VIsualizing Enterprise-Wide Security (*VIEWS*)

J. J. Brennan
The MITRE Corporation
jjbrennan@mitre.org

Don Faatz
The MITRE Corporation
dfaatz@mitre.org

Mindy Rudell
The MITRE Corporation
mrudell@mitre.org

Carson Zimmerman
The MITRE Corporation
carson@mitre.org

Abstract

This paper discusses VIEWS, a specification for building diagrams that describe the security features of systems. The authors' recent experience with providing security architecture and engineering support to organizations with large, distributed applications suggests that security architecture and assurance efforts could benefit by following other engineering disciplines, where using graphical models is the norm Security diagrams can help security architects understand a system's security posture and can assist them in detecting vulnerabilities. Additionally, diagrams facilitate communications about the security features of a design.

The output of a modeling effort using VIEWS is a diagram depicting a system's security features as well as those of the environment in which the system operates. A goal of VIEWS is to allow the display of important security features without injecting cluttering detail. This paper presents examples of security diagrams built with VIEWS.

1. Introduction

This paper reports on a specification that can be used to visually model the security features of distributed application systems. The specification, named *VIEWS*, is the result of ongoing research and technology transfer efforts, and is driven by the practical problems encountered in providing security design and engineering support to large, multi-location organizations. The rationale for developing *VIEWS* started with the belief that security engineering could benefit by using visual aids, paralleling what is generally accepted practice elsewhere. Ideally, visual representations can depict where security mechanisms are positioned, how security mechanisms relate to the system design, and how security requirements are met. An expected benefit of providing a diagram of a system's security position is improved communications, not only between security engineers and architects and their systems engineering counterparts, but also with others who have a vested interest in a system's security, such as the system owner.

The ability to convey the essence of a security design visually, while following some formalism for constructing its visual representation, is not generally available. Even where security features are pictured, for instance in a network diagram, the features themselves usually give only vague hints as to the actual security posture in place.

Within a short time after starting work on *VIEWS* it was clear that a number of other factors summoned for the type of effort described here. Organizations have found themselves delivering an increasing number of services over the Internet or internal versions thereof. For many, the situation is further complicated as they adopt new information technologies such as directories, which themselves might be important components of a security design. These services can now be distributed across multiple networks or geographical locations, and be deployed within multi-tier configurations. It is also not unusual to find different security policies in effect depending on the location or responsible owner of a service. In summary, many organizations must adapt to new methods of delivery, to an accompanying increased complexity in their systems and system designs, and to new classes of threats that follow from allowing wider and different styles of access to the organization's data and services. In such environments, having tight, clear descriptions of both system and security designs is invaluable. The authors believe that *VIEWS* can greatly assist in developing such descriptions.

VIEWS marries system and security representations to bring together features that reveal a system's security posture. It turns out that system-level security is remarkably resistant to being described piecemeal; the security of each piece is dependent on the security of many others. A significant challenge is to build diagrams that convey a large amount of security-relevant information, while at the same time melting away much of the detail that would obscure the diagram's readability, and thus its impact.

Tufte [8] has produced a number of works on the use of graphic arts to convey information, the reference cited being just one example. That interaction among security controls is an important consideration is noted in [6]. Suggestions that distributed environments pose new challenges for dealing with security appear in [4].

The work in [5] attempts to make UML [7] entities "security aware" by adding low-level descriptions of the security properties associated with entities. While the approach is useful for documenting the properties where they exist, such low-level descriptions fail to adequately represent security as it exists at the level of a system. UML, even with extensions, is geared to help programmers, not security architects. Flechais et al. [2] also use UML to depict security properties to support software development.

The focus of this paper is the presentation of examples of the use of *VIEWS*. The main example is an adaptation of a simple system encountered in practice, and illustrates the types of features that *VIEWS* incorporates. The examples illustrate how security depends on system components acting in concert. The next section provides a brief summary of the formalisms underpinning *VIEWS*.

2. The *VIEWS* Approach

The output of a modeling effort using *VIEWS* is a diagram depicting a system's security features as well as those of the environment in which the system operates. *VIEWS* itself is a set of definitions and rules of construction governing creation of a visual model of a system's security features. *VIEWS* makes no direct attempt at prescription. Rather, it is a method that helps security architects describe the security features of systems, leaving to them the choice of whether the description represents what exists or what should exist.

A major goal of *VIEWS* is to support the display of important security characteristics while allowing unimportant or cluttering detail to be moved out of a diagram entirely, i.e., creating models at a useful level of abstraction. Among the features that carry useful security information and lend themselves to abstract representation are: a) the important system elements and the connections between them, b) the position of security mechanisms, in particular the places or services making security-related decisions, c) the strength of mechanisms, and d) the security-relevant information flows between system elements. *VIEWS* is not intended to replace other design documentation. Rather, diagrams produced with *VIEWS* provide a convenient point of convergence for security-related information from other sources.

VIEWS diagrams can contain both logical and physical elements, and normally do. Using logical and physical elements together permits showing security features in context, for example a decision point (logical control) shown within its physical surroundings. When assessing the effectiveness of a security design, considering where security-related decisions are made is almost always important. Just as logical and physical elements can be mixed, *VIEWS* can combine elements and features from different frames of reference: network, operating system, or application.

VIEWS is a specification for building diagrams. In order to provide diagram features with unambiguous and consistent meanings, a formalism that guides diagram construction is embodied in the following parts of *VIEWS*:

- *Type Definitions* guide the construction of a diagram's "piece parts," and how the parts relate to one another. For example, specific model instances of a database and a communications segment are built using the Database and Segment types defined in *VIEWS*.
- *Renderings* are the specifications for the graphic representation of objects or groups of objects. (*VIEWS* currently employs Visio to build diagrams. However the style and shape of objects and the tool used to draw them are independent of the remainder of the specification.)
- *Interpretations* are statements about what rendered objects, or groups of objects, mean.

Additionally, *VIEWS* makes use of two other items which lie outside the formal specification itself:

- *Security Models* are abstractions that deal with fundamental security mechanisms, such as authentication, and are used to guide the development and enhancement of *VIEWS*.
- *Conventions* are suggestions to model builders which the authors find useful but are not part of the formal specification.

While the specification is formal it also attempts to minimize forced choices, providing considerable flexibility in diagram construction to the security architect.

3. Main Example

This section introduces the paper's main example which demonstrates a number of the features of *VIEWS*. The example is realistic, being an adaptation of a system the authors have encountered. However it is simple compared to what might be found in practice, having relatively few system and security elements, and only straightforward relationships between them. Nevertheless, the example is illustrative and exercises a number of *VIEWS* features. Rather than show a single, completed model, the example is exposed in two stages to simplify the discussion. *VIEWS* terminology appears in italics when first described.

The example models the following situation. A financial services company (FSC) allows its customers to access their financial information via an application and associated database resident at the company's facilities. Using a WWW browser, a customer connects to the FSC facilities through WWW servers at a site hosted by a third-party service provider. The modeling task is to describe what security mechanisms are in place, taking the viewpoint of FSC. As noted previously, a clear description of the mix of security controls actually in place can support a security architect's efforts to determine whether the controls are sufficient to meet FSC security requirements and to prescribe remedies if they are not.

A shaded rectangle surrounds the customer, a rectangle being the *VIEWS* representation for a *zone*. Zone definitions are a convenient way to make a statement about all objects contained within a zone's borders. Zones are useful in depicting locations, hardware, and groups of common services, although there is really no restriction on how they are used. Each zone can have a name, *zone type*, and set of *zone properties* associated with it. Zone types and zone properties as well as the zone name appear in the lower right hand corner of the zone in a separate rectangle with two of its sides rendered in grey, to distinguish its enclosed area from a zone itself. The inclusion of zone types, properties, or names is optional. In the figure, the leftmost zone shows a zone type using a building icon, the *VIEWS* representation for "any location." The fact that a customer can communicate with FSC from any location, including those not under the control of FSC, is noteworthy since no physical security controls are assumed to be operative. The middle and rightmost zones have a different building icon, this time indicating that they are computing centers, another noteworthy fact since this typically indicates that a formal security policy guides facility operations. The stop sign is a property symbol indicating physical security controls are in place.

References (unrestricted text) can appear alongside most *VIEWS* objects, and in the case that ambiguity might occur between the text for a name and a reference, *VIEWS* includes a reference object which can appear with its associated text anywhere on a diagram. While the authors have found that references are the convenient way to include detail, it is also possible to extend the set of objects in *VIEWS* to provide more specific information about some aspect of what is modeled. In the diagram, the center zone contains a reference to another document, with "# number" being the modeler's choice of how to show this reference.

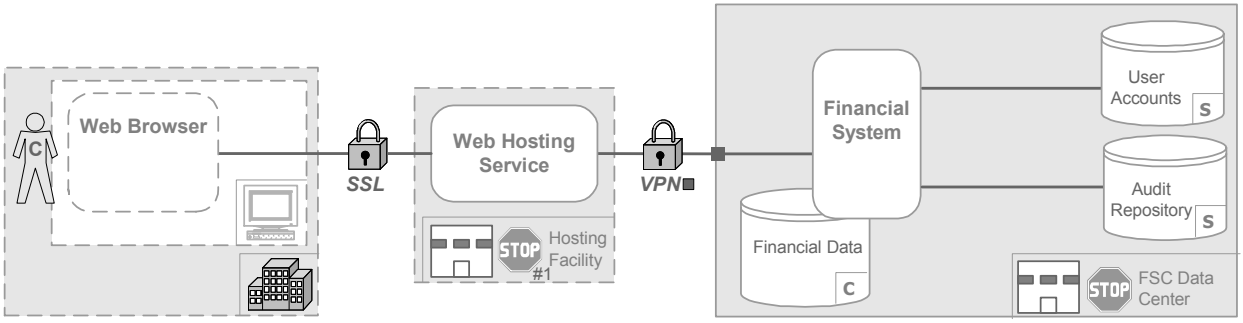

Figure 1. Model of FSC Infrastructure Using *VIEWS*

3.1 Depicting the Environment

Figure 1 presents a high-level visual model of the system design, its physical infrastructure, and some security features that are closely tied to the physical infrastructure. At the left of the diagram there is a stylized human figure representing an FSC customer, the customer being an instance of an actor type. The diagram builder can distinguish types of actors by defining type abbreviations and showing the abbreviation on the actor icon. In the example, C denotes a customer.

VIEWS takes advantage of the fact that the border of a zone can be shown in different styles. The border, either a solid line or a broken, indicates the presence or absence of a "secure" property, the definition of secure being left to the modeler. There is almost always some criterion which distinguishes operations or locales controlled by the system-owner from everything else, so varying the border style can model this distinction to advantage. Our experience suggests that doing so is almost always a necessity. In the hypothetical example shown, all networks within the FSC Data Center (at the right in Figure 1) are trusted to the extent that traffic across them does not require privacy protection. At the same time, FSC requires that communications with its customers be private. In this sense, the FSC Data Center is "secure" compared to other environments to which it connects; the other environments have broken zone borders to make this distinction.

The leftmost zone showing the location of the user also contains a hardware zone, hardware being just another case of the zone abstraction. The diagram indicates that the hardware zone type is "workstation." The fact that the user icon touches the border of the hardware zone creates an actor-zone cluster whose interpretation is that everything inside the zone is under direct control of the actor.

This zone also contains an *application service* named "Web Browser," with applications or application services indicated by rectangles with rounded corners. The other zones in the diagram contain other applications, and the FSC Data Center contains three drums, each being the representation for a *data store*. The use of data store objects indicates only that data resides in them permanently, not the actual type of storage mechanism employed. *VIEWS* also defines two other storage objects: *directories* and *non-persistent storage*, the latter being a useful way to show the existence of cached data when relevant. Like zones, the lower right portion of the object contains type information, in this case the type of data held in storage. In the example, the C is customer data, while S is security data. Applications and actors are in a class of objects called system entities.

Communication between system entities traverses segments. Segments are logical and indicate that the potential for communication exists between the connected entities, whether the communication has direct security relevance or not. In the example, the segment connecting the Web Browser and Web Hosting Service indicates that communications can flow between them. In reality, a single segment might be realized using elaborate communications devices and a number of connected physical links. *VIEWS* diagrams do not show this detail since is usually unimportant in describing a system's security position.

While segments are logical, they can have communications medium properties, network, wireless, mail, and the like. By default, a segment's communications medium property is deemed to be ordinary network connectivity using wire or fiber, and is not shown, or more correctly is rendered as NULL. If the segment has noteworthy properties other than the default, *a segment property symbol* on the segment can indicate this. It is possible to define segments with mixed media and use multiple segment property symbols to do so, although we have not encountered a need to do this in practice.

Also by default, segments are bi-directional. If it is desirable to show that communications on a segment are uni-directional, for instance mailing a PIN to a user, *VIEWS* defines a uni-directional segment object. While how to depict communications between a segment's end points is a subject taken up shortly, there is no requirement to show communications or properties on a segment. A diagram can also imply the existence of a segment rather than showing it explicitly. In the example, the Financial System and Financial Data database overlap, indicating that a segment exists between them, but the model builder decided to depict these entities as an *application cluster*, where segments are implied between overlapping elements.

The example shows a segment property that obtains frequent use – a lock symbol indicates that the segment is encrypted. The scope of a segment property uses the convention that properties extend along the segment on which a property symbol appears, and any adjoining segments, until reaching the first actor, application, or storage element. Knowing this, it is intuitive that SSL privacy protection exists between the Web Browser and Web Hosting Service. The default scope convention for segment properties, although seemingly arbitrary, helps de-clutter diagrams. However, modeling a VPN between the Web Hosting Service and the FSC Data Center can not be done with out the help of an explicit property scope declaration using a segment scope symbol.

Another type of system entity, the *EndEntity*, fills the role of a scope symbol when the latter is needed. EndEntities depict system components that the model builder happens to want to show in minimal detail, like a network device that terminates the virtual private network segment indicated in the example. EndEntities also terminate segments. However they are invisible to segment properties, i.e., they do not terminate a segment property unless they are explicitly declared to do so. In the example, the position of the EndEntity scope symbol to the right of VPN means go to the right to find the first matching symbol and terminate the property there. Incidentally, EndEntities

can and should be shown inside or outside a zone's border if where the entity sits is a relevant detail.

The abbreviated diagram of Figure 2 shows how segment properties and scope get used. Application Services A and B communicate using SSL. This is shown using the lock icon with SSL on the center segment of the figure. By convention, the SSL property bores through both zone boundaries to A and B. In practice it is often the case that communications between facilities employs link encryption or VPN services. Assuming that the zones shown in the figure are facilities, a lock icon with VPN and a scope symbol stopping the VPN property explicitly at zone boundaries, models a VPN from one facility to another. More complicated situations would likely involve using multiple scope symbols. If the detail gets complicated to the point of distracting clutter, our approach is to build smaller, detailed diagrams and incorporate them by reference in a master diagram built at a higher level of abstraction.

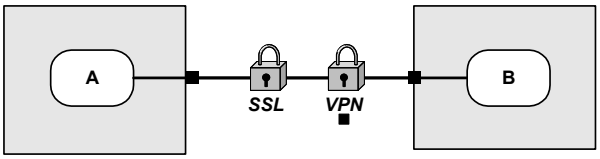

Figure 2. Example of Use of Segment Property Scope

4. Expanded Example

VIEWS provides abstractions for showing how and where security-related decisions are made. The exposition proceeds by continuing the example to show how diagrams can model authentication, access control, and audit decisions.

A fundamental *VIEWS* construct is that of a channel. Channels are virtual, one-way connections between entities, and they span the segments and entities between the channel's two end points. A pair of *bitbuckets* defines a channel and mark its beginning and end. Referring to Figure 3, the small rectangle with an outward arrow positioned to the right of Web Browser marks the beginning of a channel whose other end is marked to the left of the Financial System using a small rectangle with an inward pointing arrow. The placement of the bitbuckets and direction of the arrows indicate that the Web Browser is at the source end of a channel connecting it to Financial System.

Not only does *VIEWS* support modeling a system's important decision points, it also supports modeling the data on which security decisions depend as that data moves from place to place within a system. *VIEWS* diagrams use channels to deliver security relevant information, namely authenticators, privilege or identity credentials, reference data extracts, security decisions, access requests, and audit data. The type of security relevant information is shown as the contents of a bitbucket as will be explained below.

To assist the reader in following the examples, Figure 4 contains a legend with some of the symbols used in *VIEWS*. The authors readily admit that some *VIEWS* notation is minimally related to the underlying model components. However, most of the project effort went to developing the *VIEWS* structure; easily distinguished shapes available on Visio stencils suffice for now.

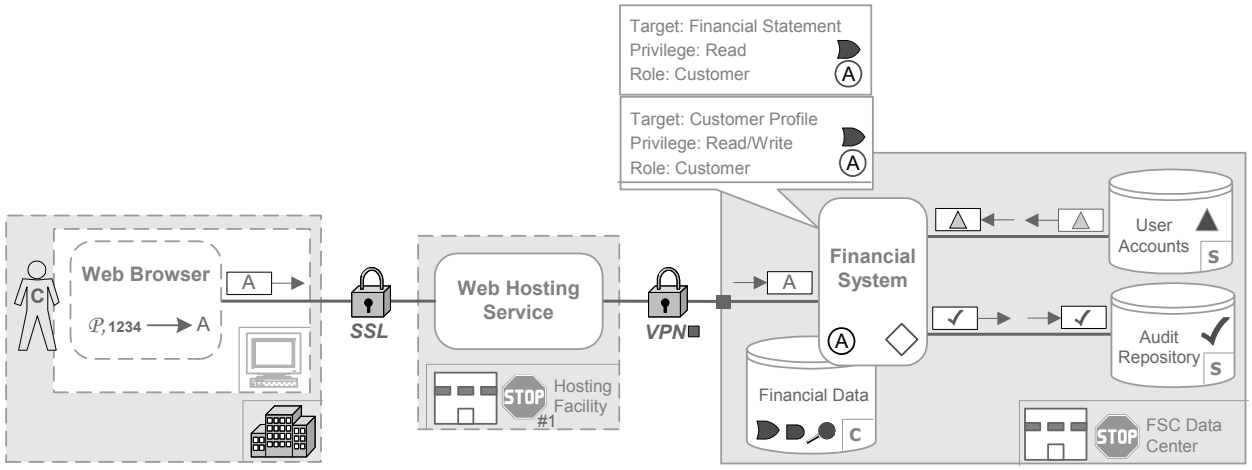

Figure 3. Expanded Model of FSC Services Using *VIEWS*

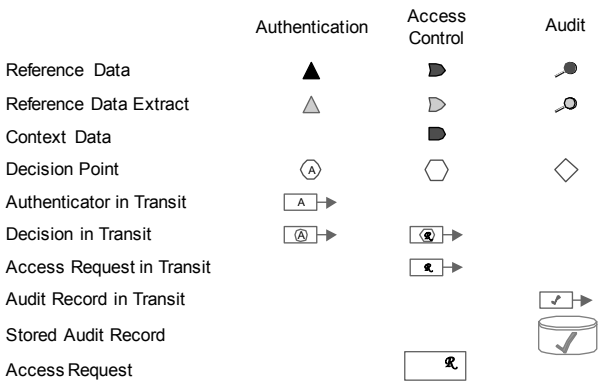

Figure 4. Symbol Legend

4.1 Authentication

The *VIEWS* conceptual authentication model, which accommodates many different authentication scenarios, is that of a decision point where a claimed identity of a system entity is either accepted or rejected. The decision takes as inputs the claimed identity, evidence of the claim provided by the claimant, some authentication reference data, either in raw or extracted form, and perhaps some state information. In *VIEWS* the term "authentication reference data" means the authoritative data on which an authentication decision is grounded. However it is often the case that the decision point does not have direct access to such data and must instead rely on some extract, copy, or compilation of the authoritative source. The latter data is referred to as an authentication reference extract.

Figure 3 depicts authentication of a customer by the Financial System application. The customer uses a PIN and password for application-level authentication at the Financial System. To model this, a $\mathcal{P}$ and the numerals 1234 appear on the blunt side of an arrow situation at the Web Browser. $\mathcal{P}$ and 1234 are the *VIEWS* representations of password and PIN, respectively, while the arrow represents a cause-effect relationship or operation, the cause appearing on the blunt side. Note that the details of such operations, such as whether the PIN and password are hashed, are left to be described elsewhere.

The letter A on the right side of the arrow is an authenticator, which normally consists of a claimed identity, such as a username, and evidence to verify the right to use that identity. In the example, the fact that a user's name is part of the authenticator is implicit, although it could be shown. The authors' convention that using an A for the authenticator carries the information that this authentication is at the application level, as opposed to the enterprise, operating system, or network levels which have their own authenticator symbols, E, O, and N. The authenticator travels to the Financial System over the channel shown.

Note that in this simple transfer the diagram exposes security controls at different layers of the system and in different places, and clearly shows the protections afforded this particular authenticator. In fact, whatever the actual form of the authenticator, there appears to be some exposure at the Web Hosting Service, since the channel lacks communications privacy protection there. In practice, FSC should require special physical and procedural controls for Web Hosting Service and a revised diagram might show these with a zone and zone properties.

Small circles with an authenticator symbol in them mark the location of an authentication decision point, shown in Figure 3 at the Financial System application. Here the authenticator is compared to authentication reference data and system state information, the latter called context data in *VIEWS*. The User Accounts database shows a small, filled triangle, the representation of authentication reference data. The fact that the triangle is filled with black is significant – all authoritative data sources are rendered filled with black while extracts of authoritative sources are rendered with a grey fill. Thus what is depicted at the FSC Data Center is an extract of authoritative authentication reference data moving across a channel from the User Accounts database to the Financial System application, where an authentication decision is made. The Financial System also has access to context data in the Financial Data database, this data shown as a filled AND-gate.

4.2 Access Control

As in the case of authentication, *VIEWS* uses a conceptual model to develop the specifics of how to depict access control decisions. However in distributed systems, modeling access control decisions is a bit more complicated than it might first appear, mainly because the points where authentication, the access control decision, and access control enforcement occur can all be different, along with the fact that access control requests can arrive at either a decision or enforcement point. The main inputs to an access control decision are proof of a verified identity, an access request, access control reference data (some form of privileges or policy which match the identity), and context data. The simple situation depicted in the example should impart a feel for how *VIEWS* depicts access control. In the example, the Customer can request two types of system access from the Financial System.

VIEWS diagrams use a rectangular callout to show details of an access control decision. In Figure 3, one

appears to the upper left of the Financial System application. The callout has three parts – one or more request blocks, an action block, which appears at the lowest part of the rectangle separated from request blocks by a horizontal line, and a pointer. An access request block shows Target, Privilege, and Role. The target of an access request is the resource that will be accessed if the request is allowed; the privileges are the actions, such as read or write, which may be performed on the target; the subject is the system entity that will access the target. A single callout can show multiple request blocks providing that their decision points and enforcement points match. The request block can also contain references to "Privilege" or "Role." In the example, across from "Privilege" is an access control reference data symbol matching that in the Financial Data database. The intent is to bind the request for privileges to the source of the policy data used to grant the request. The authentication decision point symbol across from "Role" links the entity making the request to the location of the authentication decision. In some circumstances, placing references in request blocks permits showing the flow of security relevant information by implication rather than explicitly.

The pointer touches either the decision or enforcement location, whichever one receives the access control request first. The action block then contains the name of the other location, the one not pointed at. If the enforcement and decision points are the same, the pointer touches this single entity and the action block is empty, the situation in this example.

At the decision point, an access request is compared with access control reference data. In the example, this data resides in the Financial Data database and is represented by a filled OR-gate. Context data is also available to the decision point.

4.3 Audit

In the example, the Financial System generates event records for the authentication and access control decisions it makes. The *VIEWS* conceptual audit model follows what happens with UNIX syslog [3]. With syslog, system entities generate event records which are sent to an audit decision point which compares the event records to an audit policy (audit reference data). Based on the audit reference data, the audit decision point selects some event records to be saved in an audit repository. Authentication decisions and access control decisions are automatically event record generators. An override symbol, not shown, turns the default off.

Event records are represented by bitbuckets carrying a check mark and the same check mark in a data store represents stored audit records. Audit reference data is represented by a magnifying glass (an icon chosen solely because its shape is easily distinguished from others). Audit data records destined for the Audit Repository move over the channel shown.

5. Policy Server Example

This section presents a different example. A user connects to a WWW service which consults a separate security policy server to receive decisions about authentication and access control. A security service like the one in the diagram would normally be available to a number of other services or applications in a larger system. Building a diagram of a reasonably complicated service can help communicate what the service delivers to relying applications and how it interacts with them.

The situation depicted is typical of what exists in many modern WWW-based systems and the policy server itself is representative of functionality that can be found in commercial products. The example shows a user, U, who will attempt to connect to the service named Web Services.

Authentication proceeds in a manner similar to the previous example. The user sends an enterprise-level authenticator, E, to Policy Server over the channel

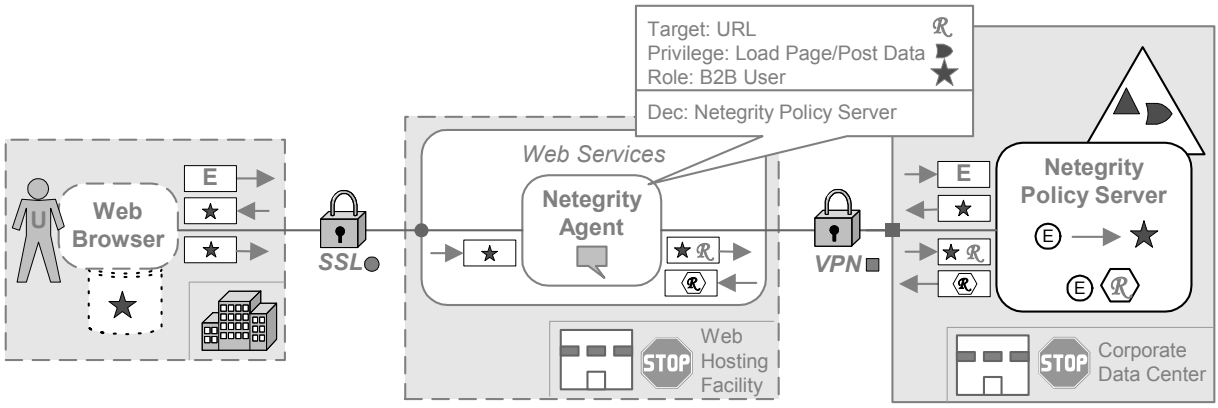

Figure 5. Policy Server Example

shown. Policy Server uses the authenticator and authentication reference data stored in its directory (triangle) to make an authentication decision. If successful, Policy Server creates a token (cookie), represented by the star, and returns it to the user's Web Browser. Again the cause-effect abstraction is used with the arrow indicating that the creation of a token flows from the act of making an authentication decision. The token is stored at the user's location and sent to Web Services when a request for service is made. The token is shown stored in non-persistent storage (drum with a dotted border).

After authenticating, the user makes a request for services. The rectangular callout at Web Services shows that the user is requesting access to a URL to load a Web page or to post data. In this example, the access control decision and access control enforcement occur in different places, the decision at the Policy Server and the enforcement at Web Services. Since the user's request is sent to the Web Services, the callout attaches there, and indicates where the decision point is in its action block.

The top portion of the callout, the request block, has a name $\mathcal{R}$, which can be found in the block's upper right hand corner. Web Services passes the request and the token to Policy Server, which contains an explicit access control decision point marking, a hexagon containing a reference to the request. The decision point uses the token and access control reference data stored in its directory to make the necessary access control decision. Finally, the results of the decision are sent back to Web Services for enforcement. The way *VIEWS* handles this is to show the decision point mark within a bitbucket to indicate the movement of a decision across a channel as depicted in the figure.

The example surfaces the question of how *VIEWS* deals with event sequencing or protocol descriptions. The answer is that it doesn't. The authors believe that sequence descriptions are best left to existing methods specifically designed to build state or sequence diagrams, such as UML. A *VIEWS* diagram can reference the sequence diagram if it becomes necessary to explore a design at that level of detail. However, the authors do employ a convention that gives at least partial information about the sequence of events. The positioning of bitbuckets top to bottom at the end of a channel shows the order in which information departs or arrives.

6. Conclusion

This paper introduced the *VIEWS* specification by presenting examples that demonstrated a number of its features. *VIEWS* began with and evolved from the authors experience in dealing with security designs in distributed applications environments, and its development was informed by what actually happens in practice. The authors have already built models of eight systems, the smallest roughly the size of the example to one that contains 200 model elements (to the example's 37) including 38 system entities, 28 channels, and 14 decision points. While the main example presented above depicts a small system built following a contemporary design, our experience includes building models for systems of various ages, from those spanning decades of operation to those still being designed.

VIEWS can model authentication, access control, audit, and communications security in these environments and we believe it can be extended to cover other security controls and environments as well. It is a specification that deals with security first, and allows building models that are not retrofitted to a network engineering or a systems development view.

VIEWS formalizes the construction of visual models of security, the formalization producing a number of important benefits. A formal specification brings with it control of diagram semantics, so two different people familiar with *VIEWS* can reach a common understanding when looking at the same diagram. For organizations whose applications are mainly independent but which utilize common infrastructure services, building models of security features with *VIEWS* offers the possibility of developing a view of security that is consistent across the applications. The Policy Server example presented suggests that organizations can model their common security services once, the model being applicable to the numerous applications using common services.

Besides the fact that diagrams can assist in extracting the essentials of a security design from volumes of documentation, they have proved to have practical benefits in other ways. An important use of *VIEWS* is in fact finding to resolve conflicting information. Conflicting information can arise for a number of reasons, often because of the different viewpoints of various interested parties, e.g., network, application, or operations personnel. *VIEWS* diagrams can show features that are physical and logical, that exist at different locations, and that arise in different layers, network, application, etc. Diagrams are also useful in determining what is actually implemented versus what is thought to be implemented, including a system's security features. *VIEWS* diagrams help show how a variety of controls cooperate to effect security.

Near term, we will continue to expand *VIEWS* modeling capabilities, driving enhancements by modeling additional systems in the environments where we have used *VIEWS* before, and by seeking out new environments and the challenges that come with

them. Modeling system and security administration, which the authors have not yet done, seems to demand building separate, administration diagrams, the security requirements and controls present usually being different than those that deal with the general user population.

We will also investigate the possibility of integrating *VIEWS* with other tools, for example, system design tools. We will examine the renderings to determine if their visual appeal can be improved. A useful addition to *VIEWS* would be a model checker which we believe could be easily built. We have not yet investigated the possibility of performing automated analysis based on definitions of the components used to build diagrams, although moving in such a direction seems feasible. We also plan to report on the formal structure of *VIEWS*.

7. References

[1] R. J. Anderson, *Security Engineering*, Wiley, 2001.

[2] I. Flechais, M. A. Sasse, and S. M. V. Hailes, Bringing Security Home: A Process for Developing Secure and Usable Systems, *Proceedings of the 2003 Workshop on New Security Paradigms*, 2003.

[3] A. Frisch, *Essential System Administration, 3rd Edition*, O'Reilly, 2002.

[4] I. Georgiev and I. Georgiev. A Security Model for Distributed Computing. In *The Journal of Computing in Small Colleges*, v.17 n.1 p.178-186, Oct. 2001.

[5] T. Lodderstedt, D. Basin, and J. Doser, SecureUML: A UML-Based Modeling Language for Model-Driven Security, 5th International Conference on the Unified Modeling Language, 2002.

[6] R. Nelson, Integrating Formalism and Pragmatism: Architectural Security, *Proceedings of the 1997 Workshop on New Security Paradigms*, 1998.

[7] J. Rumbaugh, I. Jacobson, G. Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[8] E. R. Tufte, *Envisioning Information*, Graphics Press, 1990.