

NISTIR 7466

**Building Information Services and
Control System (BISACS): Technical
Documentation, Revision 1.0**

Alan Vinh

NISTIR 7466

Building Information Services and Control System (BISACS): Technical Documentation, Revision 1.0

Alan Vinh

*Building Environment Division
Building and Fire Research Laboratory*

November 2007



U.S. DEPARTMENT OF COMMERCE
Carlos M. Gutierrez, Secretary

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
James M. Turner, Acting Director

Executive Summary

The goal of the Building Information Services and Control System (BISACS) is to create a computer based system that will allow external users to be verified and then logged into the system in order to gain access to building information such as its structural layout and/or to monitor a particular building or set of buildings for alerts. The “control” aspect of the BISACS is to allow external personnel to control certain essential devices located within a building.

The BISACS will be used to test a variety of scenarios, protocols and messages for demonstrating methods and metrics for linking external services to building information and control systems.

This document is intended for the technical audience that has a good understanding of computers, the Internet, general Web Services architecture, x509 certificates and other related subject matter. This document describes the components that make up the BISACS and how they are integrated in order to support the Building Information Services and Control System’s various goals. As a project, the BISACS has not matured fully; it is the purpose of this document to describe the underlying support system implemented for the BISACS as well as what has been implemented to date in supporting the alerts monitoring feature of the BISACS.

Table of Contents

EXECUTIVE SUMMARY	I
TABLE OF CONTENTS	II
DISCLAIMER	1
INTRODUCTION	2
THE BISACS ARCHITECTURAL OVERVIEW	3
THE BISACS BASE SERVER (BBS).....	3
THE BISACS PROXY SERVER (BPS).....	3
USER VERIFICATION PROCESS	5
CERTIFICATE AUTHENTICATION PROCESS.....	5
USER AUTHENTICATION PROCESS.....	7
BISACS WEB SERVICES INTERFACES	9
COMMON ALERTING PROTOCOL (CAP) PARAMETERS	13
COMMANDS AND DATA EXCHANGE	15
COMMUNICATION AND USER VALIDATION DATA FLOW	17
ALERTS MONITORING	19
DATA GATHERING.....	19
DATA PROPAGATION.....	19
INTEGRATING THE CLIENT APPLICATIONS	21
APPLICATION SPECIFIC CLIENT	21
WEB CLIENT AND THE USER AUTHENTICATION PROCESS	23
WEB CLIENT SCREEN SHOTS.....	24
SAMPLE CONFIGURATIONS AND SCREEN SHOTS OF THE ALERTS	30
DRILLING DOWN FOR ALERT INFORMATION	38
FUTURE GOALS	40
LESSONS LEARNED	41
REFERENCES	42
APPENDIX A - LIST OF SOFTWARE TOOLS	43
APPENDIX B - HARDWARE PLATFORM USED	44

Disclaimer

Certain trade names or company products are mentioned in this document to specify adequately the software and/or hardware used for developing and supporting the Building Information Services and Control System (BISACS). In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the software and/or hardware used is the best available for the purpose.

Introduction

The operation of modern buildings can support a vast amount of static and real-time data. Static information such as building schematics is vital for security and rescue purposes. There is a need for first responders to be notified of designated building alerts in real-time so that actions can be performed promptly [1]. With modern technology, the capability to keep the first responders community updated with the latest building information during emergency situations as well as the ability to remotely control certain building devices and processes can be realized. The Building Information Services and Control System (BISACS) is an infrastructure consisting of multiple computers and various software modules integrated together to accomplish these goals.

The goals of the BISACS are to develop an information exchange architecture, standards, and performance measures, that enable secure real-time communication of building system information to emergency first responders and to allow limited emergency remote control of building systems. The BISACS is used as a tool to explore alternative ways for how building information such as sensor data, alert data and floor plans can be propagated to external entities such as emergency first responder operations centers and mobile units. The demonstration of the BISACS' functions provides insights on the research and standards issues that must be addressed in order to enable the development and deployment of commercial products. As a project, the BISACS has not matured fully; this document describes the technical aspects of how the various BISACS functions have been implemented. The technical details discussed in this document include: (1) the mechanism implemented to verify users based on user authentication and user authorization processes. (2) The mechanism implemented to remotely access static building information such as building schematics. (3) The mechanism implemented to securely notify remote clients of real-time building alerts. (4) The architecture implemented for a scalable network of computers capable of monitoring alerts from a small set of buildings up to nation-wide coverage of alerts. (5) The infrastructure implemented to support remotely controlling a building's various devices.

This document describes the technology and mechanisms used in a computer based system to allow users to be verified then logged into the system in order to gain access to building information such as its structural layout and/or to monitor a particular building or set of buildings for alerts. The "control" aspect of the BISACS is to allow local or remote personnel to control certain essential devices located within a building; the "control" function of the BISACS is currently under development as of this publication. The basic mechanism for obtaining building information such as schematics is in place and is demonstrated by using Joint Photographic Experts Group (JPEG) images [6]; as of the release of this publication, the actual data storage and retrieval processes using databases for building information data have not been implemented.

This document presents the basic software infrastructure that has been developed as the building blocks to support the functions required for the BISACS. The BISACS serves as a proof of concept showing how building information exchange and controls could be managed to support the needs of emergency first responders.

In order to reach the various goals for the BISACS, a user verification process and a data exchange process must be established. The user verification process involves authenticating the user ID and password along with a certificate of authenticity as well as authorizing the user's access privileges. For the user verification process, a server was created to accept and validate electronic x509 certificates [3] via a secure socket port. A web services interface was created to support user identification and password verification. For data exchange, other web service interfaces were created to be generic enough for bidirectional data exchange such that alerts can be sent out of the building while commands can be sent in to the building to control devices or processes. Using software modules, a mechanism for monitoring alerts for a network of buildings was demonstrated. The technical aspects of how these processes work as well as how the alert monitoring mechanism works is described in this document.

In order to understand the interaction between the components of the BISACS, this document describes the basic building blocks that have been implemented then it presents how these components fit together to form the system. Basic building blocks such as the user verification process, the data exchange process and the protocol used for the data exchange are presented first before the integration of the BISACS is described.

The BISACS Architectural Overview

The BISACS consists of a network of servers that monitor entities such as sensor devices. Devices such as building sensors send alerts to the BISACS servers and the BISACS servers propagate the information to various nodes within its network. Client applications monitor one of these nodes from the BISACS network of servers to obtain their alert information and appropriate personnel can respond to the alerts accordingly.

The two main components of the BISACS network are the BISACS Base Server (BBS) and the BISACS Proxy Server (BPS). The following sections will describe them in details.

The BISACS Base Server (BBS)

The BISACS Base Server is the component (a.k.a. the BBS node) that accepts alert notifications from various devices or from Services Interfaces (SIs). The SI is a software component that controls one or more devices belonging to a network of monitored devices. The specifics of how various devices communicate within a network can be protocol specific; hence the SI behaves as the proxy for its own network to hide any proprietary information. The SI takes care of managing all devices that it controls and sends any alerts from those devices to the BBS if configured to do so. The SI also handles all requests and commands destined for devices that it controls via its web services interfaces.

The BBS resides at the lowest node in the BISACS network of servers and provides the following functions:

- User authentication and authorization services
- Proxy for certificate services
- Repository for alerts from all devices that it monitors
- Propagates outgoing alerts and manages incoming requests and commands from external clients

The BISACS Proxy Server (BPS)

The BISACS Proxy Server is the component (a.k.a. the BPS node) that queries for alert notifications from various BISACS Base Servers or BISACS Proxy Servers. The BPS operates as an active client; it polls for alerts from other nodes that it monitors within the BISACS network of servers. Since the BISACS network of servers is composed of multiple web servers exposing their web services interfaces, the BPS is designed to actively poll those web services for alert information. With proper authentication and authorization, this architecture allows for any web browser to view alert information from any of the nodes within the BISACS network of servers. The BPS resides one or more levels above the BBS in the BISACS network of servers and provides the following functions:

- User authentication and authorization services
- Repository for alerts from all nodes that it monitors

Figure 4 below illustrates a BBS protecting two Services Interfaces, the BACnet Services Interface (BSI) and the Sensor Network Services Interface (SNSI). External to the firewall resides a BPS that first responders such as fire fighters and the police department can monitor for alert notifications by using a web client or an application specific client. With proper authentication and authorization (more on this later), devices and Services Interfaces external to the BBS's firewall may also send alerts to the BBS. Notice that alerts can be monitored via the BPS but any requests or commands destined for devices must go through the BBS since it has access and control over those devices.

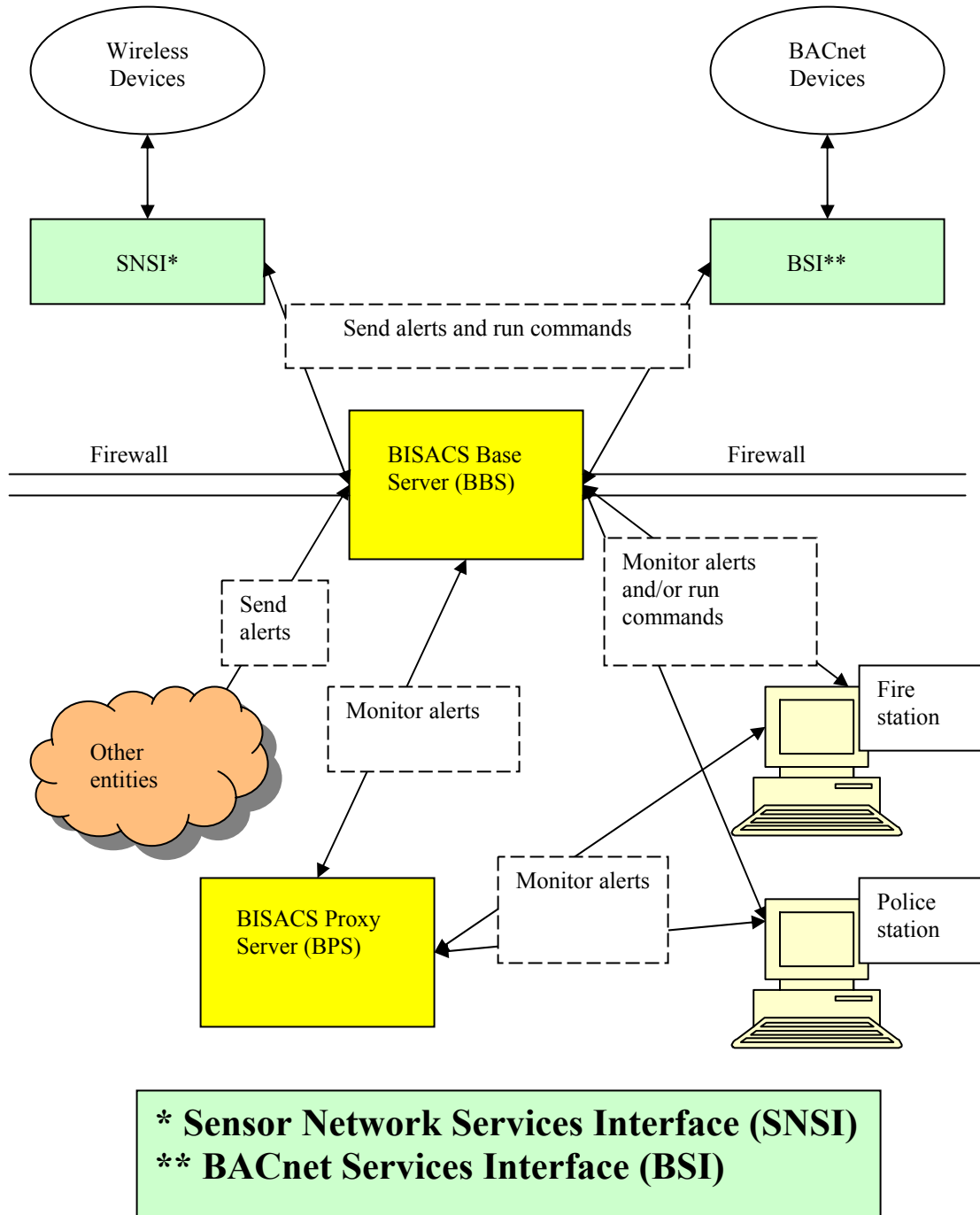


Figure 1 - BISACS Overview Diagram

User Verification Process

The user verification process involves authenticating the user as well as authorizing user privileges along with authenticating an official certificate of authenticity (i.e., an electronic x509 digital certificate [3]). The authentication part of the user verification process requires verifying the user's identifier and password along with verifying a certificate of authenticity. The authorization part of the user verification process supplies the user's access privileges for the system and currently has not been implemented. All parts of the user verification process will be supported by the BISACS.

Certificate Authentication Process

When authenticating a user or the work station that the user uses to log into the BISACS, this process requires authenticating a certificate of authenticity, i.e., the login process requires not only the user ID and password but also some form of official certificate of authenticity. Officially signed x509 certificates for our server and client applications were obtained from Thawte (<http://www.thawte.com>). Using the Eclipse extensible development platform, the CertificateValidator server class was created to validate these certificates. The certificate authentication process involves the following steps (see Figure 2 below):

- 1) The client connects to the CertificateValidator's socket port, exchanges the Transport Layer Security (TLS) handshake by sending the client's certificate that is signed by a trusted Certificate Authority (CA) such as Thawte and Verisign.
- 2) The client and server exchange public keys, then the CertificateValidator server validates that the client's public key signature and the client's alias match with one of the signatures and corresponding aliases that are stored in the CertificateValidator's keystore. A keystore file is a key database file that contains both public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. The keys are used for a variety of purposes, including authentication and data integrity.
- 3) The CertificateValidator's keystore contains all valid client certificates, if the client's certificate is not stored in the CertificateValidator's keystore, then the client is considered invalid. The CertificateValidator server receives the client's alias as part of the certificate authentication process, the CertificateValidator server then does a lookup in its keystore for a match, if there is a match for the client's alias in the keystore, the CertificateValidator server then retrieves the public key signature that it has for that client. If the public key signature that the CertificateValidator server retrieved from its keystore does not match with the incoming client's public key signature, then the string "InvalidCertificate" will be returned. If the public key signature that the CertificateValidator server retrieved from its keystore does match with the incoming client's public key signature, then an encrypted security key will be returned. It is up to the client software to process the string returned by the CertificateValidator's class accordingly.

Using the CertificateValidator server for certificate authentication allows the client software to validate any x509 certificates that are properly stored in the CertificateValidator's keystore. This allows client applications to validate certificates for people as well as for any objects such as devices.

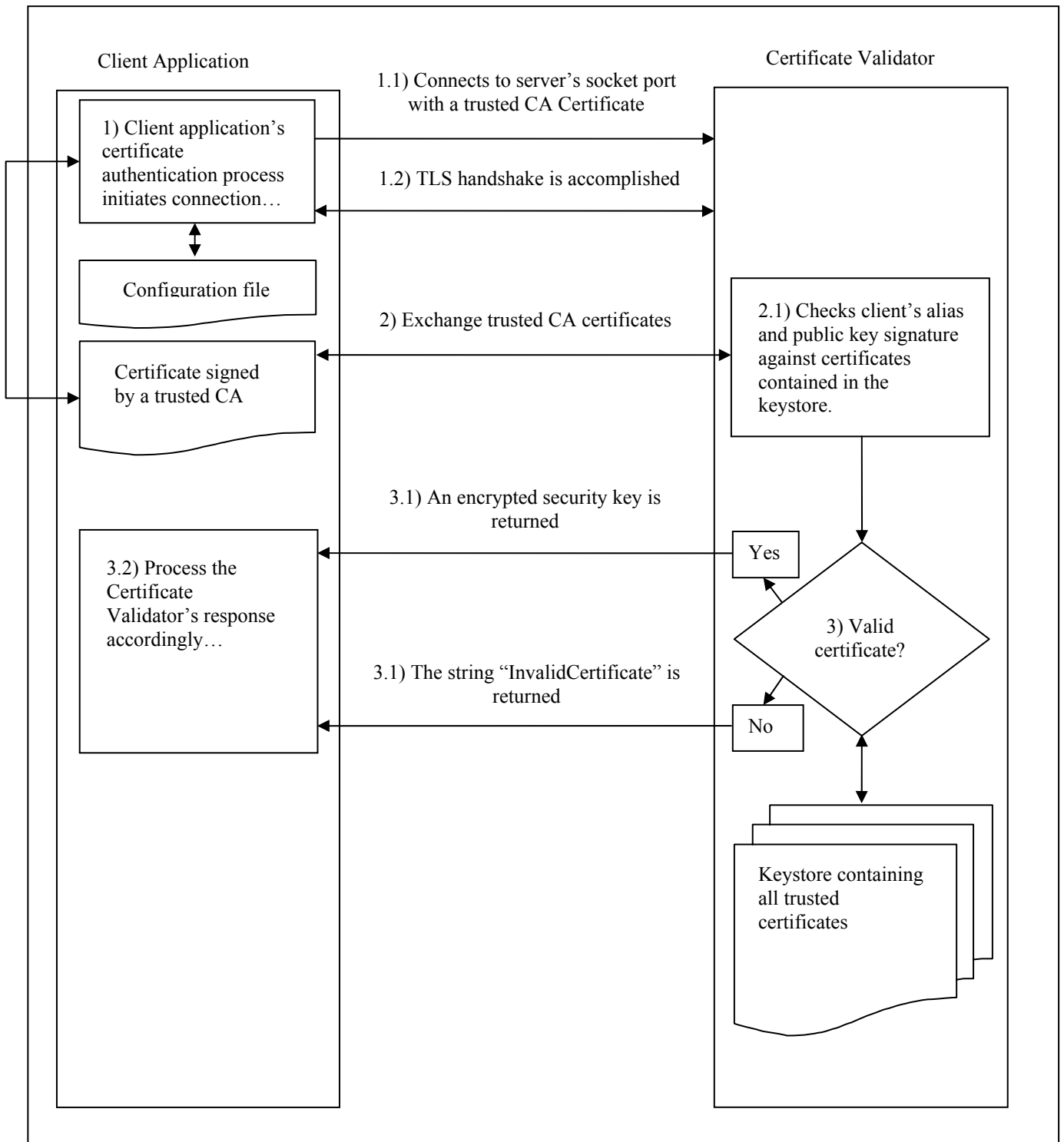


Figure 2 - Certificate Authentication Process

User Authentication Process

The user authentication process requires the system to store information about the user such as the user ID, and the password and user privileges. A data store is required in order to store users' information. We tested a Lightweight Directory Access Protocol (LDAP) server as well as a Database Management System (DBMS) to store the user data. OpenLDAP was chosen as the LDAP Server and MySql was chosen as the DBMS. For the purposes of the User Authentication Process, only the LDAP Server or the DBMS was needed as the data store, i.e., both data store mechanisms were not concurrently required for the user authentication process. We opted to use the MySql DBMS for the data store.

Using the Eclipse extensible development platform (<http://www.eclipse.org/>), the BisacsPrimaryPortal class was created to do user authentication. Using the Eclipse Web Tools Platform (WTP, <http://www.eclipse.org/webtools/>), the BisacsPrimaryPortal class' user authentication method was described using the Web Services Description Language (WSDL) and stored in a file, this file will be referred to as its WSDL. The WSDL can be used to publish the BisacsPrimaryPortal Web Services methods to a Universal Description, Discovery and Integration (UDDI) registry, the UDDI subject will be discussed in more details later in this document. Information about UDDI can be found at http://www.uddi.org/pubs/uddi_v3.htm [4]. Information about WSDL can be found at <http://www.w3.org/TR/wsdl> and <http://www.w3.org/TR/wsdl20/> [5].

Using the WTP, the BisacsPrimaryPortal user authentication method was ported to our application server as a Web Service. The Apache Tomcat Product was chosen as our Application Server (<http://tomcat.apache.org/>). The BisacsPrimaryPortal user authentication method can be accessed programmatically by using the correct endpoint, e.g., <https://seurat.cbt.nist.gov:8443/BisacsRpcWebPortal/services/BisacsPrimaryPortal>. The WSDL for the BisacsPrimaryPortal Web Services can be accessed using a browser by using the correct address, e.g., <https://seurat.cbt.nist.gov:8443/BisacsRpcWebPortal/services/BisacsPrimaryPortal?wsdl>.

Once the BisacsPrimaryPortal user authentication method is made available via a web service endpoint, i.e., supported by the Application Server, the User Authentication Process for a client application involves the following steps (see figure 3 below):

- 1) The client application connects to the BisacsPrimaryPortal endpoint such as the following address <https://seurat.cbt.nist.gov:8443/BisacsRpcWebPortal/services/BisacsPrimaryPortal>; it then sends the security key, the user identifier string and the user's password to the verifyUser method as described by the BisacsPrimaryPortal WSDL. The security key obtained from the Certificate Authentication process is used as the "info" parameter and this security key is checked for validity before the user identifier and password is authenticated. Notice that the endpoint address is a secured address using TLS so that all communications with the endpoint's Uniform Resource Locator (URL) are encrypted.
- 2) The BisacsPrimaryPortal verifyUser method takes the input parameters, validates the security key then does a search using its data store to authenticate the user identifier and password. If the security key is invalid, the string "ServiceError" is returned. The data store currently supported is the MySql DBMS.
- 3) If the user identifier exists in the BisacsPrimaryPortal's data store and its password matches the incoming request's password, then the verifyUser method will return an array of strings with the first string in the array containing the word "OK". If the user identifier does not exist in the BisacsPrimaryPortal's data store or if the incoming password does not match the data store's password for that user identifier, then the verifyUser method will return an array of strings with the first string in the array containing the word "noDataFound". It is up to the client software to process the string array returned by the verifyUser method accordingly.

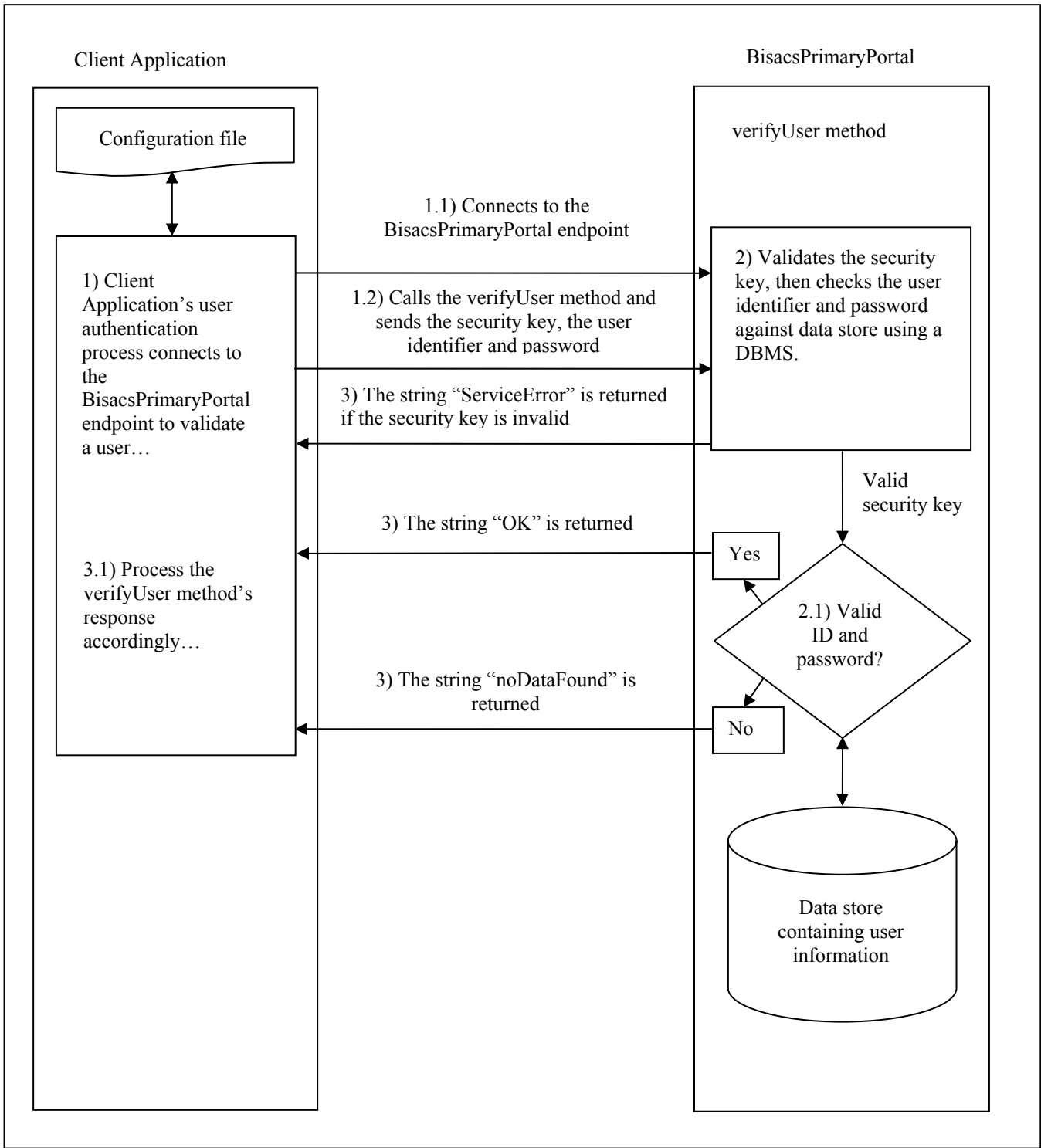


Figure 3 - User Authentication Process

BISACS Web Services Interfaces

The current specification for communicating with the BISACS Web Services Interfaces, discussed in this section, will continue to evolve. The supported BISACS Web Services Interfaces (indicated with a number next to them), their inputs and responses are specified below; they all return an array of strings for their results:

1) `String[] verifyUser(String userID, String passWord, String info)`

Input:

userID – the user ID

passWord – the corresponding password for the userID

info – reserved for future use, perhaps certificate information

Response:

The first item in the returned string array contains the status. If there is any data, it is contained in the second item of the returned string array:

Status:

- “ServiceError” - the security key passed in as the “info” parameter is invalid. This is a negative response.
- “OK” - the userID and its related passWord are verified.
- “noDataFound” - the userID and/or passWord fields are not recognized. If an internal error has occurred, then this is the default response. This status string is used as a negative verification.

2) `String[] processRequest(String request)`

Input:

request – the request or command needing action, valid requests are as follows:

“alarmStatus” or “alarmStatusDelimited” – both requests are for all active alerts in the system

“certificateValidatorInfo” – request for the Certificate Validator’s information

Response:

For “alarmStatus” and “alarmStatusDelimited”:

The status is the first item of the returned array and should be checked:

- “serviceError” – can not service this request at this time.
- “noDataFound” - there are no active alerts in the system.
- For “alarmStatus”, if the first item in the returned string array is not equal to the string “noDataFound” or the string “serviceError”, then the whole returned array of strings contains strings describing all active alerts in the system. At each index in the string array, the string represents an alert’s information in the form of a Common Alerting Protocol message (in its XML format). Refer to the Common Alerting Protocol, v.1.1 document for more information including parameter details and valid parameter values [2].

- For “alarmStatusDelimited”, if the first item in the returned string array is not equal to the string “noDataFound” or the string “serviceError”, then the whole returned array of strings contains strings describing all active alerts in the system. At each index in the string array, the string represents an alert’s information in a delimited form. The delimiter used is the string “!:!” and each alert string conforms to the following format:

“source!;!sender!;!category!;!event!;!urgency!;!severity!;!certainty!;!color!;!timeToLive!;!eventTime”

Where:

- *source* is the alert’s unique identifier coming from each Services Interface
- *sender* is the UR of the host that sent the alert
- *category*, see the CAP v.1.1 document
- *event* is used as the sub-category for the alert
- *urgency*, see the CAP v.1.1 document
- *severity*, see the CAP v.1.1 document
- *certainty*, see the CAP v.1.1 document
- *color* is for visual HTML/user indicator purposes
- *timeToLive* is the time in minutes this alert will live before being automatically deleted by the system
- *eventTime* is the number of milliseconds since January 1, 1970, 00:00:00 GMT

For “certificateValidatorInfo”:

The status is the first item of the returned array and should be checked:

- “serviceError” – can not service this request at this time.
- “OK” – Data available about the Certificate Validator is contained in items 2 and 3 of the array.

Array[1] – Certificate Validator’s machine/domain name, e.g., “my.bbs.state.gov”

Array[2] – The socket port the Certificate Validator is listening on, e.g., “6222”

3) String[] processRequestArray(String[] request)

Input:

request – the request or command array needing action. The request/command is contained in the first item of the string array. Valid requests are as follows:

“addAlarm” –

Add/update an active alert in the memory resident database. Note that the parameters for this command will change in future implementations where a CAP message will be used instead. The second item in the request string array must contain the unique alert identifier. For the rest of the required parameters, please see the section for “Common Alerting Protocol (CAP) Parameters” below. For an example of this request please see the “Examples for the processRequestArray() Method” section below.

“deleteAlarm” –

Delete the specified alert from the memory resident database. The second item in the request string array must contain the unique alert identifier. For an example of this request please see the “Examples for the processRequestArray() Method” section below.

“getAlarm” or “getAlarmDelimited” –

Get the specified alert’s information from the memory resident database. The second item in the request string array must contain the unique alert identifier. For an example of this request please see the “Examples for the processRequestArray() Method” section below.

Response:

General response, the status is contained in the first item of the returned string array:

- “unknownRequest” – the incoming request/command is not supported. The second item in the returned string array contains a message echoing the incoming request.
- “serviceError” – the incoming request/command can not be serviced. The second item in the returned string array contains the related error message.

For “addAlarm”:

The status is the first item of the returned array and should be checked:

- “serviceError” – invalid number of parameters received.
- “OK” – the request was successfully processed.

For “deleteAlarm”:

The status is the first item of the returned array and should be checked:

- “serviceError” – invalid number of parameters received.
- “OK” – the request was successfully processed.

For “getAlarm” and “getAlarmDelimited”:

The status is the first item of the returned array and should be checked:

- “serviceError” – invalid number of parameters received.
- “noDataFound” – alert’s unique identifier is not contained in the database.

- “OK” – the request was successfully processed.
 - o For “getAlarm”, the alert’s information is contained in the second item of the returned string array in the form of a Common Alerting Protocol message (in its XML format). Refer to the Common Alerting Protocol, v.1.1 document for more information including parameter details and valid parameter values.
 - o For “getAlarmDelimited”, the alert’s information is contained in the second item of the returned string array. This string represents an alert’s information in a delimited form. The delimiter used is the string “!:!” and each alert string conforms to the following format:

“source!;!sender!;!category!;!event!;!urgency!;!severity!;!certainty!;!color!;!timeToLive!;!eventTime”

Where:

- *source* is the alert’s unique identifier coming from each Services Interface
- *sender* is the URL of the host that sent the alert
- *category*, see the CAP v.1.1 document
- *event* is used as the sub-category for the alert
- *urgency*, see the CAP v.1.1 document
- *severity*, see the CAP v.1.1 document
- *certainty*, see the CAP v.1.1 document
- *color* is for visual HTML/user indicator purposes
- *timeToLive* is the time in minutes this alert will live before being automatically deleted by the system
- *eventTime* is the number of milliseconds since January 1, 1970, 00:00:00 GMT

Common Alerting Protocol (CAP) Parameters

This section applies to the “addAlarm” command for the interface method processRequestArray. Future implementation for the parameters to this command will involve using a CAP message in lieu of the following which is currently being used. The string “!:!” is used as the delimiter between the parameters. Refer to the Common Alerting Protocol, v.1.1 document [2] for parameter details and for their valid values. The “addAlarm” request requires the following parameters for their respective elements within the CAP message:

Alert Element:

- status** – this is the status for the alert.
Example: “status!:!Exercise”
- scope** – this is the scope for the alert.
Example: “scope!:!Public”
- msgType** – this is the message type identifier.
Example: “msgType!:!Alert”
- source** – this is the unique string identifying the device that created the alert.
Example: “source!:!alarm1bundle.sensor01”

Info Element:

- category** – *Example: “category!:!Fire”
- event** – describes the alert sub-category
Example: “event!:!Smoke”
- urgency** – *Example: “urgency!:!Immediate”
- severity** – *Example: “severity!:!Extreme”
- certainty** – *Example: “certainty!:!Observed”
- description** – free form text describing the alert
Example: “description!:!Smoke detector, building 101, 5th flr, room B510”
- timeToLive** – time in minutes for this alert to live, valid range is 1-20160
Example: “timeToLive!:!15”

Resource Element:

Not currently used.

Area Element:

Not currently used.

* See the CAP v.1.1 document for valid values [2]

Examples for the processRequestArray() Method

Using the sample data from the “Common Alerting Protocol (CAP) Parameters” section above, examples of the requests that can be used for the web services processRequestArray() method are listed below:

addAlarm request:

Array index	Array content
1)	“addAlarm”
2)	“source!::!alarm1bundle.SmokeDetector01”
3)	“status!::!Exercise”
4)	“scope!::!Public”
5)	“msgType!::!Alert”
6)	“category!::!Fire”
7)	“event!::!Smoke”
8)	“urgency!::!Immediate”
9)	“severity!::!Extreme”
10)	“certainty!::!Observed”
11)	“description!::!Fire detector, building 101, 5 th floor, room B510”
12)	“timeToLive!::!15”

deleteAlarm request:

Array index	Array content
1)	“deleteAlarm”
2)	“alarm1bundle.SmokeDetector01”

getAlarm request:

Array index	Array content
1)	“getAlarm”
2)	“alarm1bundle.SmokeDetector01”

getAlarmDelimited request:

Array index	Array content
1)	“getAlarmDelimited”
2)	“alarm1bundle.SmokeDetector01”

Commands and Data Exchange

The above Web Services Interfaces allow for future implementation of any type of commands/requests and responses that are text based. The current implementation only supports the commands listed above. Any unrecognized commands will result in an array of strings with the first item in the array indicating a bad status, i.e., the first string in the returned array of strings is the word “unknownRequest”.

Once the BisacsPrimaryPortal Web Services Interface methods are made available via a web service endpoint, i.e., supported by the Tomcat Application Server, the request/command/data exchange process for a client application involves the following generic steps (see figure 4 below):

- 1) The client application connects to the BisacsPrimaryPortal endpoint such as the following address <https://seurat.cbt.nist.gov:8443/BisacsRpcWebPortal/services/BisacsPrimaryPortal>; it then sends the request or command string to the appropriate method as described by the BisacsPrimaryPortal WSDL. Notice that the endpoint address is a secured address using TLS so that all communications will be encrypted between the applications.
- 2) The BisacsPrimaryPortal Web Services method receives the request/command from the client and does the appropriate work in order to generate a response in the form of an array of strings. The current implementation does not make use of the security key obtained from the Certificate Authentication process.
- 3) It is up to the client application to process the response containing the array of strings accordingly.

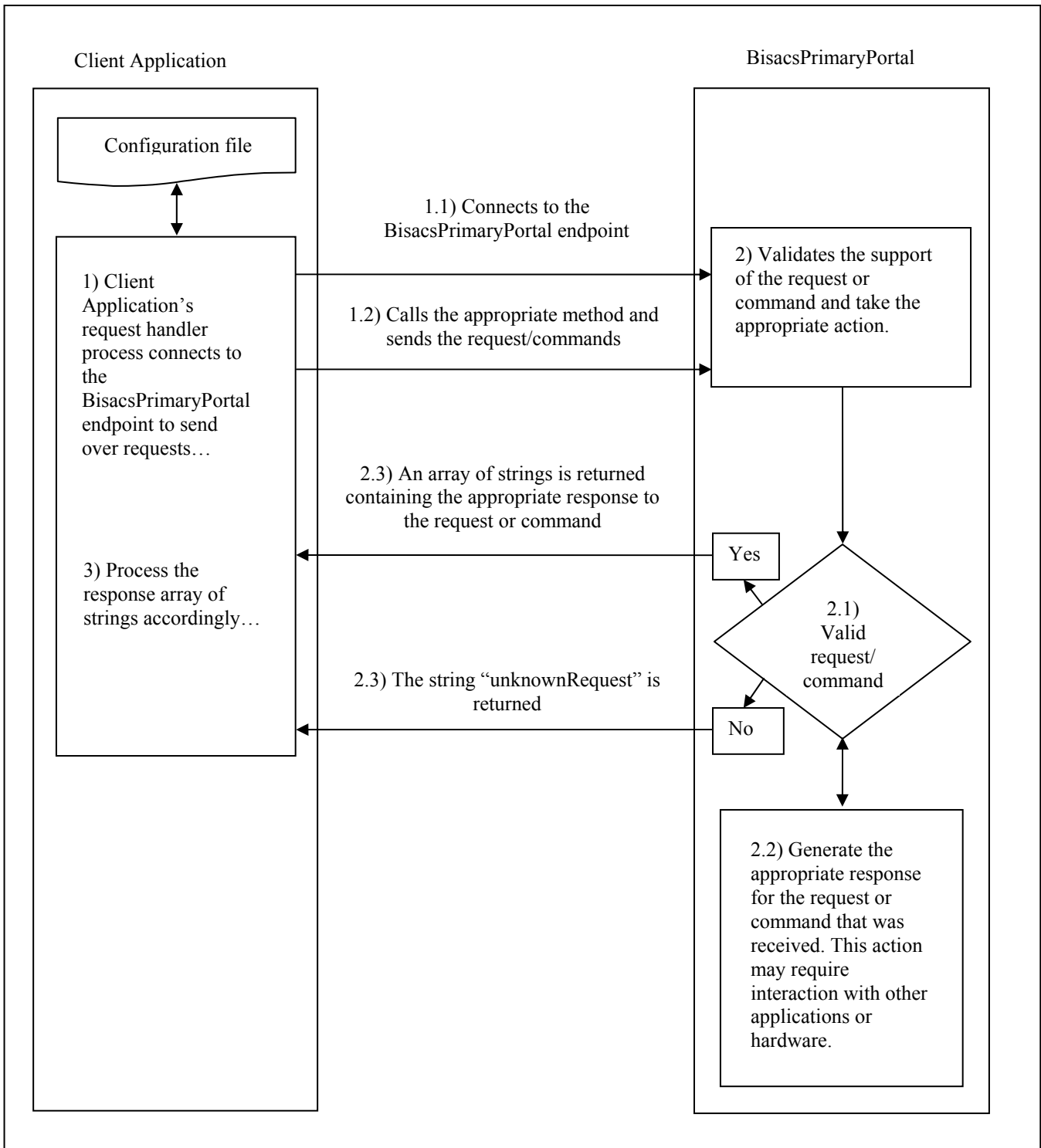
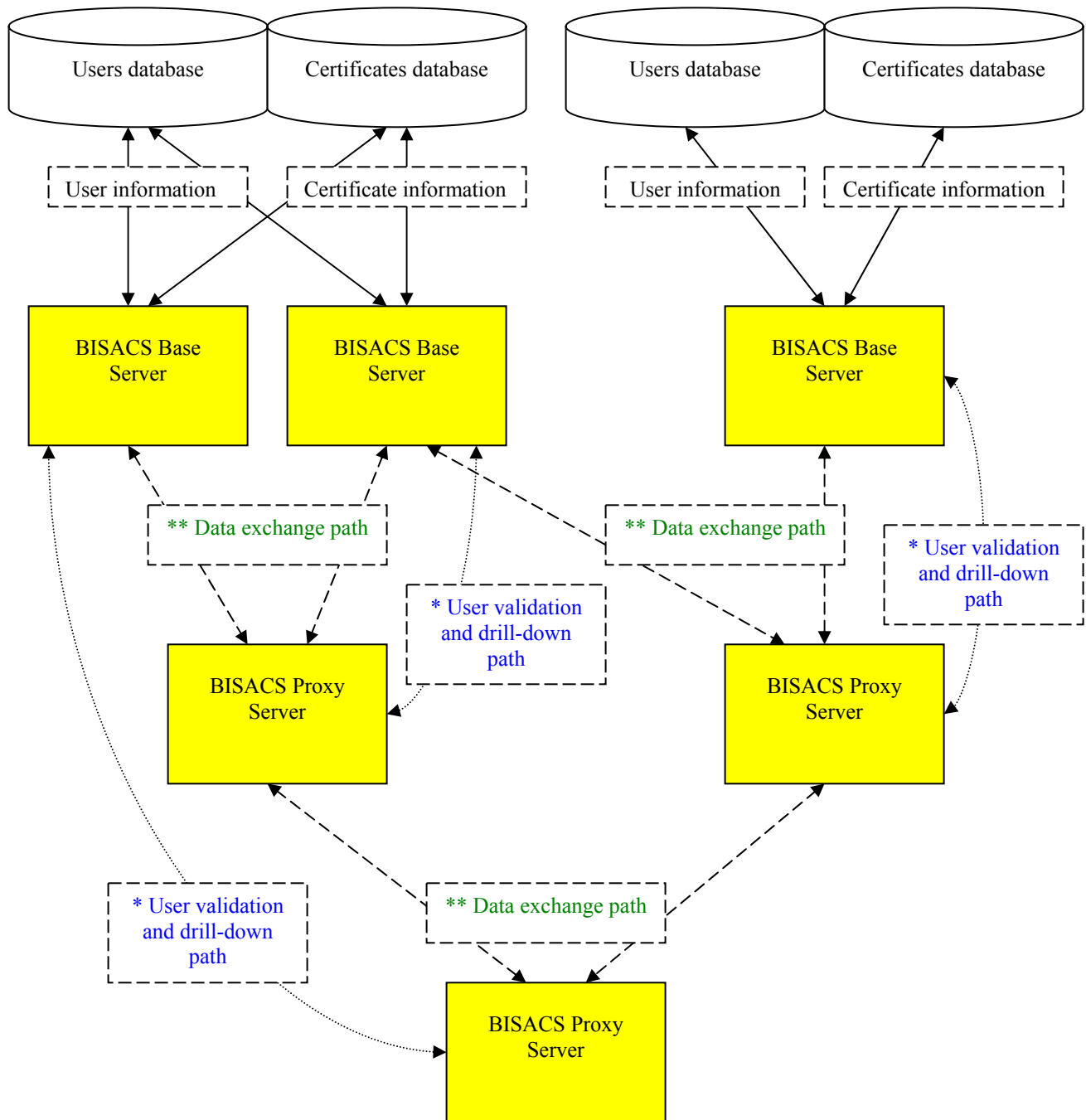


Figure 4 - Commands and Data Exchange Process

Communication and User Validation Data Flow

Figure 5 below shows the BISACS communication and user validation processes in greater detail. To gain access to the BISACS network, the user must connect and log into one of the nodes that is part of the BISACS network. Although the user authentication process may pass through a BISACS Proxy Server, the accessing of the data stores is actually done at the BISACS Base Server node. Notice that any drill down processing or commands/requests to access or control a device has to go through the BBS. The data stores containing the user information and certificate information may reside inside the firewall (i.e., behind the BBS) or outside of the firewall (i.e., externally at a remote location). The data stores may contain information for “internal” users such as employees of a building and/or “external” users such as first responders. The BISACS can be configured to handle the authentication of internal and external users, and it will maintain secure communication for each user session.



** The BISACS Proxy Servers act as active clients to other nodes/servers using the path with dashed lines.

* For user validation or drill-down functions, the BISACS Proxy Servers must contact the BISACS Base Servers directly using the path with dotted lines.

Notice that the BISACS Base Servers can be configured to validate users using a centralized user and certificate database combination, or they can use multiple user and certificate database combinations. The validation process requires both databases.

Security tokens are given out by the BISACS Base Servers as part of the user validation process and they can be used at any level for requesting information (e.g., read-only client mode). All device manipulations and commands can only be done at the BISACS Base Server level where user access control can be configured to be more restrictive.

Figure 5 - BISACS Communication and User Validation Data Flow Diagram

Alerts Monitoring

Data Gathering

One of the goals for the BISACS is to monitor devices for alerts within one or more buildings (or areas). At the lowest level in the BISACS network, alert notifications are sent to the BISACS Base Server (BBS) from devices or Services Interfaces by using the command “addAlarm” and calling the Web Services Interface method processRequestArray (refer to the section on BISACS Web Services Interfaces above). Each alert that is sent to the BBS has a corresponding lifespan known as the “timeToLive” attribute and its value represents time in minutes. The timeToLive attribute is there so that if the entity that reported the alert goes offline or dies, the reported alert would stay alive in the system for the specified amount of time. The protocol is for all entities that send alert notification to the BBS via the “addAlarm” requests, they should also send “deleteAlarm” requests to delete those alerts that are no longer active hence keeping the information “real time”.

Should the entity that sent an alert notification go down or get disconnected from the BBS, the timeToLive attribute will be used by the BBS to delete those alerts that have exceeded their lifespan. Note that the timeToLive attribute represents the time in minutes between the “sent” time of the alert and the “expires” time of the alert. The “sent” sub-element is part of the “alert” element in the CAP message and the “expires” sub-element is part of the “info” element of the CAP message. The BBS stores all alerts in its in-memory database for client applications such as the BPS to query.

Data Propagation

The BISACS Proxy Server (BPS) is designed to monitor one or more BBS and/or one or more BPS. Since the BISACS network is Web Services based, the BPS must act as an active client and polls all nodes that it monitors for alert information. All alerts retrieved by the BPS are stored in the BPS in-memory database. The BPS also supports requests for alert notifications from other clients; this capability allows the BISACS to build its network of servers. Currently all alert notifications are propagated from the BBS all the way up the BISACS network hierarchy, a mechanism for filtering the alert information by the BPS can be put in place but this mechanism is currently not implemented. The filtering of the alert notifications is currently being done in the client application itself. With proper configuration of the BISACS Base Servers and BISACS Proxy Servers, a hierarchical network of servers can be achieved as shown in figure 6 below.

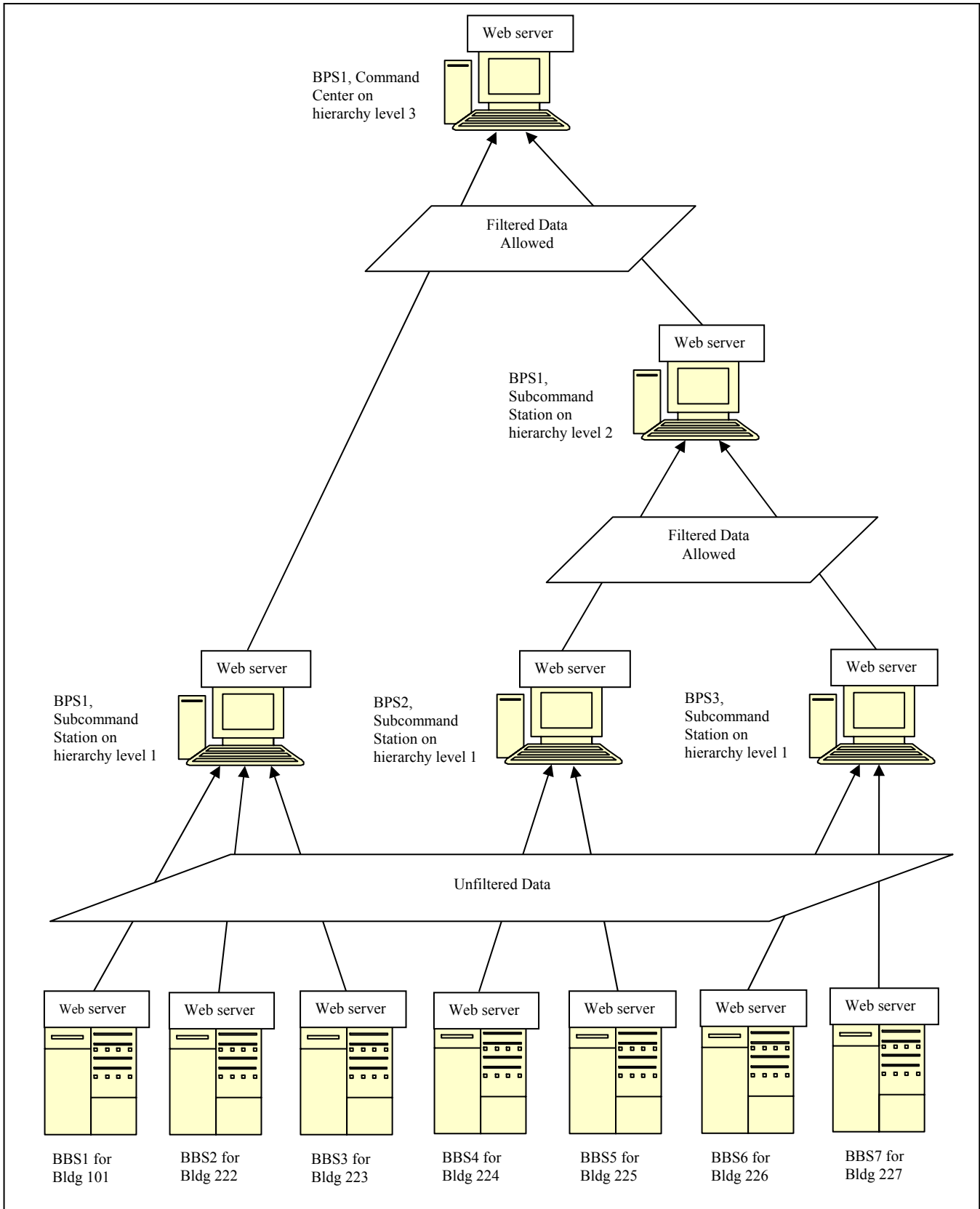


Figure 6 - BISACS Alert Notification Hierarchy

Integrating the Client Applications

Application Specific Client

Currently only the web application for the BISACS BBS and BPS nodes has been implemented; the web application can only monitor alert notifications and is discussed below in the “Web Client” section. An application specific client that can make use of certificates for authentication purposes, monitor alert notifications as well as send requests and/or commands to the BBS are in the plans for future development. The process for how the application specific client will interact with the BISACS to authenticate its certificate is described below.

- 1) The Application Specific Client (ASC) connects to the BisacsPrimaryPortal (Web Services portal), calls the processRequest method to get information on where the Certificate Validator resides.
- 2) It then connects to the appropriate Certificate Validator’s socket port to validate its certificate with the BISACS’s certificate keystore. If the certificate used by the ASC is invalid, an appropriate error status will be sent back, otherwise the calling application will receive a security key and the log in process continues to the next step.

By keeping the certificate authentication process as a separate step from the user verification process, allows for clustering, load balancing and a clean isolation of the certificate authentication process (see figure 7 below).

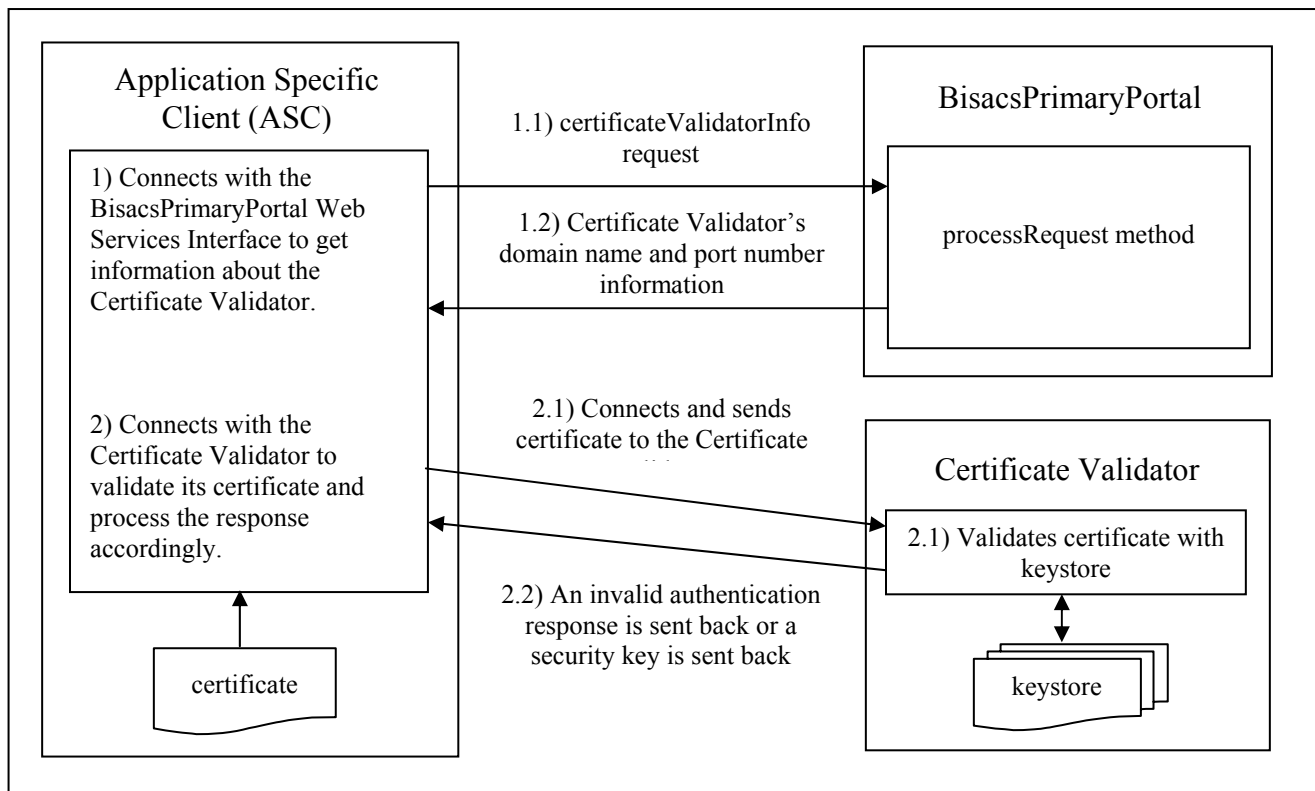


Figure 7 - Application Specific Client, Certificate Authentication Process

Once the certificate authentication process is successful and the Application Specific Client has obtained the security key:

- 1) The ASC connects to the BisacsPrimaryPortal (Web Services portal), calls the verifyUser method and proceeds by sending its user ID, password and security key for authentication purposes.
- 2) Depending on the information it receives, the verifyUser method will respond accordingly (see figure 8 below).

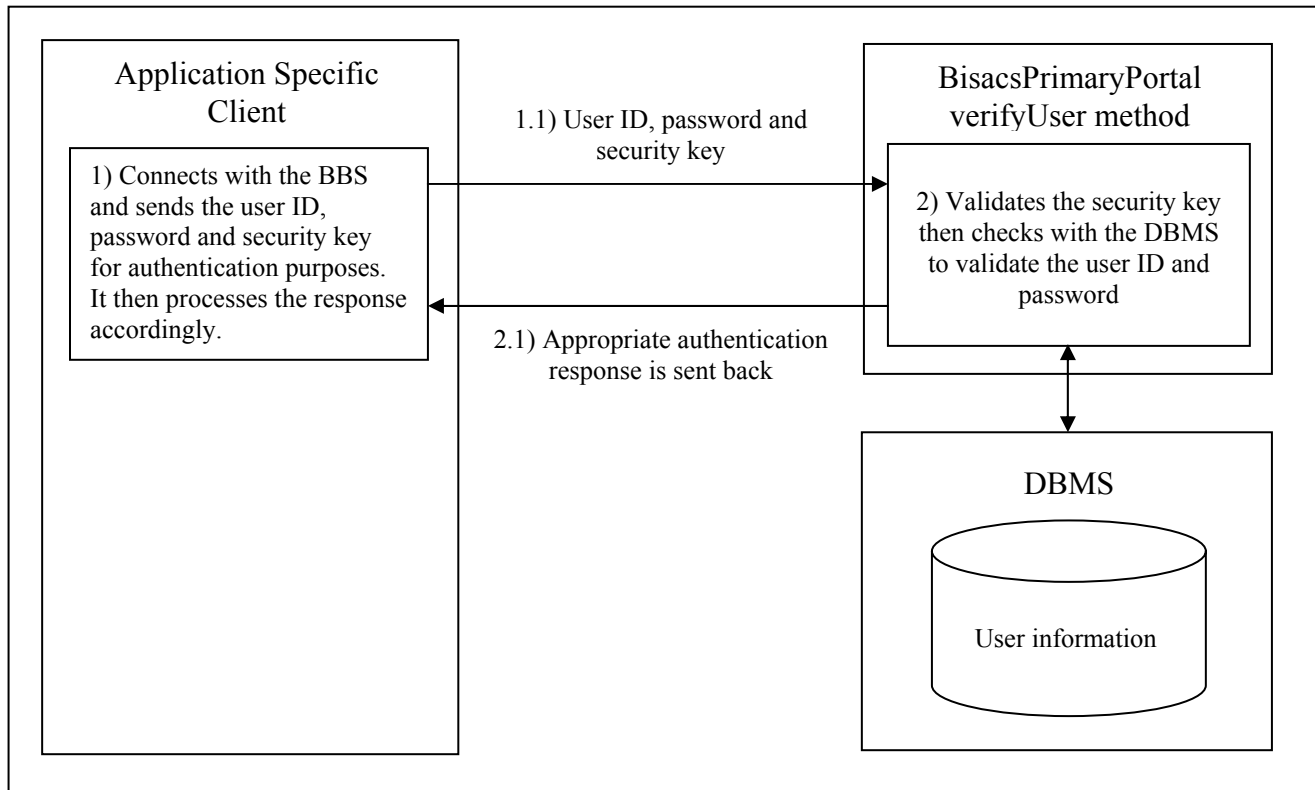


Figure 8 - Application Specific Client, User Authentication Process

Web Client and the User Authentication Process

Currently all BBS and BPS nodes contain a Web Client that allows users to log into those nodes and monitor all alerts that are stored in their in-memory databases. The Web Client also allows for filtering alert information, this will be discussed in more detail later.

Unlike the Application Specific Client, the Web Client does not support certificate validation (see figure 9 below):

- 1) It connects to the BBS BisacsPrimaryPortal (Web Services portal), calls the verifyUser method and proceeds by sending its user ID, password and its own security key for authentication purposes.
- 2) Depending on the information it receives, the verifyUser method will respond accordingly.

Since the Web Client can be accessed by using any web browser, restrictions on the source IP addresses may be implemented for security purposes in the future, currently all BISACS Web Services nodes (BBS & BPS) that are exposed on the Internet can be accessed by any web browser.

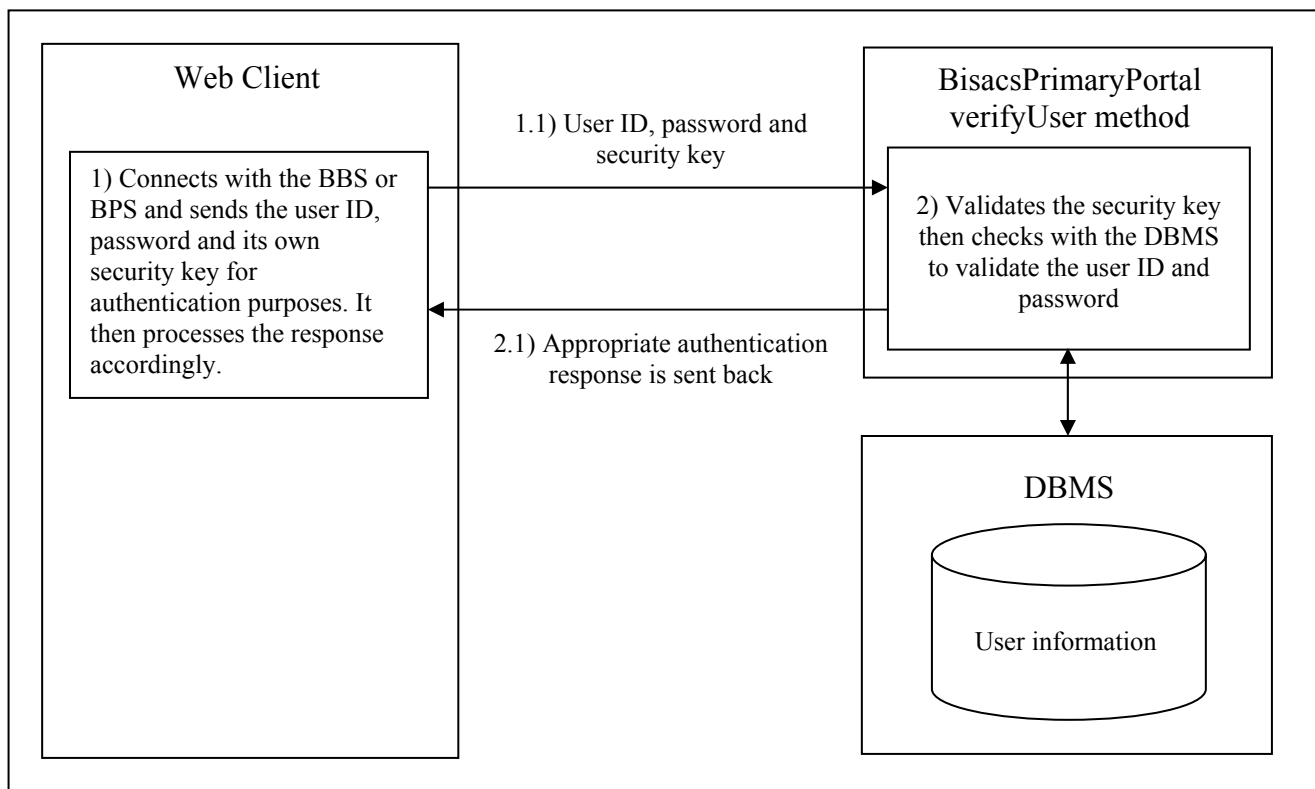


Figure 9 - Web Client, User Authentication Process

Web Client Screen Shots

Below, figure 10 shows the Web Client login screen and figure 11 shows the error message if an invalid user ID or password is used.

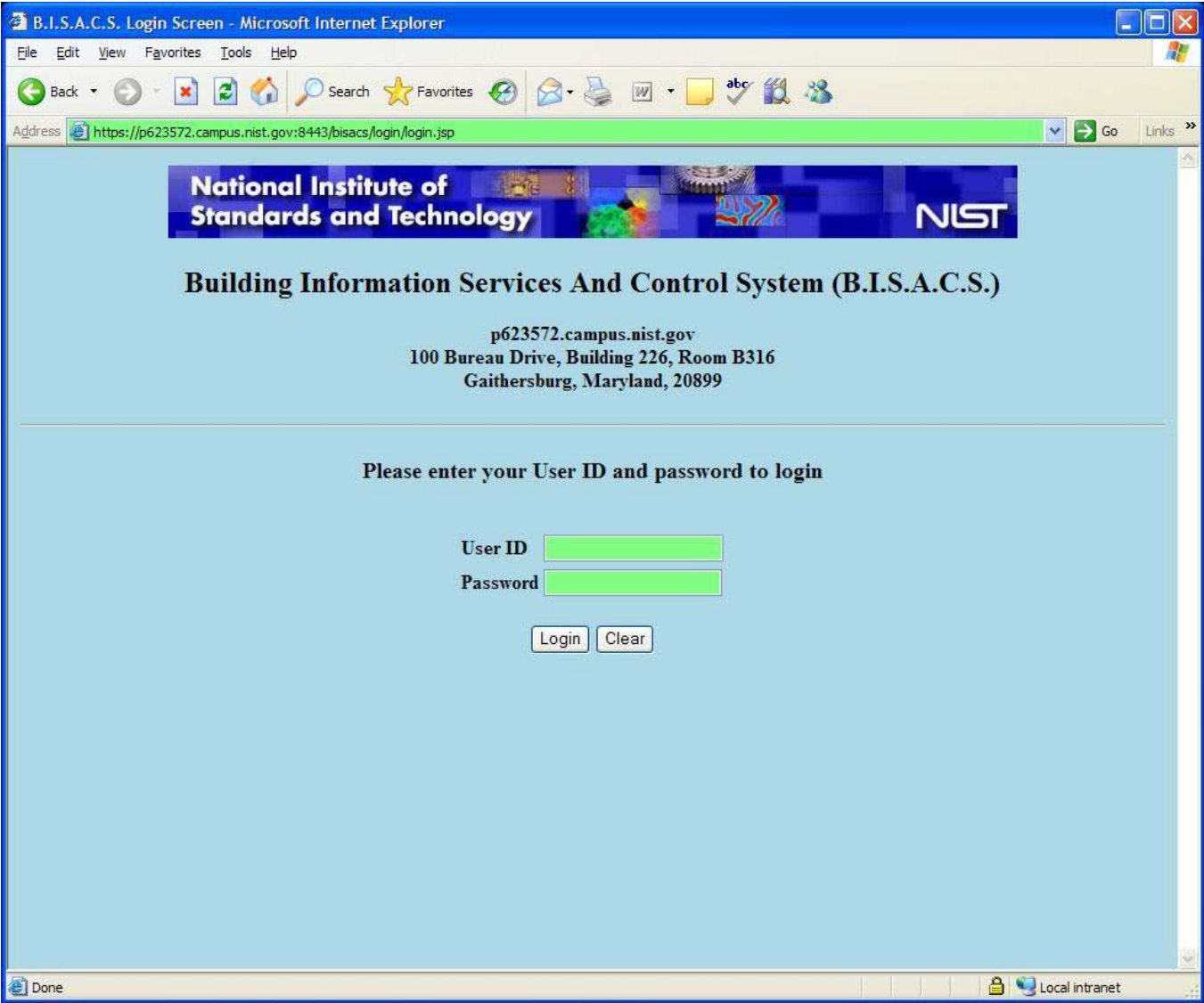


Figure 10 - Web Client, User Login Screen

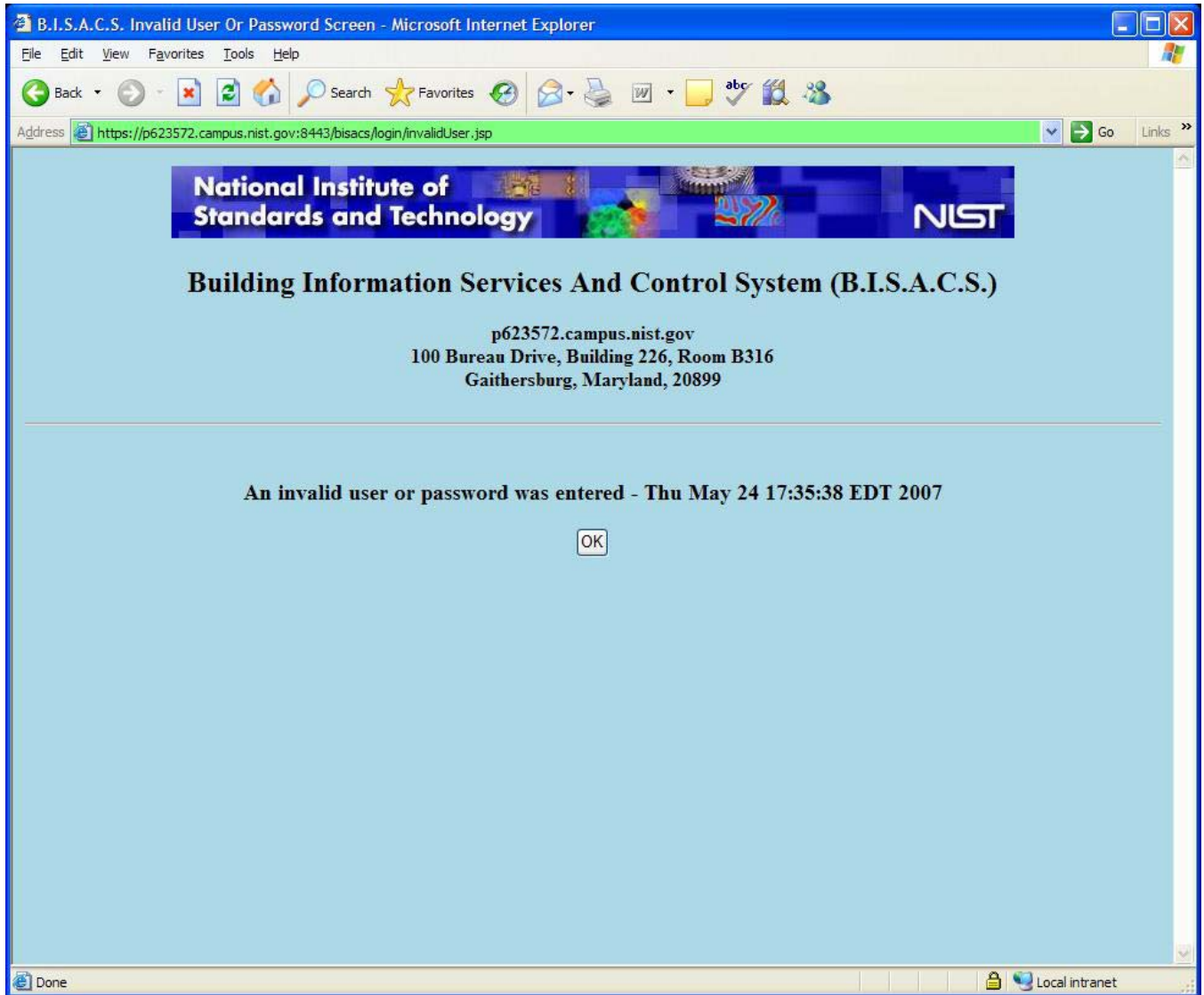


Figure 11 - Error Message for When an Invalid User ID or Password is Entered

Figure 12 shows the Main Screen once the user is successfully logged into the system.

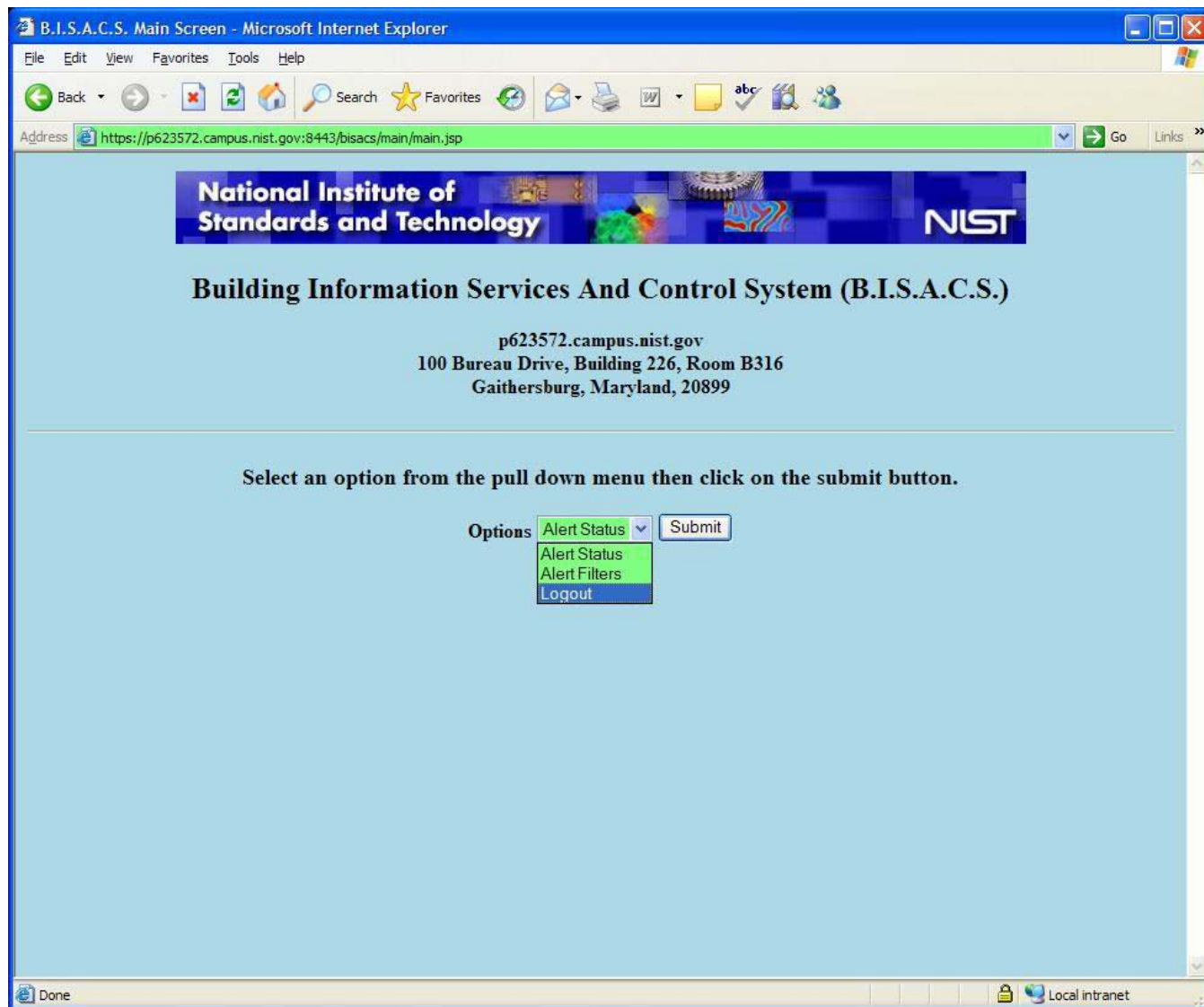


Figure 12 - The Main Options Screen

Figure 13 shows the Filters Screen. This screen allows the user to set up filters for monitoring specific information. The filter options are based on the data contained in the CAP messages/alerts that each node in the BISACS network contain, these filters allow the user to focus and display data of interest.

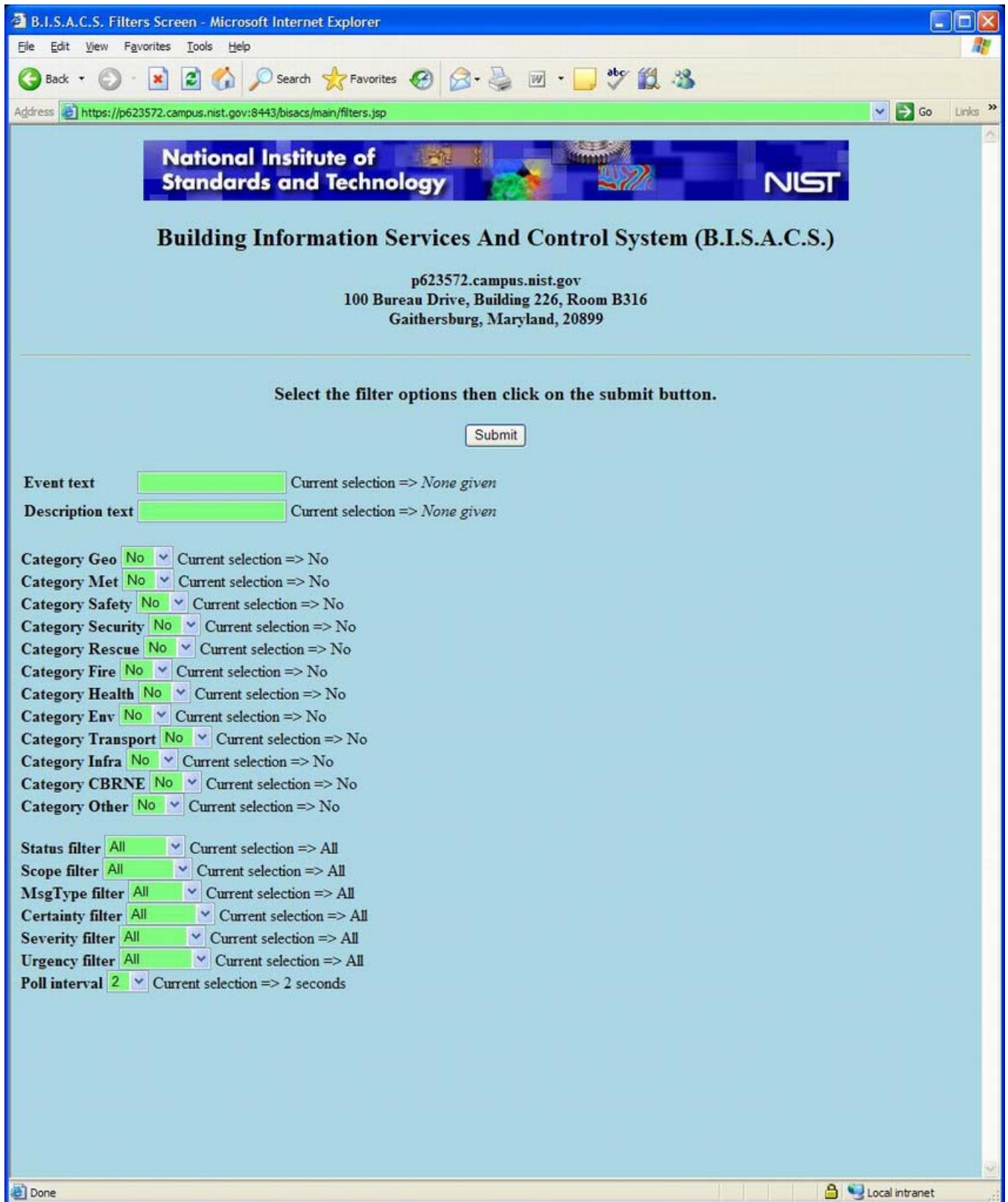


Figure 13 - The Filters Screen

Figure 14 shows an example of the CAP message that encapsulates the alert information contained at each node of the BISACS network [2]. The filter options specified by the user using the “Filters Screen” (see figure 13 above) are used to filter the data from a CAP message such as the CAP message shown below.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <alert xmlns="urn:oasis:names:tc:emergency:cap:1.1">
  <identifier>1179353147004</identifier>
  <sender>https://p623572.campus.nist.gov:8443/bisacs</sender>
  <sent>2007-05-16T18:05:47-04:00</sent>
  <status>Exercise</status>
  <msgType>Alert</msgType>
  <source>alarm1bundle.sensor01</source>
  <scope>Public</scope>
- <info>
  <category>Fire</category>
  <event>Smoke</event>
  <urgency>Immediate</urgency>
  <severity>Extreme</severity>
  <certainty>Observed</certainty>
  <expires>2007-05-16T18:06:47-04:00</expires>
  <description>Smoke detector, building 226, 3rd floor, room B346.</description>
</info>
</alert>
```

Figure 14 - Sample CAP Message

Choosing the "Alert Status" option from the main screen takes the user to the "Alert Status Screen". The figure below, Figure 15, shows that there are no alerts in the server's database. The filter parameters are indicated at the bottom of the screen.

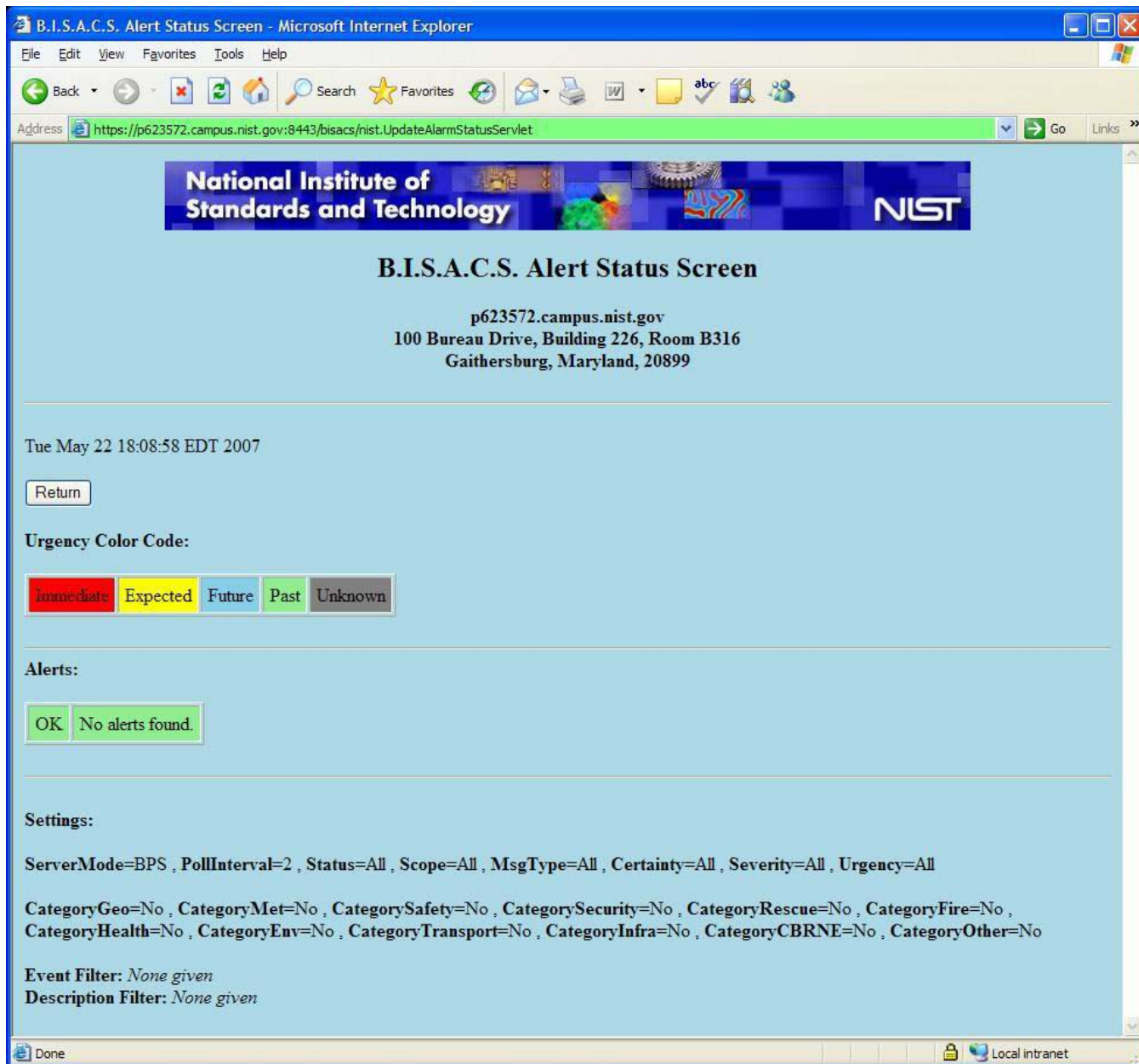


Figure 15 - The Alert Status Screen

Sample Configurations and Screen Shots of the Alerts

Figure 16 shows a configuration where the work station is monitoring a BISACS Base Server (BBS) and figure 17 shows the corresponding Alerts Screen.

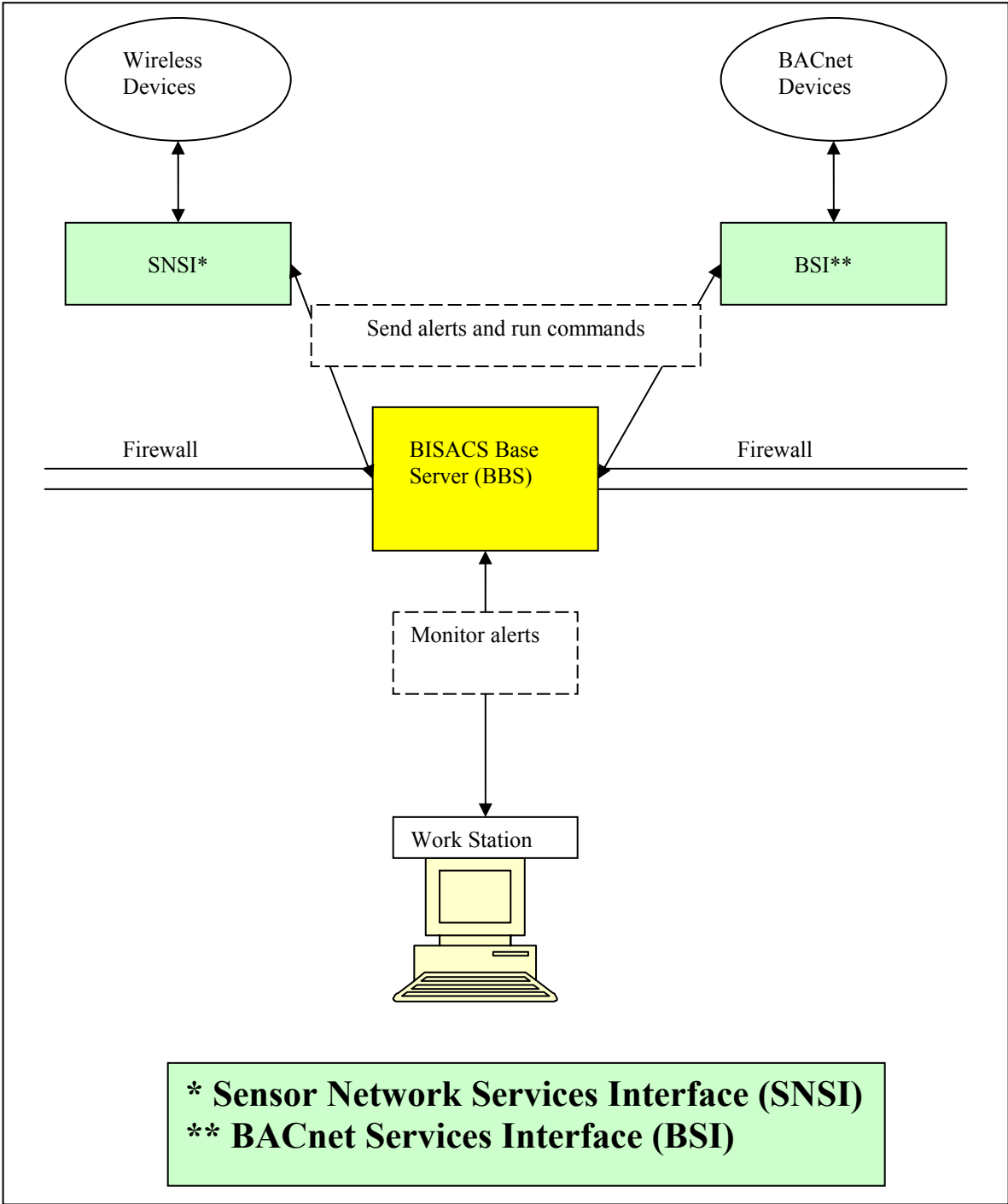


Figure 16 – A Work Station Monitoring a BISACS Base Server

Notice that the “Alert ID” column shows the alert identifiers without any trailing paths back to the originating BBS since for this configuration, we are monitoring the BBS where the alerts originated.


B.I.S.A.C.S. Alert Status Screen - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Search Favorites

Address <https://seurat.cbt.nist.gov:8443/bisacs/nist.UpdateAlarmStatusServlet>

Links Dictionary.com Merriam-Webster Online

National Institute of Standards and Technology 

B.I.S.A.C.S. Alert Status Screen

seurat.cbt.nist.gov
100 Bureau Drive, Building 226, Room B315
Gaithersburg, Maryland, 20899

Thu Oct 04 12:09:09 EDT 2007

Urgency Color Code:

Alerts:

Alert ID	Category	Event	Description	Event Time
radonDetector1	Safety	Gas	This is the radon detector in building 226, 2nd floor, hallway A, near room 226.	Thu Oct 04 12:09:06 EDT 2007
smokeDetector1	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 316.	Thu Oct 04 12:09:06 EDT 2007
smokeDetector2	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 2nd floor, hallway A, near room 255.	Thu Oct 04 12:09:06 EDT 2007
smokeDetector3	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 350.	Thu Oct 04 12:09:06 EDT 2007
temperatureSensor1	Other	Value:Temperature	72.5 degrees Fahrenheit	Thu Oct 04 12:09:06 EDT 2007

Settings:

ServerMode=BBS , PollInterval=2 , Status=All , Scope=All , MsgType=All , Certainty=All , Severity=All , Urgency=All

CategoryGeo=No , CategoryMet=No , CategorySafety=No , CategorySecurity=No , CategoryRescue=No , CategoryFire=No , CategoryHealth=No , CategoryEnv=No , CategoryCBRNE=No , CategoryOther=No

Figure 17 - The Alert Status Screen for Monitoring a BBS

Figure 18 shows a configuration where the work station is monitoring a BPS and figure 19 shows the corresponding Alerts Screen.

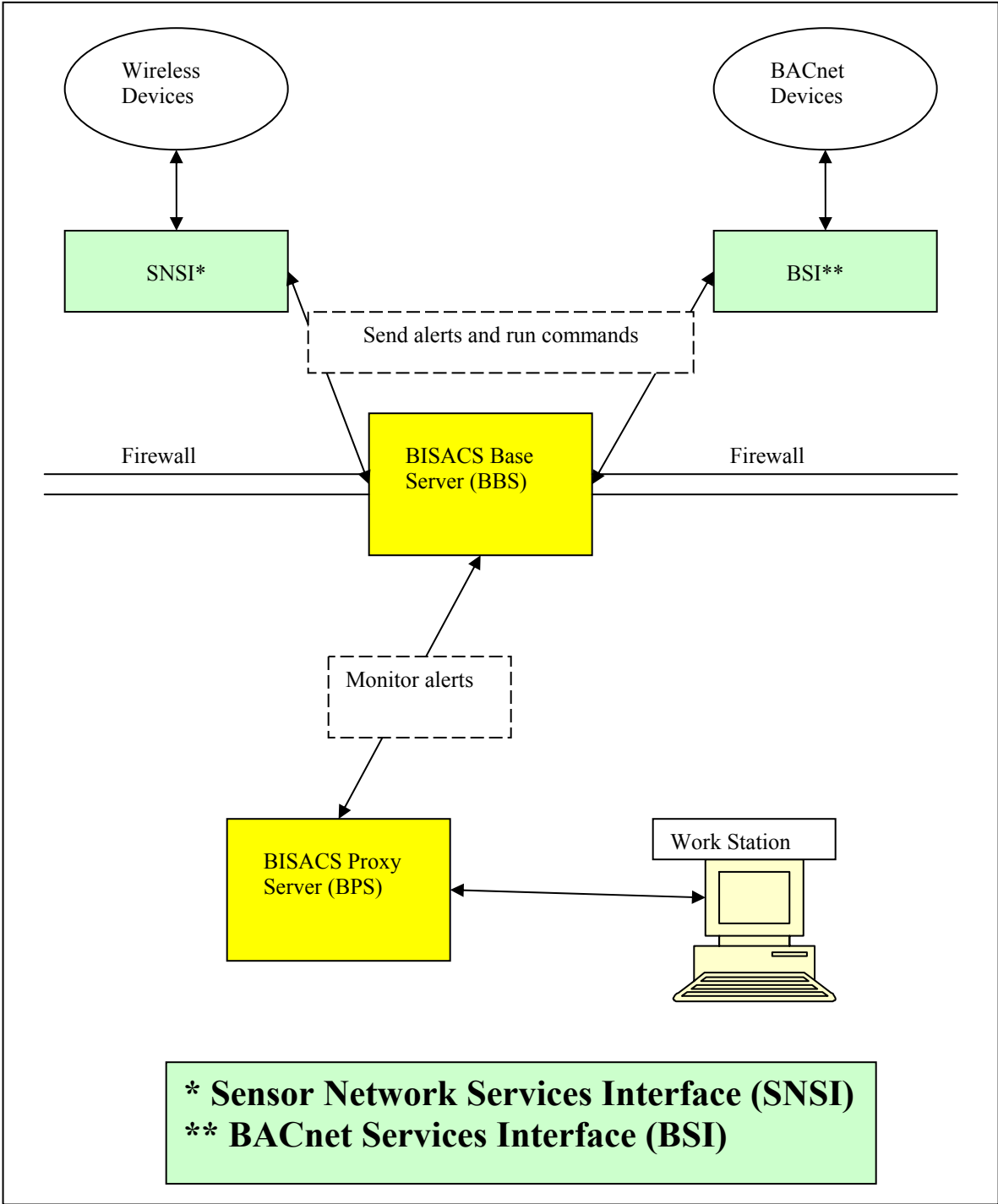


Figure 18 - A Work Station Monitoring a BISACS Proxy Server Residing 1 Level Up From a BBS

For this configuration, we are monitoring a BPS that resides one level above a BBS (see the figure above). Notice that the "Alert ID" column has the alert identifier with the character '@' separating the name of the server that the alert originated from, e.g., "smokeDetector3@seurat.cbt.nist.gov" where the server is "seurat.cbt.nist.gov". This layout informs the user that these alerts came from different nodes than the current proxy node since they have paths indicating the servers that they originated from.

Thu Oct 04 11:45:30 EDT 2007

[Return](#)

Urgency Color Code:

[Immediate](#) [Expected](#) [Future](#) [Past](#) [Unknown](#)

Alerts:

Alert ID	Category	Event	Description	Event Time
radonDetector1@seurat.cbt.nist.gov	Safety	Gas	This is the radon detector in building 226, 2nd floor, hallway A, near room 226.	Thu Oct 04 11:45:15 EDT 2007
smokeDetector2@seurat.cbt.nist.gov	Fire, Safety, Security	Smoke	This is the smoke detector in building 226, 2nd floor, hallway A, near room 255.	Thu Oct 04 11:45:15 EDT 2007
smokeDetector3@seurat.cbt.nist.gov	Fire, Safety, Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 350.	Thu Oct 04 11:45:15 EDT 2007
smokeDetector1@seurat.cbt.nist.gov	Fire, Safety, Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 316.	Thu Oct 04 11:45:16 EDT 2007
temperatureSensor1@seurat.cbt.nist.gov	Other	Value: Temperature	72.5 degrees Fahrenheit	Thu Oct 04 11:45:16 EDT 2007

Settings:

ServerMode=BPS, PollInterval=2, Status=All, Scope=All, MsgType=All, Certainty=All, Severity=All, Urgency=All

CategoryGeo=No, CategoryMet=No, CategorySafety=No, CategorySecurity=No, CategoryRescue=No, CategoryFire=No, CategoryHealth=No, CategoryEnv=No, CategoryTransport=N
 CategoryCBRNE=No, CategoryOther=No

Figure 19 - The Alert Status Screen for Monitoring a BPS Residing at 1 Level Up From a BBS

Figure 20 shows a configuration where the work station is monitoring a BPS that monitors a BBS and another BPS. Figure 21 shows the corresponding Alerts Screen.

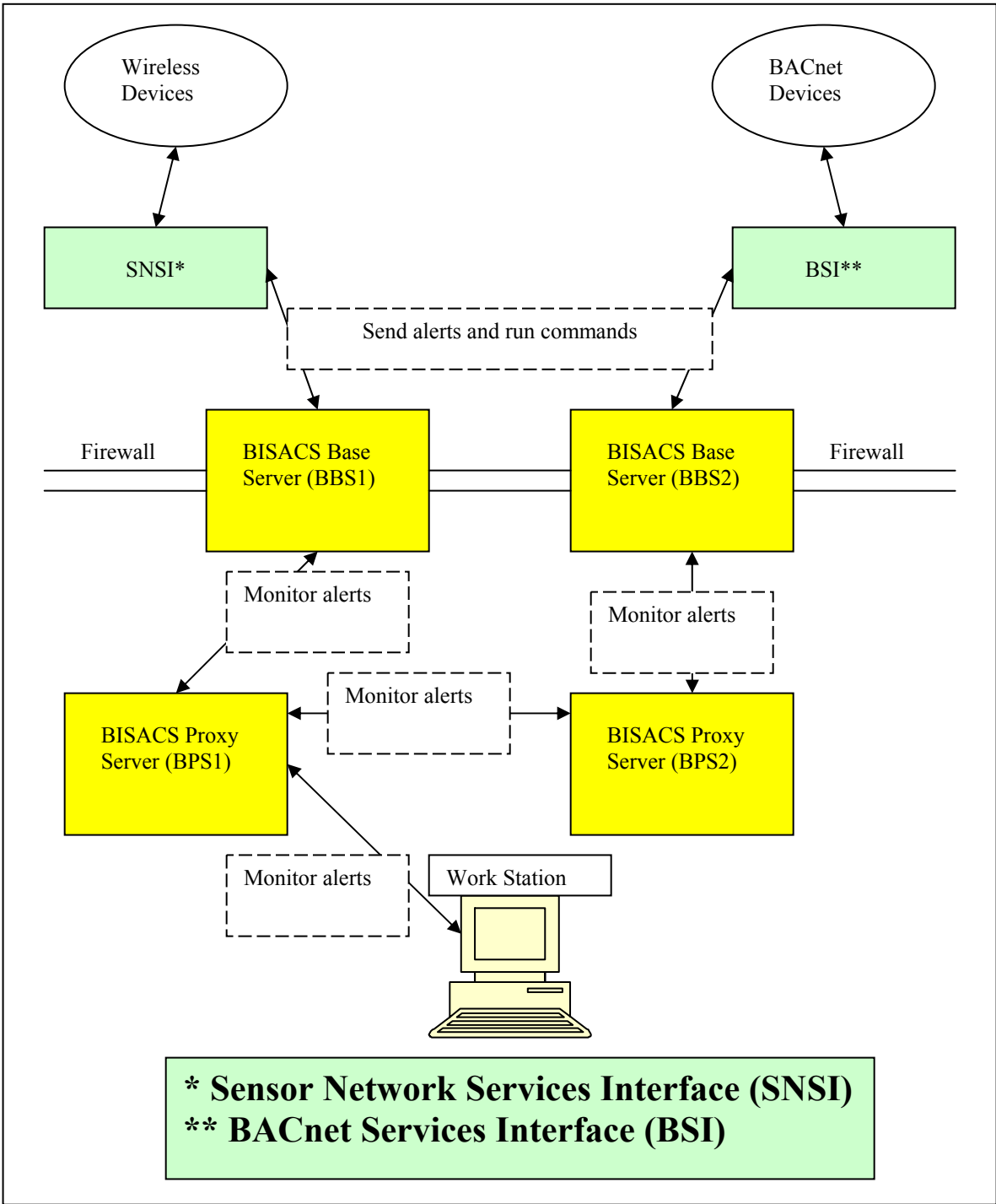


Figure 20 - A Work Station Monitoring a BPS That Monitors a BBS and another BPS

For this configuration, we are monitoring a BPS that monitors a BBS and another BPS (see the figure above). Notice that the "Alert ID" column has the alert identifier with the character '@' separating the name of all of the servers that the alerts had hopped through to get to the current node in the network hierarchy; i.e., some of those alerts went from node "seurat.cbt.nist.gov" (BBS2) to node "p623572.campus.nist.gov" (BPS2) to the current BPS node (BPS1) that is at level 2, e.g., "smokeDetector3@seurat.cbt.nist.gov@p623572.campus.nist.gov". For this example, notice that BBS1 is also "seurat.cbt.nist.gov".

National Institute of Standards and Technology **NIST**

B.I.S.A.C.S. Alert Status Screen

yz250f.nist.gov
100 Bureau Drive, Building 226, Room B316
Gaithersburg, Maryland, 20899

Thu Oct 04 12:01:18 EDT 2007

[Return](#)

Urgency Color Code:

Immediate Expected Future Past Unknown

Alerts:

Alert ID	Category	Event	Description	Event Time
radonDetector1@seurat.cbt.nist.gov	Safety	Gas	This is the radon detector in building 226, 2nd floor, hallway A, near room 226.	Thu Oct 04 12:00:07 EDT 2007
radonDetector1@seurat.cbt.nist.gov@p623572.campus.nist.gov	Safety	Gas	This is the radon detector in building 226, 2nd floor, hallway A, near room 226.	Thu Oct 04 12:00:07 EDT 2007
smokeDetector1@seurat.cbt.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 316.	Thu Oct 04 12:00:07 EDT 2007
smokeDetector1@seurat.cbt.nist.gov@p623572.campus.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 316.	Thu Oct 04 12:00:07 EDT 2007
smokeDetector2@seurat.cbt.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 2nd floor, hallway A, near room 255.	Thu Oct 04 12:00:07 EDT 2007
smokeDetector2@seurat.cbt.nist.gov@p623572.campus.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 2nd floor, hallway A, near room 255.	Thu Oct 04 12:00:07 EDT 2007
smokeDetector3@seurat.cbt.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 350.	Thu Oct 04 12:00:07 EDT 2007
smokeDetector3@seurat.cbt.nist.gov@p623572.campus.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 350.	Thu Oct 04 12:00:07 EDT 2007
temperatureSensor1@seurat.cbt.nist.gov	Other	Value:Temperature	72.5 degrees Fahrenheit	Thu Oct 04 12:00:07 EDT 2007
temperatureSensor1@seurat.cbt.nist.gov@p623572.campus.nist.gov	Other	Value:Temperature	72.5 degrees Fahrenheit	Thu Oct 04 12:00:07 EDT 2007

Figure 21 - The Alert Status Screen for Monitoring a BPS that Monitors a BBS and another BPS

Figure 22 shows a configuration where the work station is monitoring 2 BISACS Base Servers via a BPS. Figure 23 shows the corresponding Alerts Screen.

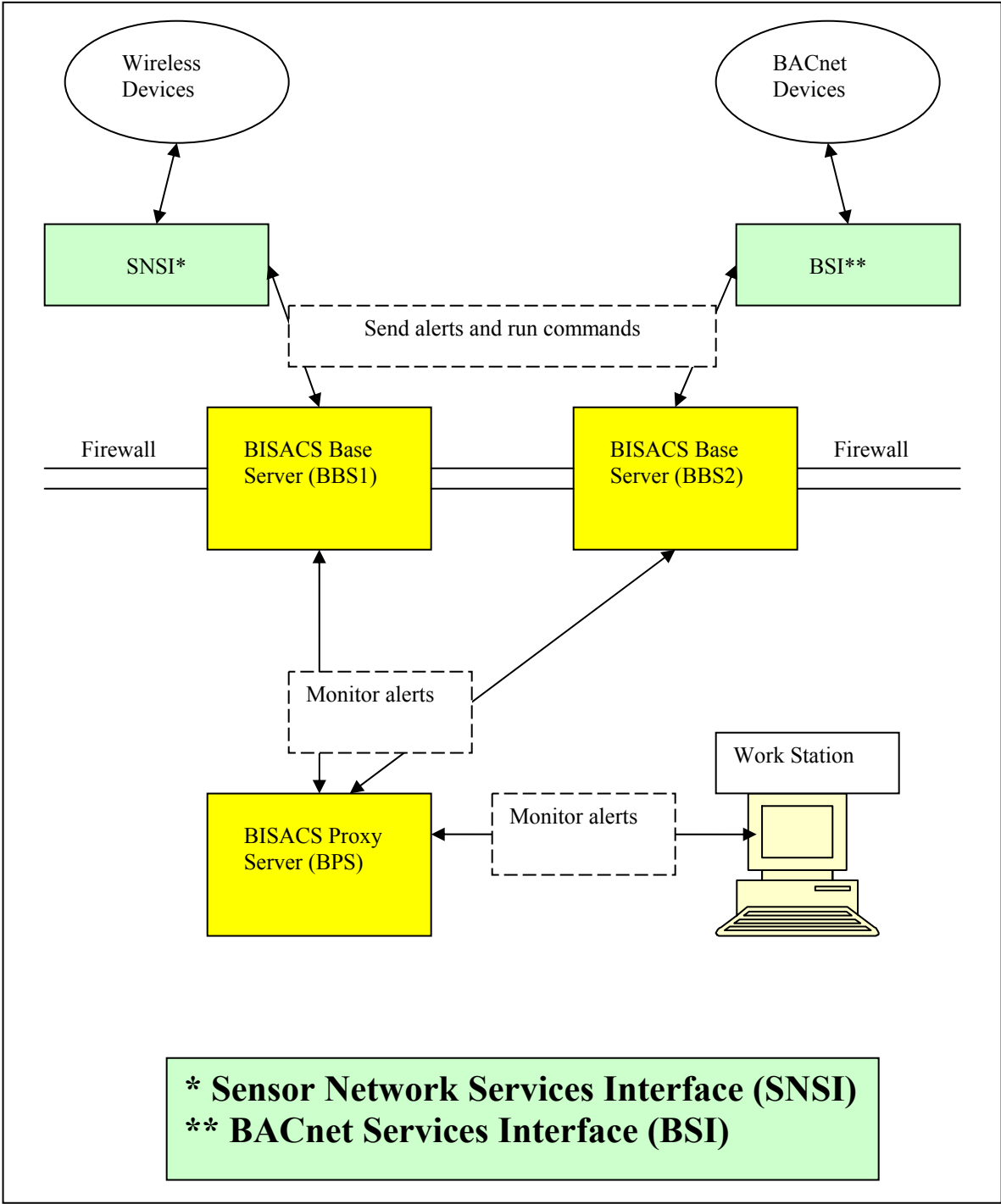


Figure 22 - A Work Station Monitoring 2 BISACS Base Servers via a BPS

For this configuration, we are monitoring a BPS that monitors 2 BISACS Base Servers (see the figure above). Notice that the "Alert ID" column has the alert identifier with the character '@' separating the name of the server that the alert originated from, e.g., "smokeDetector3@p623572.campus.nist.gov" where the server is "p623572.campus.nist.gov". This layout informs the user that these alerts came from different nodes than the current proxy node since they have paths indicating the servers that they originated from; i.e., some of those alerts came from node "seurat.cbt.nist.gov" and some of them came from node "p623572.campus.nist.gov".

National Institute of Standards and Technology **NIST**

B.I.S.A.C.S. Alert Status Screen
 yz250f.nist.gov
 100 Bureau Drive, Building 226, Room B316
 Gaithersburg, Maryland, 20899

Thu Oct 04 12:19:15 EDT 2007

[Return](#)

Urgency Color Code:
 Immediate Expected Future Past Unknown

Alerts:

Alert ID	Category	Event	Description	Event Time
radonDetector1@p623572.campus.nist.gov	Safety	Gas	This is the radon detector in building 226, 2nd floor, hallway A, near room 226.	Thu Oct 04 12:19:03 EDT 2007
smokeDetector1@p623572.campus.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 316.	Thu Oct 04 12:19:03 EDT 2007
smokeDetector2@p623572.campus.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 2nd floor, hallway A, near room 255.	Thu Oct 04 12:19:03 EDT 2007
smokeDetector3@p623572.campus.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 350.	Thu Oct 04 12:19:03 EDT 2007
temperatureSensor1@p623572.campus.nist.gov	Other	Value:Temperature	72.5 degrees Fahrenheit	Thu Oct 04 12:19:03 EDT 2007
radonDetector1@seurat.cbt.nist.gov	Safety	Gas	This is the radon detector in building 226, 2nd floor, hallway A, near room 226.	Thu Oct 04 12:19:05 EDT 2007
smokeDetector2@seurat.cbt.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 2nd floor, hallway A, near room 255.	Thu Oct 04 12:19:05 EDT 2007
smokeDetector3@seurat.cbt.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 350.	Thu Oct 04 12:19:05 EDT 2007
smokeDetector1@seurat.cbt.nist.gov	Fire,Safety,Security	Smoke	This is the smoke detector in building 226, 3rd floor, hallway B, near room 316.	Thu Oct 04 12:19:06 EDT 2007
temperatureSensor1@seurat.cbt.nist.gov	Other	Value:Temperature	72.5 degrees Fahrenheit	Thu Oct 04 12:19:06 EDT 2007

Figure 23 - The Alert Status Screen for Monitoring 2 BISACS Base Servers via a BPS

Drilling Down for Alert Information

Clicking on the Alert ID from the Alert Status Screen takes the user to the Alert Information Screen. Figure 24 shows all of the related information from the Common Alerting Protocol message for the selected alert.

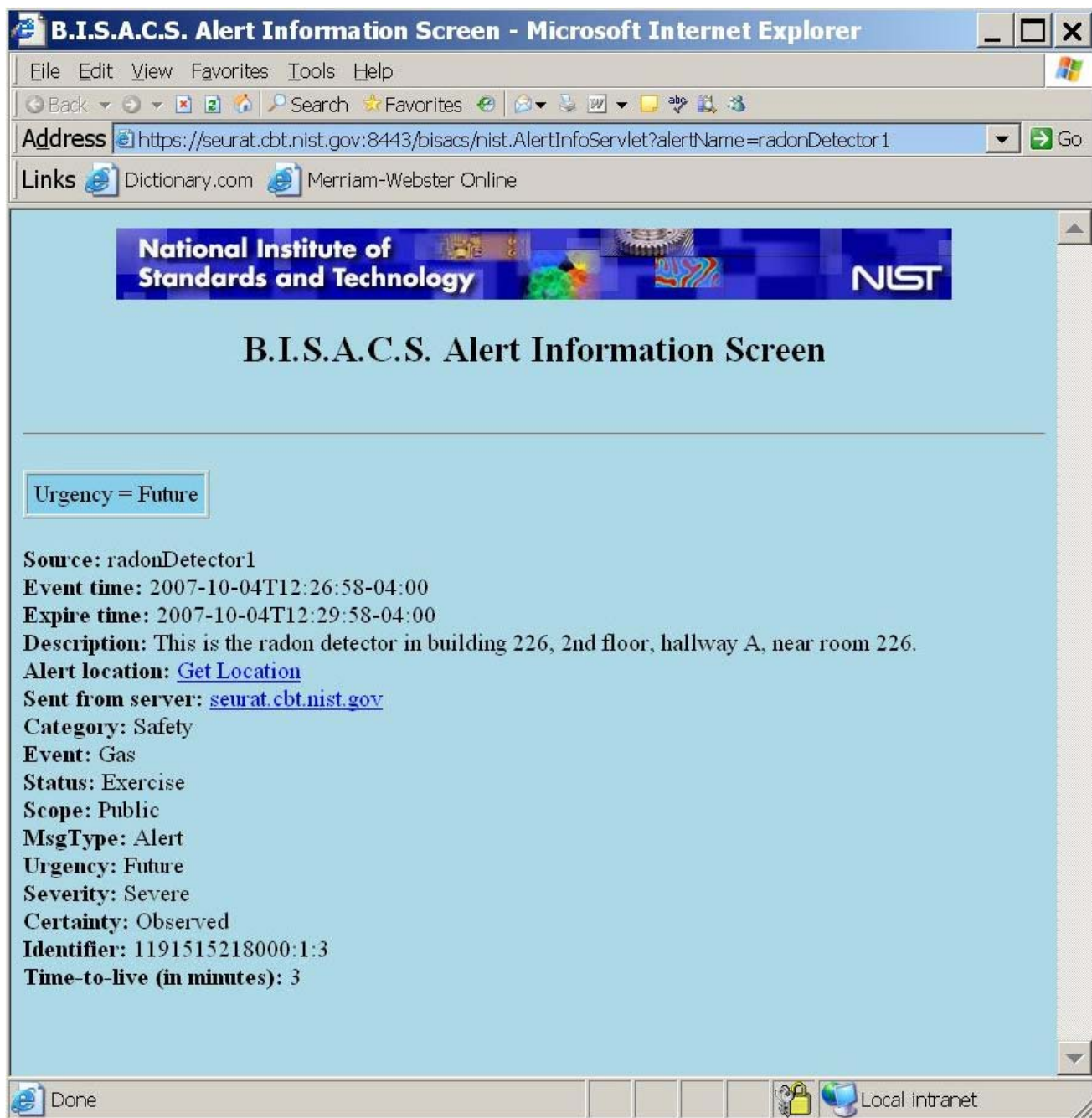


Figure 24 - The Alert Information Screen Showing Information for a Specific Alert

Clicking on the “Sent from server” link will take the user to the login screen for the BISACS Base Server that propagated the alert; this allows quick access to the BBS where the alert was originated from. Clicking on the “Alert location” link brings up the floor plan and shows where the sensor device resides in the building that propagated the alert. Figure 25 shows an example of such a floor plan.

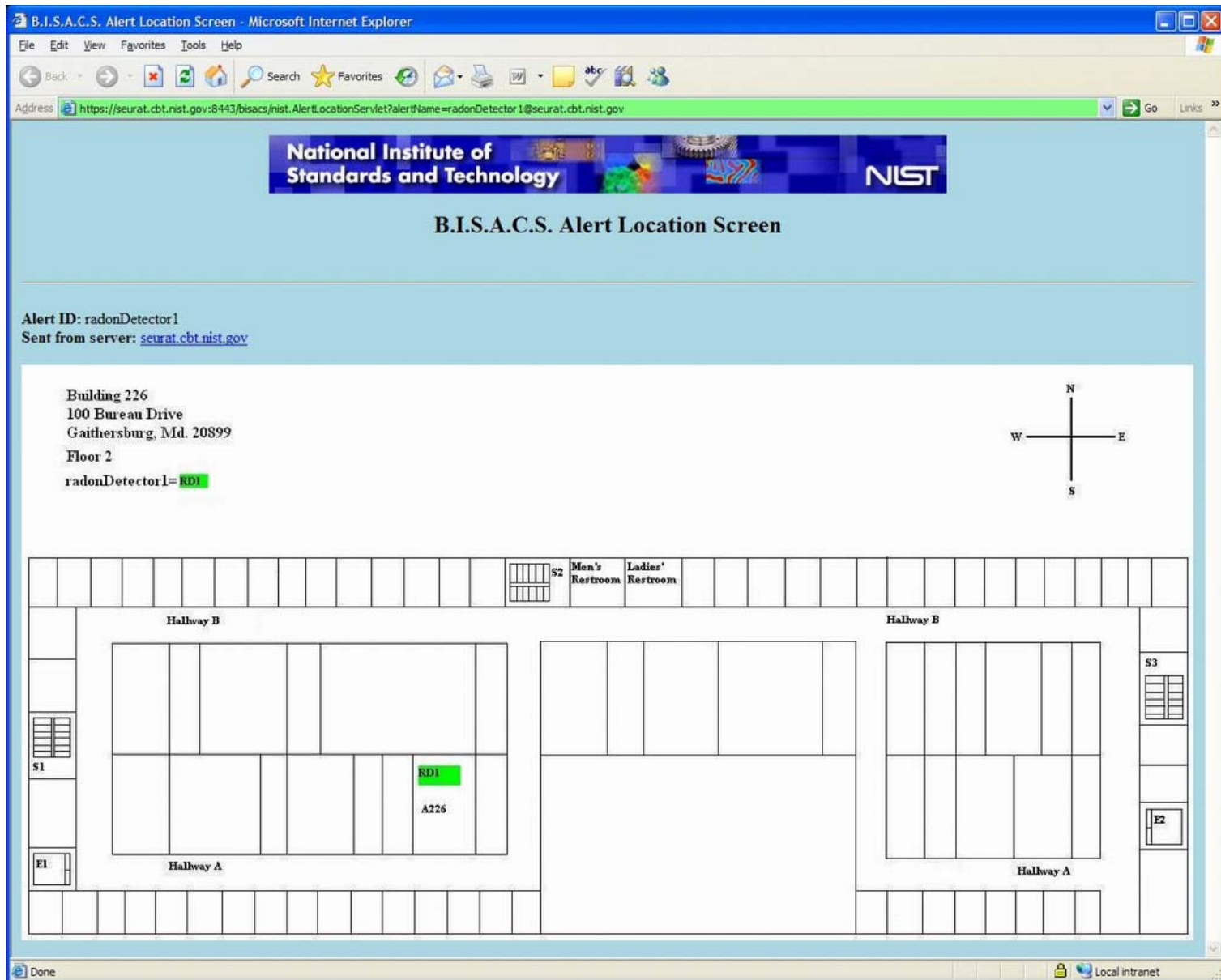


Figure 25 - The Alert Location Screen Showing the Floor Plan of Where the Sensor Resides

Notice that on this screen, clicking on the “Sent from server” link will also take the user to the login screen for the BISACS Base Server that propagated the alert.

Future Goals

With the underlying software framework in place supporting certificate authentication, user verification, and information exchange for the BISACS, future goals for the BISACS include but are not limited to the following:

- 1) Support the communication process with sensors and devices to obtain real time status information such as active/inactive status and other information those devices may have to offer.
- 2) Develop a mechanism to remotely store and retrieve building information such as building floor plans.
- 3) Develop a mechanism to remotely execute or forward commands to various devices within a building such as a switch or valve.
- 4) Develop a mechanism to demonstrate role-based user authorization.
- 5) Develop a mechanism for using a key or token so that only verified users will be allowed to access the BisacsPrimaryPortal web services, i.e., so that a user can not use his own software to communicate with the BisacsPrimaryPortal directly without going through the user verification process. This key or token can only be obtained once the user verification process is successful. Currently, the basic infrastructure to support this function is in place but the actual usage of the security key has not yet been utilized.
- 6) Demonstrate that the BISACS provides an effective and scalable solution for propagating alerts to emergency responder facilities and for authorized personnel to effectively and remotely control building devices.

Lessons Learned

Using existing technology, the BISACS project demonstrated the ability to collect data available from various building sensor devices and to form alerts where appropriate in order to notify external clients such as first responders. The capability to inform the emergency first responder community of various alerts in real-time can play a major role in the decision making process for dispatching the appropriate personnel and equipment to emergency incidents. The real-time information can also aid the emergency first responder community in dealing with the incidents once the personnel are on site.

The BISACS project demonstrated the ability to encrypt data channels over the Internet so that sensitive information can not be readily available to unauthorized users. The use of certificates of authenticity along with user identifier and password combinations restrict access to authorized users only. These attributes of the BISACS constitute a safe and secure environment for dealing with sensitive information such as building alerts over the Internet.

The design of the BISACS network demonstrated the ability to configure servers to monitor a small amount of devices up to much larger network of buildings and their related devices. This architecture showed that the BISACS network is scalable and robust enough to potentially handle site wide or city wide or country wide network of BISACS servers.

References

- 1) Holmberg, David G., Davis, William D., Treado, Stephen J., Reed, Kent A., “Building Tactical Information System for Public Safety Officials, Intelligent Building Response (iBR)”, NIST Internal Report 7314, January 2006.
- 2) Common Alerting Protocol, v. 1.1, OASIS Standard CAP-V1.1, available at http://www.oasis-open.org/committees/download.php/15135/emergency-CAPv1.1-Corrected_DOM.pdf, October 2005.
- 3) Web Services Security, X.509 Certificate Token Profile, OASIS Standard 200401, available at <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf>, March 2004.
- 4) UDDI Version 3.0.2, UDDI Spec Technical Committee Draft, available at http://uddi.org/pubs/uddi_v3.htm, October 19, 2004.
- 5) Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, available at <http://www.w3.org/TR/wsdl20/>, June 26, 2007.
- 6) ISO/IEC 10918-1: Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines, available at <http://www.w3.org/Graphics/JPEG/itu-t81.pdf>, September 1992.

Appendix A - List of Software Tools

Various Java technologies were used in developing the Building Information Services and Control System (BISACS), this section acknowledges what software products/packages were used in developing the BISACS. Software packages that were involved with developing and supporting the BISACS included but not limited to the following:

- The Apache Tomcat Project was used as the Application Server for hosting the BISACS applications. <http://tomcat.apache.org/>

- The Eclipse extensible development platform was used as the Integrated Development Environment (IDE) for developing the Java code used by the BISACS. <http://www.eclipse.org/>

- The Eclipse Web Tools Platform (WTP) project was used for developing Web Services Interfaces and for publishing Web Services. <http://www.eclipse.org/webtools/>

- The Apache Axis Project was used to handle all Simple Object Access Protocol (SOAP) communication. <http://ws.apache.org/axis/>

- Sun's Java Development Kit (JDK) and Java Runtime Environment (JRE) were used for developing and testing the BISACS. <http://java.sun.com/>

- The Apache jUDDI Project was used as the Universal Description, Discovery, and Integration (UDDI) registry server. <http://ws.apache.org/juddi/>

- MySql was used as the Database Management System (DBMS) for the BISACS. <http://dev.mysql.com/>

Appendix B - Hardware Platform Used

Software development was done on a Dell Precision 380 computer with the following specification. This machine also served as a test machine:

- Windows XP Professional, Version 2002, Service Pack 2
- Intel Pentium 4 CPU 3.6GHz
- 1GB of RAM
- 80GB hard drive

The target server machine was a Dell Power Edge 650 computer with the following specification:

- Red Hat Enterprise Linux 3
- Intel Pentium 4 CPU 3.06GHz
- 4GB of RAM
- 80GB hard drive

A laptop PC used for testing purposes acting as a BBS and as a BPS was a Dell Latitude D820 computer with the following specification:

- Windows XP Professional, Version 2002, Service Pack 2
- Intel Centrino Duo, Core2 CPU @ 2.33GHz each
- 2GB of RAM
- 95GB hard drive

The software was created using the Java computer language that is platform independent, hence any platform supporting Java can run the BISACS software.