mandatory access policy (e.g., during a transaction regrade).

4.6 Non-Evaluated Components

Because of the operating system-enforced process isolation and integrity protection of the Guard processes from non-Guard processes, the Guard processes can execute safely under the protection of the STOP operating system. Neither the Guard framework nor the Operating System Services (OSS) will share data with non-Guard software, nor will the Guard applications share data among each other except through the controlled Guard transaction processes.

BAE - IT wrote SAGE in ANSI C according to strict coding standards to ensure that SAGE-based Guards would consist of consistent, portable, easy-to-accredit code. While SAGE itself has not been evaluated by the National Security Agency (NSA), SAGE has been used to develop accredited, operational Guards. It has been proven trustworthy through certification and accreditation of SAGE-based Guards by NSA, the Defense Information Systems Agency (DISA), the Air Force, and the Department of State. SAGE was developed by BAE - IT using the same software development and configuration management practices used for the

STOP operating system’s development; SAGE has also undergone rigorous correctness testing. SAGE design documentation will be made available, under Non-Disclosure Agreement, to designated accrediting authorities to enable certifiers and accreditors to perform the necessary analysis of the SAGE design and implementation.

4.7 Additional Assurance from the use of the STOP operating system

As a trusted system, the XTS-400 supports auditing of specified events, including site-defined events. SAGE allows the administrator to specify Guard events to be audited (e.g., transaction regrade). In addition, the XTS-400 supports separate operator and administrator roles. Only a designated system administrator can set the mandatory levels of devices, change the clearances of users, install privileged software, or disable audit events. If desired, SAGE’s log files may be “fed” into the STOP operating system audit collection mechanism to provide a single, integrated audit trail for Guard application- and system-level events.

The XTS-400 provides a trusted distribution mechanism for released software. Checksums of SAGE release media are computed at the factory and separately mailed to customers. For the developer and distributor of the Guard application, this same process can be followed. The developer, the production factory and the receiving customer all use the same STOP-provided tool to checksum the media and assure that the end user is running bit for bit what the developer developed and the approving authority approved.

5 SAGE DELIVERABLES

5.1 SAGE Developer's Toolkit

The current version of the SAGE Developer's Toolkit, SAGE 3.n, requires an XTS-400 running STOP Revision 6 or later and the STOP Software Development Environment (SDE). A minimum 18 gigabyte hard disk space and 512 megabytes of memory are required. The SAGE Guard developer should be conversant with ANSI C, the contents of the XTS-400 Application Programmer's Reference Manual, the contents of the SAGE Application Programmer's Reference Manual, and the user's security policy requirements (release and admittance).

The system on which the resulting SAGE-based Guard will run must be an XTS-400 running STOP Revision 6 or later. If SAGE will be used to develop an SMTP or FTP Guard, the XTS-400 must contain at least two Ethernet ports (which will be connected to at least two different local area networks). The system on which the Guard will run does not have to run the SAGE Programmer's Toolkit. SAGE Guards run independently of the development environment from which they came. However, the developers will need the Guard execution configuration for testing any SAGE-based Guards they has built, and for running the Application Template Guard (described below). The Guard administrator and operators should be trained in the use of the STOP trusted commands and trusted editors. BAE - IT offers formal courses in its Herndon Virginia facility.

The SAGE Developer's Toolkit includes:

- PAL executables, including SAGE_cmd, Monitor, and Regrader functions;
- SAL executables, including basic functions, Client Initiator, Standard Server routines, Log and Transaction Managers, Status Monitor, Standard Client;
- Administration utility executables;
- Object code for SAL and PAL configuration files and configuration tools;
- Object code libraries for standard portions of Client Initiator and Standard Server routines;
- Source code templates and makefiles for application-unique portions of Standard Servers and Client Initiator routines;

- SAGE Application Layer (SAL) header file and function library;
- PPI header file and function interface library that supports the SAL libraries;
- Compiler and linker for compiling application-unique components of the Client Initiator and Standard Servers and linking them with standard Client Initiator and Server components;
- SAGE installation tools;
- Application Template Guard;
- Documentation.

5.1.1 Application Template Guard

The SAGE Developer's Toolkit also includes the Application Template Guard (ATG). The ATG is a complete, executable. Fully-functional one-way (low-to-high) SAGE Guard implementation with all support tools in place. The ATG uses all of the SAGE Standard Servers and Custom Daemons, and demonstrates the use of security categories and source file deletion from the input directory. The ATG can be used immediately for demonstration purposes, or to run operationally in environments where a simple low-to-high Guard is required. Installation of the ATG as part of the SAGE Toolkit is strictly optional.

The ATG source code is also provided, as an example to the Guard developer of the form of the Guard application modules. The ATG source code can serve as a starting point for development of a SAGE Guard. The SAGE Guard Administration Manual provides user-level documentation for the ATG, including installation procedures and an acceptance test suite that can be used to demonstrate the performance and features of a generic SAGE-based Guard.

5.1.2 Documentation

The SAGE Developer's Toolkit also includes a generic set of user documentation designed to be customized by the developer to reflect application-specific implementation details. The SAGE Development Environment also comes with the SAGE Application Programmer's Reference Manual. As with SAGE itself, all SAGE design documentation is maintained by BAE - IT under strict configuration control in anticipation of accreditation requirements. In addition, BAE - IT will make any proprietary SAGE development

documentation available, under appropriate Non-Disclosure Agreement, to qualified accrediting authorities.

6 GUARD DEVELOPMENT SERVICES

While Guards can be customer developed using the XTS-400, STOP and the SAGE Development toolkit, it may be advantageous in reducing time to deployment and even in reducing cost to have a group experienced in SAGE-based Guard development participate in or perform the development of your Guard. BAE - IT has a group that has developed many Guards using these same tools and their experience in developing and documenting for accreditation (and even their already developed application specific code) may be leveraged to provide you a quicker and potentially cheaper Guard than doing it yourself. On the other hand, if you desire to develop your own Guard, BAE - IT can act solely as the platform provider and provide the hardware, operating system and SAGE middleware to enable you to focus on just the filter and accreditation.

7 WHY WOULD YOU USE THE XTS-400, STOP AND SAGE TO DEVELOP YOUR GUARD?

Why use XTS-400 instead of other products for a security demanding application?

- Because the XTS-400 promises a more robust set of security enforcement features and a higher assurance that those features do what they are documented to do than competing platforms.
- Because they are built on existing, evaluated, accredited, and proven foundations.
- Because they are built by a group which has successfully taken multiple products through multiple successful evaluations.
- Because multiple successful accreditations have been performed on applications hosted on the XTS. BAE - IT is experienced in assisting the application achieve accreditation.
- Because BAE - IT has applications development experience that can provide everything from product to a turn key application. Services familiar with building security enforcing applications are available from BAE - IT to help you with everything from requirements analysis to software development to accreditation support to deployment support to education to life cycle maintenance. BAE - IT can supply you with as little or as much as you want.
- Because the basic design of the XTS has been repeatedly analyzed and penetration tested by the security experts.
- Because the people who wrote the operating system are committed to the support and enhancement of the product. The help line is answered by a developer.
- Because custom devices or functionality are possible within the security architecture. The group has a history of successfully developing “special” capabilities in response to customer requirements.
- Because software updates and “fixes” are produced using the same processes (tools, CM processes, documentation changes, designs, peer reviews, trusted distribution, etc) that produced the original products.
- Because the hardware and software has a history of successful updates and migration paths that are customer friendly.
- Because SAGE reduces the amount of code you need to develop to produce your Guard.
- Because the history of successfully accredited SAGE-based Guards eases the accreditation process for your Guard thus decreasing your time to deployment.
- Because SAGE comes with accreditation supporting documentation that accrediting authorities are familiar with. This eases the accreditation process for your Guard thus accelerating your deployment.
- Because BAE - IT can design, code and help you accredit your Guard, can act as a platform supplier or anywhere in between.
- Because BAE - IT can support (or help you support) the development, deployment, hardware, software and application for the life cycle of the program’s needs including needed improvements.
- Because the secure product business is one of BAE - IT core strengths and you can be assured of life cycle support for your program.

© Copyright 2002-2004 by BAE Systems Information Technology, LLC.

XTS-300, XTS-400, SAGE and STOP are trademarks of BAE Systems Information Technology, LLC

UNIX is a trademark of AT&T UNIX Systems Laboratories.

FORTEZZA is a registered trademark of the U.S. National Security Agency.

Linux is a trademark of Linus Torvalds.

Red Hat and RPM are trademarks of Red Hat Software Inc.

Intel and Pentium are registered trademarks and Xeon is a trademark of the Intel Corp.

