ABSTRACT

LANGUAGE DESIGN FOR TOKEN SERVER AUTHENTICATION POLICIES

by Rebecca Lynn Busch

There exist computer networks in which multiple services forward the credentials of their users to a central server for authentication. Though the communication lines between the services and the central server may be encrypted, each service has plaintext access to the username/password pairs and can easily store copies. A solution to this security problem is the use of a temporary token in place of the password. Substituting short-lived tokens for the real passwords solves the problem of compromised passwords since only the token may be stolen; however, the tokens may become compromised as well. For this reason, restrictions must be placed on the creation and validation of tokens to reduce the likelihood of their compromise. In this work, I present an authentication policy language developed to express the rules necessary to govern the token creation and validation processes.
A Thesis

Submitted to the
Faculty of Miami University
in partial fulfillment of
the requirements for the degree of
Master of Computer Science

Department of Computer Science & Systems Analysis

by
Rebecca Lynn Busch
Miami University
Oxford, Ohio
2004

Advisor____________________
Dr. Scott Campbell

Reader ____________________
Dr. James Kiper

Reader ____________________
Dr. Eric Bachmann
# TABLE OF CONTENTS

1. INTRODUCTION ................................................................................................................................................. 1

2. LITERATURE REVIEW ........................................................................................................................................... 3
   2.1 LDAP – SINGLE AUTHENTICATION SITE ........................................................................................................... 3
   2.2 TICKET GRANTING SYSTEMS .......................................................................................................................... 4
   2.3 SECURITY POLICIES ...................................................................................................................................... 5

3. TOKEN SERVER AUTHENTICATION ..................................................................................................................... 8
   3.1 LDAP TOKEN PROXY PROJECT (LTPP) ................................................................................................................ 8
   3.2 TOKEN UTILIZATION RULES ........................................................................................................................... 9
   3.3 TOKEN POLICY LANGUAGE ............................................................................................................................ 10
   3.4 LTPP POLICY PROCESSING ........................................................................................................................... 10

4. TOKEN SERVER AUTHENTICATION POLICY LANGUAGE ..................................................................................... 13
   4.1 DESIGN ......................................................................................................................................................... 13
   4.2 IMPLEMENTATION ...................................................................................................................................... 18

5. TESTING AND ANALYSIS ...................................................................................................................................... 21

6. CONCLUSION ...................................................................................................................................................... 26

7. REFERENCES ...................................................................................................................................................... 27

8. APPENDIX ......................................................................................................................................................... 28
   8.1 GRAMMAR FOR TOKEN SERVER AUTHENTICATION POLICY LANGUAGE .................................................. 28
   8.2 POLICY FOR UNIVERSITY SCENARIO ............................................................................................................... 30
   8.3 POLICY FOR SMALL BUSINESS SCENARIO .................................................................................................. 32
LIST OF TABLES

Table 1: Token Server Properties ................................................................................................. 13
Table 2: Basic Structure of Creation and Validation Rules .......................................................... 14
Table 3: Basic Form of Interval Conditions .................................................................................. 14
Table 4: Complex Match Statements .......................................................................................... 15
Table 5: Issue/Validate Statement Effects .................................................................................. 15
Table 6: Token Properties for Restriction .................................................................................. 15
Table 7: Simple Creation and Validation Rules ........................................................................... 16
Table 8: Rule Statement Combinations and Meanings ................................................................. 16
Table 9: Complex Validation Rule .............................................................................................. 17
Table 10: Complex Validation Rule Expressed as Multiple Simple Rules .................................. 17
Table 11: Example Default Rule ............................................................................................... 17
Table 12: Example cRule and Resulting Abstract Syntax Tree ................................................... 19
Table 13: Example Error Statements Produced During Parsing ................................................ 20
Table 14: University Scenario - Expected and Actual Token Creation Results ........................... 22
Table 15: University Scenario - Expected and Actual Token Validation Results ........................ 23
Table 16: Small Business Scenario - Expected and Actual Token Validation Results ............... 24
Table 17: Example Validation Rule for Subjectivity ................................................................... 25
LIST OF FIGURES

Figure 1: Authentication via a Lightweight Directory Access Protocol (LDAP) Server .......... 3
Figure 2: Username/Password Vulnerability ........................................................................... 3
Figure 3: LDAP - Single Authentication Site .......................................................................... 4
Figure 4: Kerberos Authentication ......................................................................................... 5
Figure 5: Obtaining a token from the LTPP system ................................................................. 9
Figure 6: Using a token ........................................................................................................... 9
Figure 7: Token Creation Flow Chart .................................................................................... 11
Figure 8: Token Validation Flow Chart .................................................................................. 11
Figure 9: Policy Integration into LTPP .................................................................................. 18
ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Scott Campbell, for his support, guidance, and advice. I would also like to thank Dr. James Kiper and Dr. Eric Bachmann for their time and constructive comments.

I am grateful for the encouragement provided by my family and their constant belief that I will succeed. I would like to thank Adam for being there for me at all times.

Finally, I would like to thank my Lord and Savior, Jesus. For it is by His strength that I can do all things.
1. Introduction

Security implies a degree of trust. We have to trust the owners of information systems with our data and related processing. However, we probably should not absolutely trust them. We would not hand our house keys over to a stranger, yet we often willingly, albeit perhaps unknowingly, give our passwords to the administrators of networks, services, and websites. In most cases, we do not know the identity of these people; yet, we place our trust in their honesty. Our hope is that these administrators will not take advantage of us. But, we should limit the damage that a malicious administrator can potentially cause.

The basic dimensions of computer security are integrity, confidentiality, non-repudiation, availability, and authentication. [MAC 01] Integrity constraints address issues involving trust by ensuring that only authorized people or processes have created or modified the data. Confidentiality constraints ensure that only authorized people or processes have access to data. Non-repudiation prevents a user from denying his authorship of data or actions. Availability constraints ensure that people can access data when needed. Authentication constraints ensure that the system correctly identifies its users. Protecting or enhancing the system’s ability to correctly authenticate a user is the focus of this work.

Authentication entails accurately determining the user’s identity. There are three ways to establish a user’s identity: a biometrically unique characteristic of the user, something that the user has such as a token, or something that the user knows such as a password. These authentication measures can be used alone or in combination to verify a user’s identity. Authentication is crucial to the other four previously listed security dimensions.

Access control mechanisms ensure that the user is only able to perform operations for which he has appropriate permissions. These mechanisms support integrity and confidentiality. Access control depends upon accurately identifying the user. We need to know who the user is before we can determine if they are able to perform an operation. Access control rules are generally defined via policies, which will be discussed later in this thesis.

As discussed, secure authentication is necessary to achieve secure systems. This work focuses on enhancing the security of single-site authentication, a mechanism that many organizations utilize to provide authentication for the wide spectrum of services they offer to their users. With single-site authentication, a user may log into any available service with a single username/password pair, a convenience that makes this approach very popular. However, there is an inherent security risk with this method since the user has to present his plaintext username/password pair to the service. The service verifies these credentials with the single-site authentication system. There is no mechanism, which prevents the service from storing or otherwise exploiting these credentials. For example, the administrator could use a user’s credentials to masquerade as a user and access his confidential information.

To alleviate this security risk, Dr. Campbell is researching methods to extend the authentication system to include a secure and trustworthy token server. With this server, users obtain a temporary token to use in place of their password during the authentication process. In this way, a rogue service cannot compromise a user’s password. If the token is compromised, its
temporary nature reduces the likelihood that it will still be valid when an imposter attempts to use it. Dr. Campbell’s research is embodied in an authentication server project called the LDAP Token Proxy Project (LTPP) which will be described in chapter 3.

I have created a language and associated tools for expressing and translating rules that control the use of the temporary tokens in the LTPP. Having flexible rules is important because the risk of token compromise remains. Placing conditional restrictions on the use of these tokens may reduce this risk. A universal set of restrictions for token use is not the most effective way to enhance security since environments are not universally the same. Rather, it is better to customize token restrictions according to the characteristics of the user and the network environment. In this work, I introduce a language and implementation for creating such policies for token server authentication.

The organization of this thesis is as follows. Chapter 2 describes existing security policies and their notable features, and it addresses the need for a policy language designed specifically for token server authentication. This chapter also provides information about single-site authentication and ticket granting systems. Chapter 3 explains the basic concepts of token server authentication and the LTPP system design. Chapter 4 provides the structure of the token policy language and includes simple examples to illustrate the usefulness of particular language constructs in policy formation. Chapter 5 presents scenarios for which policies are designed and tested. Concluding remarks are given in Chapter 6.
2. Literature Review

2.1 LDAP – Single Authentication Site

There are many methods for user authentication. This work focuses on authentication via a Lightweight Directory Access Protocol (LDAP) [YEO 95] server, but it is applicable to other authentication methods as well. LDAP is a centralized database of properties that supports user authentication via username/password pairs. Programs, servers, and other systems utilize LDAP to authenticate users, which then allows them to provide subsequent authorization to their services. One LDAP server securely authenticates users for multiple services. To authenticate with a service, the user provides the service with his username/password pair. The service sends an LDAP bind request containing the username/password pair to the LDAP server. The LDAP server then authenticates the user and returns a success/failure message to the service.

![Figure 1: Authentication via a Lightweight Directory Access Protocol (LDAP) Server](image1)

There is an inherent security risk with this authentication approach that extends beyond the scope of typical encryption solutions. Regardless of the protocols and encryption mechanisms employed to transfer data across the network, the service has the user’s plaintext username/password pair. Even with the use of encryption like Secure Sockets Layer (SSL) [FRE 04] that protects the data as it is sent over the network, the encrypted connection ends at the service. The service has access to the unencrypted username/password pair. There is no guarantee that the service will not store a copy of the username/password pair before sending it to the LDAP server.

![Figure 2: Username/Password Vulnerability](image2)

The use of LDAP for community wide single-site authentication requires that all services be given access to unencrypted username/password pairs. While this is acceptable for systems run directly by the organization, it is not acceptable for all services that will use the LDAP server for authentication. For example, here at Miami University, many student projects use the LDAP
server to authenticate the users. However, this gives the students direct access to username/password pairs, which is unacceptable.

Figure 3: LDAP - Single Authentication Site

2.2 Ticket Granting Systems

One solution to this problem is to use a ticket granting system (TGS) like Kerberos. [KERBEROS 04] Kerberos utilizes private key encryption and acts as a trusted third party in the authentication process (Figure 4). Keys for all users and services are stored in the Kerberos server. In this system, the user provides his credentials and the name of the service to which he wants to gain authentication to the Kerberos client that is installed on his local machine. The Kerberos client then begins the authentication process; all intermediate steps are transparent to the user. The Kerberos client sends a message to the Kerberos server containing the username and a request for a ticket-granting ticket (TGT). If the Kerberos server finds the user in its database, it creates a session key and sends a message containing the TGT and session key to the Kerberos client encrypted with the user’s key. The Kerberos client then tries to decrypt the TGT message. If the decryption is successful, then username/password pair supplied by the user is valid. The Kerberos client sends a new message to the Kerberos server requesting a service ticket for the desired service; this message is encrypted with the Kerberos server key and contains information from the TGT. The Kerberos server decrypts this message and checks its validity with the data from the TGT. The Kerberos server creates a service ticket with the session key and encrypts it with the key of the desired service; the Kerberos server then sends the service ticket to the Kerberos client. The Kerberos client forwards the encrypted service ticket to the service when requesting service. The service decrypts this service ticket and extracts the session key. The service’s ability to correctly decrypt the session ticket validates the user since only the user could obtain a valid session ticket. Finally, the service responds to the service request and encrypts all data with the session key that is known to both the service and the client. [GAR 03] [SMI 02]
Ticket granting systems like Kerberos allow the user to successfully authenticate with a service without providing his password to the service. However, integrating a ticket granting system into the authentication process requires modification to existing client software and systems, an approach that we do not find acceptable. Rather, we are looking for infrastructure solutions that do not require any modification to services.

2.3 Security Policies

In many areas of computer security, policies drive the decision making processes. Access control policies are implemented to ensure that a user, who has already been authenticated, is granted access only to those resources for which he is authorized. Obligation policies require that certain actions, such as logging specific events, be carried out; these policies may be used in conjunction with accountability measures. Other policies are based on the mechanisms that provide authentication.

The goal of each policy is to define a plan of action to take given particular events and conditions that may occur. In [HP 99] a policy is defined to be a collection of rules; each rule has a set of conditions and an action to take when those conditions are met. Yet, each policy differs in the goal that it is designed to achieve. This difference in goals necessitates multiple types of policies.
[DAM 01] describes four subsets of the access control policy: authorization, information filtering, delegation, and refrain policies. Each of these policies contributes to the overall goal of the access control policy – to ensure that the user (subject) can only perform actions on resources (targets) for which he has been authorized. The authorization policy defines the set of activities that can be performed on each target by each subject. This authorization policy can define the activities either positively (permit) or negatively (disallow). Information filtering policies restrict the views of information available to each subject; a subject may only be authorized to view a subset of the target information. Delegation policies allow subjects to transfer their access privileges for target resources to other subjects. Lastly, the refrain policies specify actions that a subject must not perform even if they are otherwise permitted to do so. Together, these four policies define the access privileges that each subject has for each target resource.

Of these four access control policies, information filtering, delegation, and refrain policies are unnecessary for token server authentication. The goal of the delegation policy – to allow the temporary transfer of privileges – is undesirable in the token server. Successful authentication determines a user’s identity. A token server policy with delegation could enable a user to authenticate with services as though he was someone else.

The remaining access control policy, authorization, fails to support two objectives of token server authentication: allowing a policy rule decision to be *maybe* and enabling a policy to place limits on token use. Most authorization policies require that permission be either granted or denied. The token server policy contains multiple rules, and several of these rules may apply to a single request. The addition of a third value that neither grants nor denies the request is beneficial. In general, the majority of access control policy languages do not provide a means to set properties that affect the target object. It is desirable that a token server policy be able to explicitly limit token usage by setting token properties within policy rules. [KUD 00] expands the traditional access control model to allow for provisional actions that could potentially be used to set token properties. Rather than simply granting or denying access to objects, a policy rule can include a set of provisional actions that must be completed for the rule to be enforced. A rule in the token server policy should be enforced without a dependency on provisional actions. Although it may be possible to set the token properties by defining provisional actions, a dependency on successfully setting these properties is created. This dependency could potentially introduce unintended effects when the policy is enforced. No single access control policy has been designed with the same goals as those required for the token server policy.

Though less prevalent than access control policy languages, authentication policy languages have been developed. [STU 96] defines a language for creating authentication policies that handle the cryptographic protocols used for authentication in distributed systems. In [VEN 03] the authentication policy is extended to utilize the initial credentials provided by the user in order to determine the user’s associated permissions and roles. The policy is then able to dynamically decide which authentication mechanisms should be used to validate the user’s identity; in this way, the user is authenticated at a security level appropriate to his level of trust. This policy can require that a system administrator provide more credentials than the common user even though they both use the same system for authentication.
Token server authentication is unique in that it is a two phase process involving the creation and subsequent validations(s) of a token; the authentication process does not end with the creation of the token but continues until the token expires. The purpose of the token server policy language is to define restrictions for token creation and validation that effectively reduce the risk of the token being compromised. A policy created for token server authentication controls neither access to resources nor the mechanisms for authentication. Rather, it is designed to only allow token users who meet the requirements set forth in the policy to be eligible for authentication with the LDAP server. The token server policy also serves to explicitly limit token usage based upon properties belonging to the user, the token, and the system.
3. Token Server Authentication

I am extending Dr. Campbell’s LDAP Token Proxy Project to support a token policy language. Policies defined in this language reduce the likelihood of token compromise by restricting the token creation and validation processes. By defining these restrictions in a policy, one can tailor the token server to meet the authentication needs of a variety of computer networks. Before presenting the token policy language, it is necessary to understand the LTPP project.

3.1 LDAP Token Proxy Project (LTPP)

The LTPP extends the LDAP server to eliminate the need for users to give all services at Miami their username/password pairs. The LTPP system allows a user to be properly authenticated by the LDAP server without having to provide the traditional username/password pair to the service. Using the LTPP system, users can obtain a temporary token string from a secure and trustworthy token server. This token string replaces the user’s password when authenticating to services. The user supplies his username/token pair to the service, which accepts the token string in the same manner as the actual password. From the service’s perspective, the integration of a token server into Miami’s network is seamless; specifically, no changes to the service are necessary.

The token increases security due to its short lifetime and specificity. Whereas passwords at Miami are valid for months at a time, a token expires after a set number of seconds or other conditions. Thus, even if a copy of the username/token pair is stolen by a service, the likelihood that it retains its usefulness when an imposter attempts to use it is very low. In the event that a user’s password is stolen, chances are that the user does not even realize it; the imposter has full access to the network resources of the user. When the user becomes aware that his password has been compromised, he must change his password and learn the new one. The same rogue service could copy his new password and begin the cycle all over again. A password does not automatically expire after a short period of time or a set number of uses. By using a temporary token, the user need not worry about having his password compromised by a service since the token does expire.

Each token created by the token server has a set of properties that the token server uses to limit a token’s issuance and use. Many of these properties are initialized by the token server upon creation of the token. Examples include the creation time as well as information obtained from the LDAP server. Other properties are explicitly set during both the token creation and validation processes; these include the token lifetime and the maximum number of times that the token may be used.

The LTPP is a separate front-end to the LDAP server as shown in Figure 6. The LTPP processes LDAP requests from users or services and then forwards the requests to the LDAP server for final dispensation. The LTPP only modifies bind (authentication) requests when an appropriate token is found. The LTPP forwards all other requests to the LDAP server.

Use of the LTPP system consists of two phases: obtaining a token and using a token. When a user seeks to acquire a token, he first contacts the token server directly and provides his username/password pair to the token server. The token server authenticates the user by sending a
bind message to the LDAP server with the user’s username/password pair. If the bind is successful, the token server creates a random token string and stores this string in the token. The LTPP system stores the token, along with the username and password, in a new database record for the user. It then gives the user the associated token string. No token creation occurs if the LDAP authentication fails, and an error message is returned to the user.

![Figure 5: Obtaining a token from the LTPP system](image)

The second phase is token use. Having obtained a token, the user authenticates with a service by using the token string in place of his password. The service validates the user by sending a bind message to the token server with the supplied username/token string pair. The token server checks its internal database for a matching username/token string pair and, if found, replaces the token string in the LDAP bind request with the stored password. The token server then forwards the updated LDAP bind request to the official LDAP server. If the token server does not find a matching username/token string pair, it simply forwards the bind request unchanged to the LDAP server.

![Figure 6: Using a token](image)

3.2 Token Utilization Rules

The LTPP system must carefully manage the use of tokens in order to prevent the same security compromises that can occur when using passwords. While the use of tokens alleviates the problem of compromised passwords, measures must be taken during the token creation and validation processes to prevent the successful use of a stolen token. One approach would be to only allow a token string to be used once. However, this is not very flexible. Rather the restrictions that determine when a token may be created or validated are not applied to tokens in a universal format but are dependent on a variety of dynamic factors and configuration settings. This allows for customizing how tokens can be used on a site by site basis.

It is important to balance security with usability. Using the LTPP will limit the damage resulting from stolen passwords but users must take additional steps to obtain its benefits. Since the use of the token server is optional, users may choose to forego the added security in favor of ease of use. While high restrictions applied to all tokens such as a short lifetime and one-time use may provide increased security, they also make the token authentication process more difficult for the average user. Alternatively, appeasing the user by placing few restrictions on token use decreases security. Finding a single, hard coded set of rules that will satisfy all users and provide the level of security necessary is not practical.
Different environments lead to different needs. Users in a network community are not all alike. At Miami there are students, graduate students, faculty, and various staff members affiliated with the university; each has different computing needs, and they use different network resources. Creating the same rules for all users is not a viable option. Some users may require higher levels of security and thus more restrictive token properties. For example, a student may be allowed to use the same token five times, whereas a faculty member may be limited to two uses per token.

3.3 Token Policy Language

To maximize the LTPP system’s applicability, I have created a token policy language that allows each site to customize the LTPP’s rules. If all tokens were created and maintained with the same properties, then the token server could be designed with such built-in functionality. However, the necessity to create rules that relate to each user’s individual security needs stipulates that rules for token creation and validation be specified outside the realm of the token server. I have defined an LTPP token policy language and translator to express and implement the different rules necessary for the creation and validation of tokens by the token server.

In general, a policy is a plan, or set of rules, that specifies a course of action to take given the current state of the system and the requested operation. A token server administrator creates a policy by specifying the rules in my token policy language. Software tools then convert the policy instance into a set of actionable procedures that work with the token server to govern the validation and utilization of tokens. Specifically, the tools translate the policy into Java code that is compiled and executed by the LTPP.

There are two types of rules in the token policy language; the rules defined in the policy allow or deny the creation and validation of tokens, and they permit the further restriction of token use by explicitly limiting the range of token properties. The policy rules include many restriction-based security questions including the following: How long should the token be valid? How many times can the token be used? Are there restrictions on the dates and/or times for which a token can be issued? How many tokens can be issued to a user at any given time?

3.4 LTPP Policy Processing

The token server policy controls the issuance and validation of tokens within the context of the LTPP. The policy defines the checks that the LTPP applies to token requests. The policy translator converts a policy written in the token server policy language to Java code that is integrated into the LTPP system. When the LTPP receives a request for a new token, it first checks the credentials by submitting a bind request to the LDAP server (Figure 7). If the bind is successful, the token server creates/updates the database record for the user and then checks its policy to determine if the token can be issued. Since properties of the token are set/used by the policy, the token server instantiates a token object before its issue status is known. If the policy allows the issuance of the token, the token server checks the newly set token properties to ensure that all constraints are satisfied. The token object and password are then stored in the user record. A token creation request that is rejected by the policy or the token constraints does not result in a stored token in the database.
The token validation process begins when the user attempts to authenticate himself to a service by submitting his username/token pair (Figure 8). Since the authentication request might or might not be using a token string, the token server first checks to see if it has the username/token string pair stored in the database. If the username/token string pair is found in the database, the token server checks the policy to find out if the token can be validated. The token server also ensures that validating the token will not violate any of the token properties such as the lifetime of the token. If the token passes both tests, the token portion of the bind request is replaced with the user’s password. Finally, the token server sends the updated bind request to the LDAP server. If the username/token pair is not found in the database, the token server forwards the bind request unchanged to the LDAP server. The result of either bind request is then forwarded to the service.
The addition of the token policy language and translator to the LTPP enhances the security provided through token authentication. This language provides the token server administrator with the mechanism to create flexible policies that are more suited to the environment. The administrator can use the language to create a policy tailored to the needs of the network users and the environment; in this way, the administrator can define a policy that achieves an acceptable balance between trust and security. In the next section, I present the token policy language and its implementation.
4. Token Server Authentication Policy Language

4.1 Design

One metric of a policy language depends upon the ease with which a policy can be written. According to [LIN 00], “writing a policy should be as easy as describing what it is.” The author of a policy should be confident of its effects on the system. These principles are adhered to in the design of the token server policy language.

The purpose of a token server authentication policy is twofold; the policy controls or defines under which circumstances tokens can be created or validated, and the policy allows restrictions to be placed on token use. Since creating and validating tokens are two distinct processes, a special rule type is defined for each. Creation rules begin with the keyword, `cRule`, and are applied when a user seeks to obtain a token from the token server. Similarly, validation rules begin with the keyword, `vRule`, and are relevant when a user attempts to gain authentication with a service by supplying his username/token pair. The simplest rule is the `default` rule; this rule sets the default values of the token properties. These three rules types comprise the entire policy language.

These rules use the token’s properties, provided by the token server, and system properties in order to make decisions and set restrictions on token use. Token properties describe the token user or the token itself. Those properties that are implicitly set by the token server cannot be altered by the policy rules. Likewise, properties explicitly set by the policy rules are only assigned initial values by the token server. Neither the token server nor the policy can set the system properties.

<table>
<thead>
<tr>
<th>Token Properties</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicitly Set by Token Server</td>
<td></td>
</tr>
<tr>
<td><code>username</code></td>
<td>A unique identifier for the token user</td>
</tr>
<tr>
<td><code>group</code></td>
<td>Zero or more groups to which the user belongs</td>
</tr>
<tr>
<td><code>userIP</code></td>
<td>The IP address from which a user requests a token</td>
</tr>
<tr>
<td><code>serviceIP</code></td>
<td>The IP address of the service seeking username/token authentication</td>
</tr>
<tr>
<td><code>tokenType</code></td>
<td><code>{sync, normal}</code></td>
</tr>
<tr>
<td></td>
<td>The type of token: normal or synchronized</td>
</tr>
<tr>
<td>Explicitly Set by Token Server According to Policy Rules</td>
<td></td>
</tr>
<tr>
<td><code>numUses</code></td>
<td>The number of times that a token can be used for validation</td>
</tr>
<tr>
<td><code>lifetime</code></td>
<td>The length of time that a token is valid</td>
</tr>
<tr>
<td><code>minLength</code></td>
<td>The minimum number of characters in the token</td>
</tr>
<tr>
<td><code>maxInUse</code></td>
<td>The maximum number of tokens that a user can have at any given time</td>
</tr>
<tr>
<td>System Properties</td>
<td></td>
</tr>
<tr>
<td><code>date</code></td>
<td>The date during which a token can/cannot be created/validated</td>
</tr>
<tr>
<td><code>time</code></td>
<td>The time during which a token can/cannot be created/validated</td>
</tr>
</tbody>
</table>

Table 1: Token Server Properties
Each rule shares the same basic structure. A rule begins with a declaration of the rule type, and braces enclose the remainder of the rule statement. The rule statement of creation rules and validation rules has a tri-part substructure: the match, issue/validate, and limit statements. The match statement determines if the rule is applicable to the token request. The issue/validate statement affects the LTTP’s decision to issue/validate a token. The limits statement allows the policy author to set bounds on the token properties.

| CRULE{  
|     MATCH: conditions;  
|     ISSUE: decision;  
|     LIMIT: restrictions;  
| } |

| VRULE{  
|     MATCH: conditions;  
|     VALIDATE: decision;  
|     LIMIT: restrictions;  
| } |

**Table 2: Basic Structure of Creation and Validation Rules**

Using the validation rule structure shown in Table 2, a policy author can define a validation rule that restricts authentication with a particular service based on the location (inside or outside the environment) from which the token was obtained and the groups to which the user belongs. For example, a rule could require that administrators authenticate with an administrative service from within the same IP subnet. Another rule could strengthen token restrictions if the service is outside the environment. These restrictions could be applied based on the security level of the user’s group affiliations.

The match statement begins with the keyword, `MATCH`, followed by a colon and one or more conditions. Conditions generally have the form, `property compareOp ‘value’`. Creation rules can use the following properties in the match statement: username, group, userIP, tokenType, date, and time. Since the serviceIP is not yet known when a user requests a token, this property is only available in the match statement of validation rules. In most cases, compareOp is an equality operator (= or !=). For the date and time properties, compareOp can also be one of the inequality operators (>, >=, <, <=). Non-numeric values are always enclosed in single quotes.

The date and time properties have a special `RANGES` keyword that simplifies the way that intervals are expressed in match conditions. A range is specified in one of two ways: with the keyword operators `INCL` and `EXCL` or with parentheses and brackets to represent inclusion and exclusion respectively. The basic form of a match condition with an interval is shown below; the two match conditions are equivalent in meaning. Any of the four combinations of range operators can be used when specifying an interval of either form; however, the two types of operators cannot be used together.

```
MATCH: property RANGES ‘value’ INCL TO ‘value’ EXCL;
MATCH: property RANGES (‘value’, ‘value’);
```

**Table 3: Basic Form of Interval Conditions**

More complex match statements are formed by combining conditions with the Boolean operators `AND` and `OR`. In the token server policy language, the comma has the same meaning as the `AND` operator; it can be used to make conditional statements more concise. Order of precedence in match statements with multiple conditions is obtained through the use of parentheses.
MATCH: (condition AND condition) OR condition;
MATCH: condition, condition, condition;

Table 4: Complex Match Statements

The issue/validate statement determines when the LTPP makes a decision for a token’s issuance or validation. Since the creation rule controls the issuance of a token, the syntax of this statement is the keyword, ISSUE, followed by a colon and the decision. Alternatively, validation rules use the keyword, VALIDATE, as they apply only to existing tokens. This section controls whether to issue/validate the token immediately (NOW), to mark the token request as OK, or to reject the token request (NO) as shown in Table 5. A request that is marked with OK may be evaluated by multiple rules until it has been issued/validated with NOW, rejected with NO, or there are no more rules to evaluate. When all rules have been assessed, if the request is marked as OK, the token is then issued/validated. A rule that does not include a decision statement implies OK.

<table>
<thead>
<tr>
<th>Decision</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOW</td>
<td>Issues/validates immediately; remaining rules are not evaluated</td>
</tr>
<tr>
<td>OK</td>
<td>Neither issues/validates nor rejects; evaluate remaining rules</td>
</tr>
<tr>
<td>NO</td>
<td>Rejects token request; do not evaluate remaining rules</td>
</tr>
</tbody>
</table>

Table 5: Issue/Validate Statement Effects

The limit statement allows the policy writer to place restrictions on the token properties. This statement begins with the keyword, LIMIT, followed by a colon and one or more restrictions. The general form of a restriction is property = value. This restrictive portion of a rule is limited to the numeric token properties. Creation rules can set any of the token properties listed in the table below; however, validation rules cannot alter the minLength and maxInUse properties. Allowing changes to minLength during validation would require changing the token string. Altering maxInUse during validation could conflict with the number of existing tokens. Since properties are modifiable by multiple rules, each property has a direction of change that specifies how it affects the property. For example, the lifetime of a token can only decrease; in this way, one rule cannot override a restriction placed by another rule except to strengthen the restriction. Rules are not required to set token restrictions, so the restriction statement may be excluded.

Table 6: Token Properties for Restriction

<table>
<thead>
<tr>
<th>Property</th>
<th>Direction of Change</th>
<th>cRule</th>
<th>vRule</th>
</tr>
</thead>
<tbody>
<tr>
<td>numUses</td>
<td>decrease</td>
<td>yes</td>
<td>Yes</td>
</tr>
<tr>
<td>lifetime</td>
<td>decrease</td>
<td>yes</td>
<td>Yes</td>
</tr>
<tr>
<td>minLength</td>
<td>increase</td>
<td>yes</td>
<td>no – cannot change the length of an existing token</td>
</tr>
<tr>
<td>maxInUse</td>
<td>decrease</td>
<td>yes</td>
<td>no – cannot reduce the number of existing tokens</td>
</tr>
</tbody>
</table>

Table 7 provides examples of basic creation and validation rules. In each rule the match section must evaluate to true for the remainder of the rule to apply to the token request. Given the match condition, group != ‘faculty’, the example creation rule affects only those users who are not
faculty members. The decision statement, **ISSUE: OK**, indicates that other rules should be checked before issuing the token. Lastly, the statement, **LIMIT: minLength = 5**, sets the minimum length of the token to be five characters. The validation rule approves token use for user, buschrl1, and sets the lifetime of the token to 180 seconds. If this validation rule applies to the token request, no other rules are checked since the validate statement is **NOW**.

```plaintext
CRULE{
    MATCH: group != 'faculty';
    ISSUE: OK;
    LIMIT: minLength = 5;
}
VRULE{
    MATCH: username = 'buschrl1';
    VALIDATE: NOW;
    LIMIT: lifetime = 180;
}
```

Table 7: Simple Creation and Validation Rules

Just as match conditions can combine to create complex match statements, rules of the same type can also be joined together to create more expressive rules. The basic rule has the construct, `ruleType {simpleRuleStmt}`. A complex rule statement is comprised of multiple simple rule statements and one or more of the operators: **IF/THEN**, `{}`, **AND**, and **OR**. Unlike match statements, rule statements cannot be parenthesized to enforce order of precedence since rule statements involve more than Boolean conditions. The table below details the ways in which rule statements can combine; note that unless specified, a rule statement is either simple or complex.

<table>
<thead>
<tr>
<th>Rule Statement Combinations</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF simpleRuleStmt THEN {ruleStmt2}</td>
<td>ruleStmt2 is only evaluated if simpleRuleStmt applies to the token request</td>
</tr>
<tr>
<td>simpleRuleStmt {ruleStmt2}</td>
<td>equivalent to <code>IF simpleRuleStmt THEN {ruleStmt2}</code></td>
</tr>
<tr>
<td>ruleStmt1 AND ruleStmt2</td>
<td>both ruleStmt1 and ruleStmt2 will be evaluated</td>
</tr>
<tr>
<td>ruleStmt1 OR ruleStmt2</td>
<td>ruleStmt2 is evaluated only if ruleStmt1 does not apply to the token request</td>
</tr>
<tr>
<td>ruleStmt1 OR partialRuleStmt</td>
<td>partialRuleStmt is evaluated only if ruleStmt1 does not apply to the token request; a partialRuleStmt is a simpleRuleStmt without a match statement</td>
</tr>
</tbody>
</table>

Table 8: Rule Statement Combinations and Meanings

Table 9 shows an expressive validation rule that takes advantage of rule combination. This validation rule is comprised of three individual rule statements in the form `VRULE{simpleRuleStmt{ruleStmt OR ruleStmt}}`. The match condition of the simple rule statement must be true for the token request for the remainder of the rule to be evaluated. This validation rule applies to all service requests that are not made from the Miami University campus. The lifetime of these off-campus tokens is reduced to 90 seconds. If a faculty member seeks token authentication with an off-campus service, the faculty member will not be allowed to use this token again since its number of uses is reduced to one. Similarly, a student seeking token authentication with an off-campus service will only be allowed to use the token twice more.
This example validation rule demonstrates the effectiveness and reduction in redundancy that is achieved by creating complex rules. Table 10 shows the same validation rule as three separate rules. Dividing the validation rule into its three component rules requires that the serviceIP be checked in each rule. Not only can this repetition lead to typographical errors, but it also decreases the readability of the policy.

VRULE{
    MATCH: serviceIP != '134.53.0.0/16';
    LIMIT: lifetime=90;
    {
        MATCH: group = 'faculty';
        LIMIT: numUses=1;
        OR
        MATCH: group = 'student' OR group = 'graduate student';
        LIMIT: numUses=3;
    }
}

Table 10: Complex Validation Rule Expressed as Multiple Simple Rules

Unlike creation and validation rules, the default rule applies to all token requests and only allows restrictions to be set. Further limits placed by creation and validation rules just serve to strengthen the restrictions set by the default rule. The default rule can restrict all properties listed in Table 6. An example default rule is shown in Table 11; since the direction of change for token lifetime is decreasing, no creation or validation rule can set the lifetime of a token to be more than 3600 seconds.

DEFAULT{
    LIMIT: minLength=4, maxInUse=10, lifetime=3600, numUses=25;
}

Table 11: Example Default Rule
A token server policy is a text file. The policy language is case insensitive; keywords are accepted in any combination of upper and lower case characters. The format of the policy file is as follows: `default? cRule* vRule*`. If a default rule is included in the policy, it must be the first rule; only one default rule is allowed per policy. Zero or more creation rules follow the default rule if it exists. Finally, the policy ends with zero or more validation rules.

The LTPP never evaluates creation and validation rules together for a token request. The creation rules are checked when a user attempts to obtain a token, and the validation rules are checked when a user tries to use an existing token. Thus, conflicts between the decision to issue a token and the decision to validate a token do not arise. However, decision conflicts may occur between rules of the same type.

The decision to issue or validate a token is controlled by the keywords: `OK`, `NOW`, or `NO`. The default decision value, `OK`, is essentially a maybe answer. A matched rule with a `NOW` or `NO` unequivocally grants or rejects the token request. It is possible to create a policy that potentially grants a token request with one rule and denies the same request with another. The token policy language resolves this conflict by giving precedence to rule order. A rule that accepts a request with `NOW` or rejects a request with `NO` prevents any remaining rules from being evaluated. The expected number of rules that explicitly grant or deny token requests in a policy is few. For this reason, the policy writer should easily be able to recognize potential conflicts and appropriately order the rules.

### 4.2 Implementation

A policy written in the token policy language is intended for humans. Since the LTPP system is implemented in the Java programming language, we have chosen to integrate the policy into the LTPP system by translating its rules into Java code as shown in Figure 9. We considered interpreting the rules but chose to compile them for efficiency.

![Figure 9: Policy Integration into LTPP](image)

Of the many tools available for language translation, we chose to use ANother Tool for Language Recognition (ANTLR). ANTLR is a straightforward framework in which one can build a translator from a grammar description. Noteworthy features of ANTLR include the ability to intersperse Java code within the grammar definition and its support for abstract syntax tree construction and walking to aid in the translation process. [ANTLR 04]
The translator is implemented in ANTLR by creating a grammar file in which the lexer, parser, and tree walker are defined. The keywords and elements that contribute to the policy language are identified in the lexer. The structure of the language is formed in the parser by combining lexemes (tokens) provided by the lexer into structural components of the language. Special characters in the parser, the caret and the exclamation point, are used with tokens to determine the structure of the abstract syntax tree that the parser creates. The caret indicates that a token should be a parent node in the tree, whereas the exclamation point causes a token to be excluded from the tree. Unmarked tokens become children nodes in the tree. A well-organized tree is crucial to the translation process. Table 12 shows the abstract syntax tree that results when the example creation rule is parsed. The final component of the grammar, the tree walker, uses the abstract syntax tree created by the parser to perform the translation.

```java
CRULE{
    MATCH: group='faculty';
    ISSUE: OK;
    LIMIT: minLength=6, lifetime=300;
}
```

**Table 12: Example cRule and Resulting Abstract Syntax Tree**

ANTLR supports the inclusion of Java code with the grammar, a feature that is very valuable. I created Java functions in the parser to provide extra error checking. For example, when the match condition involves the time property, the parser calls a function to ensure that the time value given in the policy is formatted correctly. Table 13 shows the errors produced by the parser when invalid time values are given. Catching such errors during the translation process prevents their unexpected occurrence when the policy is actually used by the token server. In the tree walker, Java code simplified the translation process; for example, the code produced by a match condition depends on the property that appears in the condition. When the tree walker analyzes a match condition, it calls a function that supplies the correct translation for the given property.
MATCH: time RANGES ('5:30','7:45');

<table>
<thead>
<tr>
<th>Exception at line: 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid time value '5:30'</td>
</tr>
<tr>
<td>Proper Format: 'hh:mm (AM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exception at line: 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid time value '7:45'</td>
</tr>
<tr>
<td>Proper Format: 'hh:mm (AM</td>
</tr>
</tbody>
</table>

Table 13: Example Error Statements Produced During Parsing

The grammar for the token policy language established in the design process defines the acceptable format for a token server policy. Policies that do not adhere to the grammar cannot undergo translation into working Java code. Compiling the grammar file with ANTLR creates Java files for the lexer, parser, and tree walker. A fourth file defines all lexemes used in the grammar. The final step in constructing the translator is to create a main Java file that calls the lexer, parser, and tree walker in succession on a given input policy file.

The code produced by the translator is a complete Java class with three methods: `setTokenDefaults`, `checkCRules`, and `checkVRules` which are part of the LTPP. Each method takes a token object as a parameter and performs the appropriate checks for that token; the `checkCRules` and `checkVRules` methods return a Boolean decision value. The token server calls these functions during the token creation and validation processes and supplies each with the token object of the token request. The token object has methods that provide access to its properties and methods that allow the policy to set certain properties. In this way, the policy file produced by the translation process integrates fully into the LTPP system.
5. Testing and Analysis

To evaluate the token language and the policy code produced by the translation process, I created tests to study the policy’s effects on token creation and validation. These tests check that the actual observed results match our expected results. Since all possible policy instances cannot be tested, we developed two scenarios of use for which policies were written and tested. Both scenarios involve a central authentication server and multiple services administered by various users. The larger-scale scenario is a university environment; this scenario includes administrative services, grades services, faculty workstations, research machines, student projects, and off-campus services. The second scenario is based on a small business environment.

From a technical perspective, testing ensures that a policy operates as specified. We assert that a test is successful if the effects of the policy on token creation and validation match those that we expect. Along with technical correctness, we desire to show that the policy language exhibits ease of use, readability, and high expressivity. However, we cannot demonstrate these subjective properties via the same means with which we establish technical correctness.

In testing, we cannot evaluate a policy on a rule by rule basis; one or more rules may apply to a given token request. The properties set by one rule limit the degree to which other rules can set them. A definitive issue/validate decision for a token request also prevents later rules from being evaluated. Thus, we test the entire policy on a variety of token requests.

We used the JUnit testing framework to test the policy language. [JUNIT 04] This software includes simple assertion tests to discover if an expected result is the same as the actual result (assertEquals), if an object is null (assertNull, assertNotNull), and if a Boolean result is correct (assertTrue, assertFalse). By knowing the expected result of an action, one can assert the result to be correct. We need only set up each test and declare an expected result through an assertion test. A test case file can include multiple tests. When the test file is executed, failures of any assertion tests are listed. If all tests are successful, then our expected policy results match the actual results; the policy instance is deemed correct to the degree in which we test it.

To complete the scenario testing, we set up an LDAP server with entries for the various types of users in each scenario. For example, users at a university may be classified as students, graduate students, staff, faculty, administrators, etc. They may also belong to particular departments or groups within the university such as the systems analysis (SAN) department. In the university scenario, there are twelve LDAP entries corresponding to users of the various classes and their membership status in the SAN department. The small business scenario includes five users belonging to the groups: administrator, president, secretary, and developer.

The policy developed for the university scenario is shown in the appendix, section 8.2. The creation rules set limits based on the groups to which the token user belongs and/or the type of token being requested. In accordance with the policy, the LTPP system denies token creation requests from user, badmac, or userIP, 24.210.230.215. Table 14 shows the results of seven token creation tests. For each test the tokenType is normal, and the userIP is not 24.210.230.215. If the token creation request was successful, the token properties are shown in the gray expected
results section; otherwise no token was issued. The column, Matches Test Result?, indicates whether the expected results were the same as the actual test results. We ran similar set of tests where all input properties were the same except the userIP was 24.210.230.215. In all of these tests, the token creation request was denied, a result that we expected.

<table>
<thead>
<tr>
<th>Username</th>
<th>Groups</th>
<th>Expected cRule Result</th>
<th>Matches Test Result?</th>
</tr>
</thead>
<tbody>
<tr>
<td>buschmm</td>
<td>courtesy</td>
<td>minLength: 4, lifetime: 7200, numUses: 60, maxInUse: 12</td>
<td>yes</td>
</tr>
<tr>
<td>buschrl1</td>
<td>student</td>
<td>minLength: 4, lifetime: 7200, numUses: 50, maxInUse: 12</td>
<td>yes</td>
</tr>
<tr>
<td>jeremis</td>
<td>faculty</td>
<td>minLength: 6, lifetime: 1200, numUses: 20, maxInUse: 12</td>
<td>yes</td>
</tr>
<tr>
<td>deanv</td>
<td>administrator</td>
<td>minLength: 8, lifetime: 600, numUses: 10, maxInUse: 12</td>
<td>yes</td>
</tr>
<tr>
<td>marchad</td>
<td>graduate student</td>
<td>minLength: 4, lifetime: 7200, numUses: 50, maxInUse: 12</td>
<td>yes</td>
</tr>
<tr>
<td>middenh</td>
<td>staff</td>
<td>minLength: 6, lifetime: 1200, numUses: 20, maxInUse: 12</td>
<td>yes</td>
</tr>
<tr>
<td>badmac</td>
<td>student</td>
<td>no token</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Table 14: University Scenario - Expected and Actual Token Creation Results**

The validation rules of the university policy focus on granting or denying authentication based on the target network service of the token use request. Within these rules, other conditions are specified to further limit the effects of the rule on the token population. Of the seven properties that can be used in the match statement of a validation rule, only four were utilized in the university policy. Three of these properties, group, userIP, and serviceIP, appear as columns in Table 15. The fourth property, time, is indicated in the table where necessary as it is only a factor in a few test cases. The expected result section only includes lifetime and numUses since the validation rules can only set these properties. The third expected result column is dedicated to the validation decision. Each test case involves the immediate validation of a newly created token. When a token request is validated, the LTTP system decrements the numUses value as shown in Table 15.
<table>
<thead>
<tr>
<th>groups</th>
<th>userIP</th>
<th>serviceIP</th>
<th>Expected vRule Result</th>
<th>Matches Test Result?</th>
</tr>
</thead>
<tbody>
<tr>
<td>courtesy</td>
<td>134.53.12.13</td>
<td>134.53.10.3</td>
<td>7200</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.9.23</td>
<td>7200</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.7.23</td>
<td>7200</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.23.1</td>
<td>7200</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.25.12</td>
<td>600</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.59.30.12</td>
<td>1200</td>
<td>4</td>
</tr>
<tr>
<td>student</td>
<td>134.53.15.34</td>
<td>134.53.10.3</td>
<td>7200</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.9.23</td>
<td>7200</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.7.23</td>
<td>7200</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.23.1</td>
<td>7200</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.25.12</td>
<td>600</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.59.30.12</td>
<td>1200</td>
<td>4</td>
</tr>
<tr>
<td>faculty</td>
<td>134.53.15.55</td>
<td>134.53.10.3 (8am to 11pm)</td>
<td>1200</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.9.23 (11pm to 8am)</td>
<td>1200</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.7.23</td>
<td>1200</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.23.1</td>
<td>1200</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.25.12</td>
<td>600</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.59.30.12</td>
<td>1200</td>
<td>0</td>
</tr>
<tr>
<td>administrator</td>
<td>134.53.7.55</td>
<td>134.53.10.3 (8am to 11pm)</td>
<td>600</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.9.23 (11pm to 8am)</td>
<td>600</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.7.23</td>
<td>600</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.23.1</td>
<td>600</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.25.12</td>
<td>600</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.59.30.12</td>
<td>600</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>134.53.19.23</td>
<td>134.53.7.23</td>
<td>600</td>
<td>10</td>
</tr>
<tr>
<td>graduate-student</td>
<td>134.53.29.15</td>
<td>134.53.10.3</td>
<td>7200</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.9.23</td>
<td>7200</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.7.23</td>
<td>7200</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.23.1</td>
<td>7200</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.25.12</td>
<td>600</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.59.30.12</td>
<td>1200</td>
<td>4</td>
</tr>
<tr>
<td>staff</td>
<td>134.53.2.1</td>
<td>134.53.10.3</td>
<td>1200</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.9.23</td>
<td>1200</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.7.23</td>
<td>1200</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.23.1</td>
<td>1200</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.53.25.12</td>
<td>600</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.59.30.12</td>
<td>1200</td>
<td>4</td>
</tr>
<tr>
<td>faculty/SAN</td>
<td>134.53.112.1</td>
<td>134.53.10.3</td>
<td>1200</td>
<td>19</td>
</tr>
<tr>
<td>administrator/SAN</td>
<td>134.53.112.2</td>
<td>134.53.23.1</td>
<td>1200</td>
<td>20</td>
</tr>
<tr>
<td>staff/SAN</td>
<td>134.53.112.3</td>
<td>134.53.25.12</td>
<td>600</td>
<td>4</td>
</tr>
<tr>
<td>student/SAN</td>
<td>134.53.112.4</td>
<td>134.53.25.12</td>
<td>7200</td>
<td>50</td>
</tr>
<tr>
<td>graduate-student/SAN</td>
<td>134.53.112.5</td>
<td>134.53.25.12</td>
<td>7200</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 15: University Scenario - Expected and Actual Token Validation Results
Section 8.3 shows the policy developed for the small business scenario. Since there are few users, the policy does not include creation rules; all tokens have the same properties as specified in the default rule. We only tested the validation rules for this policy. The validation rules for this scenario concentrate on the service for which a user seeks authentication. Unlike the university policy, most of the serviceIP addresses given in the policy are not subnet addresses because all services in the business are on the same subnet. Table 16 shows the results of the small business scenario tests. Just as with the university policy, all actual results matched the expected results.

<table>
<thead>
<tr>
<th>groups</th>
<th>userIP</th>
<th>serviceIP</th>
<th>lifetime</th>
<th>numUses</th>
<th>validate?</th>
<th>Matches Test Result?</th>
</tr>
</thead>
<tbody>
<tr>
<td>administrator</td>
<td>192.168.1.5</td>
<td>192.168.1.1</td>
<td>1200</td>
<td>0</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>192.168.1.9</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.1.5</td>
<td>192.168.1.6</td>
<td>1200</td>
<td>0</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.1.9</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.1.5</td>
<td>192.168.1.7</td>
<td>1200</td>
<td>9</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.2.23</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.2.23</td>
<td>192.168.1.10</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>president OR secretary</td>
<td>192.168.1.5</td>
<td>192.168.1.1</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>192.168.1.9</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.1.5</td>
<td>192.168.1.6</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>192.168.1.9</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.1.5</td>
<td>192.168.1.7</td>
<td>1200</td>
<td>9</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.2.23</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.2.23</td>
<td>192.168.1.10</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>developer</td>
<td>192.168.1.5</td>
<td>192.168.1.1</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>192.168.1.9</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.1.5</td>
<td>192.168.1.6</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>192.168.1.9</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.1.5</td>
<td>192.168.1.7</td>
<td>1200</td>
<td>9</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.2.23</td>
<td>1200</td>
<td>10</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>192.168.2.23</td>
<td>192.168.1.10</td>
<td>1200</td>
<td>9</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>

Table 16: Small Business Scenario - Expected and Actual Token Validation Results

Each test case for the scenarios creates a token and subsequently attempts to validate the token with the token server. These tests are not designed to determine if the token server correctly checks the token properties set by the policy prior to actual token creation and validation. Before allowing a token to be created, the token server must ensure that the maxInUse property will not be violated by creating the token. Similarly, when validating a token use request, the token server must make sure that the lifetime and numUses properties will not be violated if the request is granted. We created a separate and simplified test set for these properties. The policy includes only the default rule in which the properties are initially set. In this way, the testing process is simplified; the policy allows all creation and validation requests. Only a violation of the maxInUse, numUses, or lifetime properties causes the denial of a request by the token server.

Technical testing via JUnit does not evaluate the subjective properties of the policy language, which include ease of use, readability, and high expressivity. One aspect of ease of use is that a
policy author can begin to create policies with little instruction. Though different, this property goes hand-in-hand with *readability*; the policy author should be able to understand a policy no matter how much time has passed since he wrote it. The need to satisfy these properties drove the policy language design process. Our goal is that a policy written in the token policy language closely resembles its natural language counterpart; in this way, the author can structure the policy in the token policy language similarly to its natural language counterpart. Table 17 shows an example rule from the university policy; the policy written in natural language is quite similar to the policy written in the policy language in both content and structure. The final property, *expressivity*, gauges the ability of the policy language to represent any token server policy. The combinatorial match conditions and rule statements of the policy language contribute greatly to its expressivity.

<table>
<thead>
<tr>
<th>Anyone can authenticate with student projects (subnet 134.53.25.0/24), but token use will be restricted:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Reduce numUses to 5 and lifetime to 600 for all tokens</td>
</tr>
<tr>
<td>• If token belongs to faculty or administrator, reduce numUses of token to 1 (this use)</td>
</tr>
</tbody>
</table>

```plaintext
VRULE{
  MATCH: serviceIP='134.53.25.0/24';
  LIMIT: numUses=5, lifetime=600;
  {
    MATCH: group='faculty' OR group='administrator';
    LIMIT: numUses=1;
  }
}
```

**Table 17: Example Validation Rule for Subjectivity**

We believe this policy language exhibits all three subjective properties; however, in creating the test scenarios, we discovered possible language improvements. These include being able to specify the lifetime value in minutes, seconds, or hours, and being able to create negated match conditions. The addition of a token property to test for the day of the week could also simplify policy writing. Even without these additional improvements, the token policy language can still express the lifetime and negative match statements. We can currently represent a negated match statement in one of two ways: using de Morgan’s rules or creating a complex rule using the *OR* operator.
6. Conclusion

In this work, I have presented my design and implementation of a token server policy language and have shown how the token policy language supports the creation of simple and straightforward policies that enhance the security provided by the token server. Policies written in this language are easily adapted to suit any token server environment. I created an ANTLR translator that converts the policy to Java code, which is then integrated with the token server. Through testing, I have demonstrated the operation and implementation of several sample token server policies.

This work supports Dr. Campbell’s LTPP project that addresses the problem of compromised passwords in a single-site authentication environment. The LTPP allows a user to obtain a temporary token from the token server over a secure and trustworthy connection by supplying his username/password pair. Upon successful authentication, the token server sends a temporary token string to the user. The user may then replace his password with the token string when attempting to authenticate with a service.

While authentication through the token server prevents a user’s password from being stolen by a service, the risk that his temporary token might be compromised does exist. The transient nature of the token provides safeguards from extensive use if stolen, but an imposter may still be able to inflict irreparable damage upon the user’s resources. My token policy language addresses this issue by allowing for the creation of policies that place restrictions on the issuance and usage of the token to prevent its compromise.

A common set of token restrictions does not effectively address the different security levels required by all users. It is necessary to create tokens that tailor to the security requirements of the individual user. This work presents a policy language designed specifically for token server authentication. This language allows for the creation of concise policies to control the token creation and validation processes in an effort to reduce the possibility of token compromise. A policy defined by this language cannot guarantee the complete security of the token server, but if well defined, it can greatly increase the level of security provided by the token server.

Further development of the token policy language may expand its usefulness in single-site authentication. Rather than restricting the policy rules to operate only on token requests, the rules could also regulate username/password bind requests. This would enable the LTPP to function as a general purpose authentication tool; for example, a policy rule could prevent students from logging into faculty machines.

In conclusion, I have developed, implemented, and tested a token policy language that enhances the token authentication process. Though the risk of having one’s token compromised could easily be avoided by creating one-time use tokens, few users would go through the hassle of obtaining a new token for authentication with each service. The token policy language allows the token server administrator to define rules that effectively govern the token creation and validation processes without ignoring the needs of the various users.
7. References


8. Appendix

8.1 Grammar for Token Server Authentication Policy Language

\[
policy: (defaultRule)? (createRule)* (validateRule)*
\]

\[
defaultRule: DEFAULT LBRCASE LIMIT COLON devaultVals SEMI RBRCASE
\]

\[
defaultVals: NUM_VAR EQ NUM_VAL ((AND|COMMA) defaultVals)?
\]

\[
createRule: CRULE LBRCASE cRule RBRCASE
\]

\[
cRule: cIfThenStmt AND cRule
| cIfThenStmt OR (cRule|partialCRule)
| cRuleStmt AND cRule
| cRuleStmt OR (cRule|partialCRule)
| cRuleStmt
\]

\[
cIfThenStmt: IF cRuleStmt THEN LBRCASE cRule RBRCASE
| cRuleStmt LBRCASE cRule RBRCASE
\]

\[
cRuleStmt: cMatchStmt (issueDecision)? (cLimitToken)?
| cLimitToken
\]

\[
partialCRule: issueDecision (cLimitToken)?
| cLimitToken
\]

\[
cMatchStmt: MATCH COLON cMatchCond SEMI
\]

\[
cMatchCond: LPAREN cMatchCond RPAREN ((AND|OR|COMMA) cMatchCond)?
| cCompareStmt ((AND|OR|COMMA) cMatchCond)?
\]

\[
cCompareStmt: C_RULE_MATCH_VAR (EQ|NOT_EQ) STRING_VAL
| DATE_VAR ((RANGES range) | (NUM_OP STRING_VAL))
| TIME_VAR ((RANGES range) | (NUM_OP STRING_VAL))
\]

\[
issueDecision: ISSUE COLON DECISION SEMI
\]

\[
cLimitToken: LIMIT COLON cLimitVars SEMI
\]

\[
cLimitVars: cLimitCompare ((AND|COMMA) cLimitVars)?
\]

\[
cLimitCompare: C_RULE_LIMIT_VAR EQ NUM_VAL
\]

\[
validateRule: VRULE LBRCASE vrule RBRCASE
\]

\[
vRule: vIfThenStmt AND vRule
| vIfThenStmt OR (vRule|partialVRule)
| vRuleStmt AND vRule
| vRuleStmt OR (vRule|partialVRule)
| vRuleStmt
\]

\[
vIfThenStmt: IF vRuleStmt THEN LBRCASE vRule RBRCASE
| vRuleStmt LBRCASE vRule RBRCASE
\]
vRuleStmt: vMatchStmt (validateDecision)? (limitToken)?

partialVRule: validateDecision (vLimitToken)?
   | vLimitToken

vMatchStmt: MATCH COLON vMatchCond SEMI

vMatchCond: LPAREN vMatchCond RPAREN ((AND|OR|COMMA)vMatchCond)?
   | vCompareStmt ((AND|OR|COMMA)vMatchCond)?

vCompareStmt: V_RULE_MATCH_VAR (EQ|NOT_EQ) STRING_VAL
   | DATE_VAR ((RANGES range)|(NUM_OP STRING_VAL))
   | TIME_VAR ((RANGES range)|(NUM_OP STRING_VAL))

validateDecision: VALIDATE COLON DECISION SEMI

vLimitToken: LIMIT COLON vLimitVars SEMI

vLimitVars: vLimitCompare ((AND|COMMA) vLimitVars)?

vLimitCompare: V_RULE_LIMIT_VAR EQ NUM_VAL

range: STRING_VAL RANGE_OP TO STRING_VAL RANGE_OP
   | (LPAREN|LBRACKET) STRING_VAL COMMA STRING_VAL(LPAREN|LBRACKET)

NUM_OP: GE | GTE | LT | LTE | EQ | NOT_EQ
RANGE_OP: INCL | EXCL
DECISION: OK | NOW | NO

V_RULE_MATCH_VAR: USERNAME | USER_IP | SERVICE_IP | GROUP | TOKEN_TYPE
C_RULE_MATCH_VAR: USERNAME | USER_IP | GROUP | TOKEN_TYPE

V_RULE_LIMIT_VAR: NUMUSES | LIFETIME
C_RULE_LIMIT_VAR: NUMUSES | LIFETIME | MAX_IN_USE | MIN_LENGTH

*The complete grammar is available in the LTPP project directory.*
## 8.2 Policy for University Scenario

### University Policy

**Default token values:**
- `minLength=4`, `lifetime=7200` (2 hours), `numUses=60`, `maxInUse=12`

**Limit normal tokens by group membership:**
- students and graduate students – `numUses=50`, `lifetime=20`
- faculty and staff – `minLength=6`, `lifetime=1200`, `numUses=20`
- administrators – `minLength=8`, `lifetime=600`, `numUses=10`

**Limit all sync tokens to a lifetime of 3600 (1 hour) and 30 uses.**

**User badmac cannot obtain a token**

**Token creation requests cannot come from IP address 24.210.230.215**

**Faculty and administrators may authenticate with the grades service (134.53.10.3) from on-campus but only between 8AM and 11PM. Others may not gain authentication to the grades service.**

**Only administrators, faculty, and staff can use faculty workstations (subnet 134.53.9.0/24)**

### Code Examples

#### DEFAULT
```
DEFAULT{
  LIMIT: minLength=4, lifetime=7200,
  numUses=60, maxInUse=12;
}
```

#### CRULE
```
CRULE{
  MATCH: tokenType = 'normal';
  {
    MATCH: group='student' OR
group='graduate student';
    LIMIT: numUses=50, lifetime=20;
    OR
    MATCH: group='faculty' OR group='staff';
    LIMIT: minLength=6, lifetime=1200,
    numUses=20;
    OR
    MATCH: group='administrator';
    LIMIT: minLength=8, lifetime=600,
    numUses=10;
  }
  OR //sync token
  LIMIT: lifetime=3600, numUses=30;
}
```

#### CRULE
```
CRULE{
  MATCH: username='badmac' OR
  ISSUE: NO;
}
```

#### VRULE
```
VRULE{
  MATCH: serviceIP='134.53.10.3';
  {
    MATCH: (group='faculty' OR
group='administrator') AND
    userIP='134.53.0.0/16' AND
time RANGES ('8:00 AM','11:00 PM'];
    VALIDATE:OK;
    OR
    VALIDATE:NO;
  }
}
```

#### VRULE
```
VRULE{
  MATCH: serviceIP='134.53.9.0/24);
  {
    MATCH: group='faculty' OR group='staff' OR
    group='administrator';
    VALIDATE: OK;
    OR
    VALIDATE: NO;
  }
}
```
Administrators can only authenticate with administrative services if they are on the same subnet (134.53.7.0/24). All other users cannot authenticate with administrative services.

<table>
<thead>
<tr>
<th>VRULE{</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATCH: serviceIP='134.53.7.0/24';</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>MATCH: group='administrator',</td>
</tr>
<tr>
<td>userIP='134.53.7.0/24';</td>
</tr>
<tr>
<td>VALIDATE: OK;</td>
</tr>
<tr>
<td>OR</td>
</tr>
<tr>
<td>VALIDATE: NO;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Only SAN faculty may authenticate with the research machines (subnet 134.53.23.0/24)

<table>
<thead>
<tr>
<th>VRULE{</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATCH: serviceIP='134.53.23.0/24' AND</td>
</tr>
<tr>
<td>(group!='SAN' OR group!='faculty');</td>
</tr>
<tr>
<td>VALIDATE: NO;</td>
</tr>
</tbody>
</table>

Anyone can authenticate with student projects (subnet 134.53.25.0/24), but token use will be restricted:
- Reduce numUses to 5 and lifetime to 600 for all tokens
- If token belongs to faculty or administrator, reduce numUses of token to 1 (this use)

<table>
<thead>
<tr>
<th>VRULE{</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATCH: serviceIP='134.53.25.0/24';</td>
</tr>
<tr>
<td>LIMIT: numUses=5, lifetime=600;</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>MATCH: group='faculty' OR</td>
</tr>
<tr>
<td>group='administrator';</td>
</tr>
<tr>
<td>LIMIT: numUses=1;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Token restrictions must be strengthened when tokens are used with off-campus services (not on subnet 134.53.0.0/16).
- If token belongs to faculty or administrator, reduce numUses of token to 1 (this use)
- Reduce numUses to 5 and lifetime to 1200 for all tokens

<table>
<thead>
<tr>
<th>VRULE{</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATCH: serviceIP!='134.53.0.0/16';</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>MATCH: group='faculty' OR</td>
</tr>
<tr>
<td>group='administrator';</td>
</tr>
<tr>
<td>LIMIT: numUses=1;</td>
</tr>
<tr>
<td>OR</td>
</tr>
<tr>
<td>LIMIT: numUses=5, lifetime=1200;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
### 8.3 Policy for Small Business Scenario

<table>
<thead>
<tr>
<th>Small Business Policy</th>
<th></th>
</tr>
</thead>
</table>
| **Default token values:** | **DEFAULT**
  - minLength=4
  - numUses=10
  - lifetime=1200 (20 minutes)
  - maxInUse=5 |
  ```
  DEFAULT{
    LIMIT: minLength=4, numUses=10,
    lifetime=1200, maxInUse=5;
  }
  ```

| **Only the administrator may authenticate to gain access to the router (192.168.1.1) and file server (192.168.1.6).** | **VRULE**
| The administrator must be at the admin workstation (192.168.1.5) | ```
  VRULE{
    MATCH: serviceIP='192.168.1.1' OR serviceIP='192.168.1.6';
    {
      MATCH: group='administrator',
      userIP='192.168.1.5';
      VALIDATE: NOW;
      LIMIT: numUses=1;
      OR
      VALIDATE: NO;
    }
  }
  ```

| The administrator must be at the admin workstation (192.168.1.5) | **VRULE**
| Reduce numUses of token to 1 (one-time use) | ```
  VRULE{
    MATCH: serviceIP='192.168.1.1' OR serviceIP='192.168.1.6';
    {
      MATCH: group='administrator',
      userIP='192.168.1.5';
      VALIDATE: NOW;
      LIMIT: numUses=1;
      OR
      VALIDATE: NO;
    }
  }
  ```

| Anyone can authenticate with the application/web server (192.168.1.7), but only from the office (192.168.1.0/24) | **VRULE**
| | ```
  VRULE{
    MATCH: serviceIP='192.168.1.7' AND userIP!='192.168.1.0/24';
    VALIDATE: NO;
  }
  ```

| Only developers may authenticate with the DEV server (192.168.1.10) | **VRULE**
| | ```
  VRULE{
    MATCH: serviceIP='192.168.1.10';
    {
      MATCH: group='developer';
      VALIDATE: NOW;
      OR
      VALIDATE: NO;
    }
  }
  ```