System Security

ELV Core System

Bryan Shannon

Contents

Overview

The Early Learning Ventures (ELV) CORE system has been developed using the java based Spring MVC architecture. Spring MVC is perhaps the most prevalent architecture/framework in use today for Java based applications with 43% of developers using the Spring MVC framework. This document is intended to provide details regarding security as implemented within ELV CORE. Many of the security features implemented are made available by the Spring MVC framework.

The subjects covered within this document include:

- Secure Sockets Layer (SSL)
- User Roles
- User Accounts
- Expired Password Change
- User Authentication
- Session Timeout
- Password Reset
- Password Construction/Rules
- Application Security
- Database Security
- Database Backup
- Supported Browsers
- Multiple Logins
- Parent Security Codes

Note that the current version of the Spring MVC Framework is 5.3.6. The version of the Spring Framework ELV CORE is currently utilizing is 5.3.6 (as of June 2022). In December 2018 all ELV and child care provider users were required to update their passwords to allow ELV CORE to use the most current password encryption (BCrypt) while phasing out the old encryption standard SHA-256. The password changes were implemented as normal "Your password has expired…" application workflow, so the impact to users was minor. See the Expired Password Change section below for workflow details.

There are 5 web applications currently running on the Cordero hosted server:

- ELV CORE
- ELV Kiosk
- ELV Parent Portal
- ELV Mobile Rest
- ELV Provider Payments

All user passwords are encrypted before storage in the database using the encryption BCrypt.

Secure Sockets Layer (SSL)

It's critical that all data transmitted between your web browser and the ELV Core system server is encrypted prior to being transmitted over the internet. The SSL encryption is applied to every page in the ELV CORE system.

Users navigate to the ELV CORE application by entering the URL

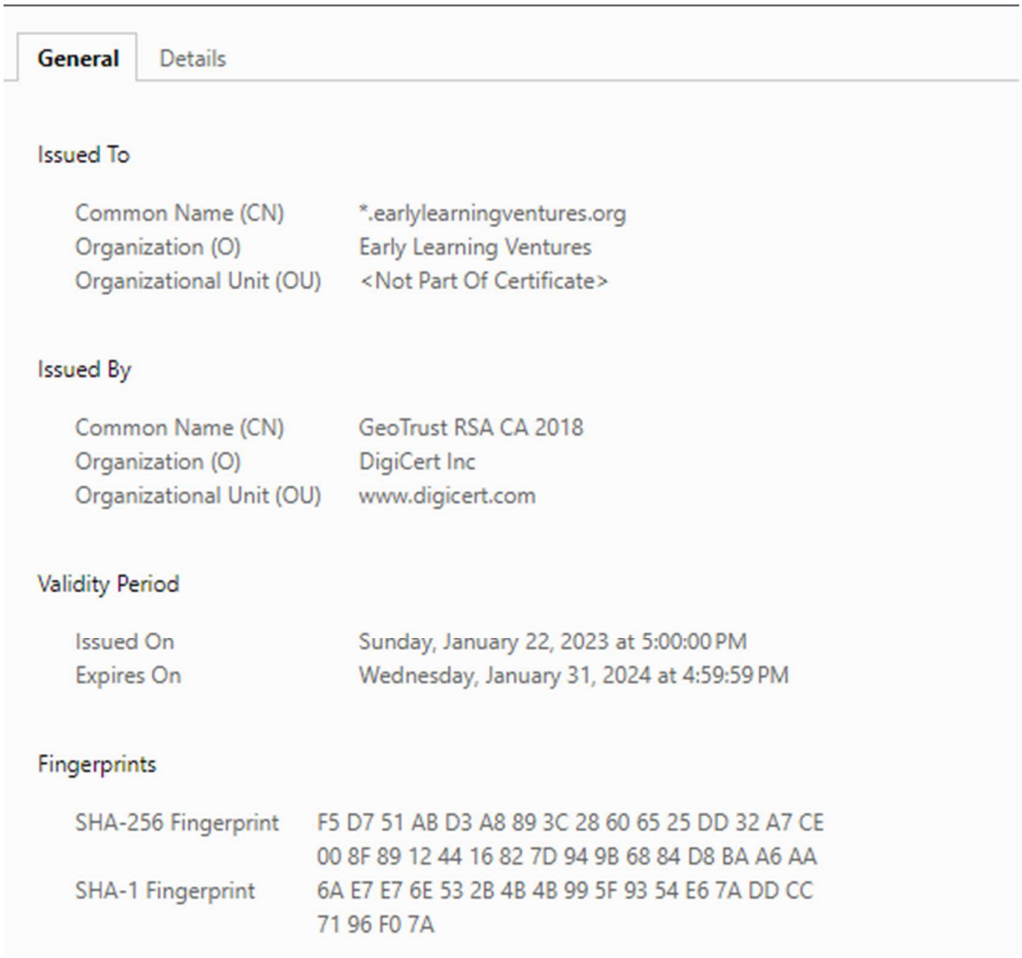
https://elv.earlylearningventures.org/elv/login

into the web browser address bar.

The first thing to notice about the above URL is the prefix of “https”. The protocol identifier **https** indicates that users will access the system using a protocol called Secure Sockets Layer (SSL). The SSL protocol forces the browser to establish an encrypted communication session with the server that is receiving the request (i.e., the server identified in the URL). This means that all data sent to and returned by the server from the user’s browser will be encrypted before being sent.

While there is some overhead associated with SSL sessions, SSL data encryption guarantees that data cannot be evaluated by others while the data is in transit over the internet. The SSL certificate used by the ELV CORE system is the most current 2048 bit encryption available to commercial web sites (algorithm is PKCS #1 SHA-256 With RSA Encryption) issued by the certificate authority GeoTrust.

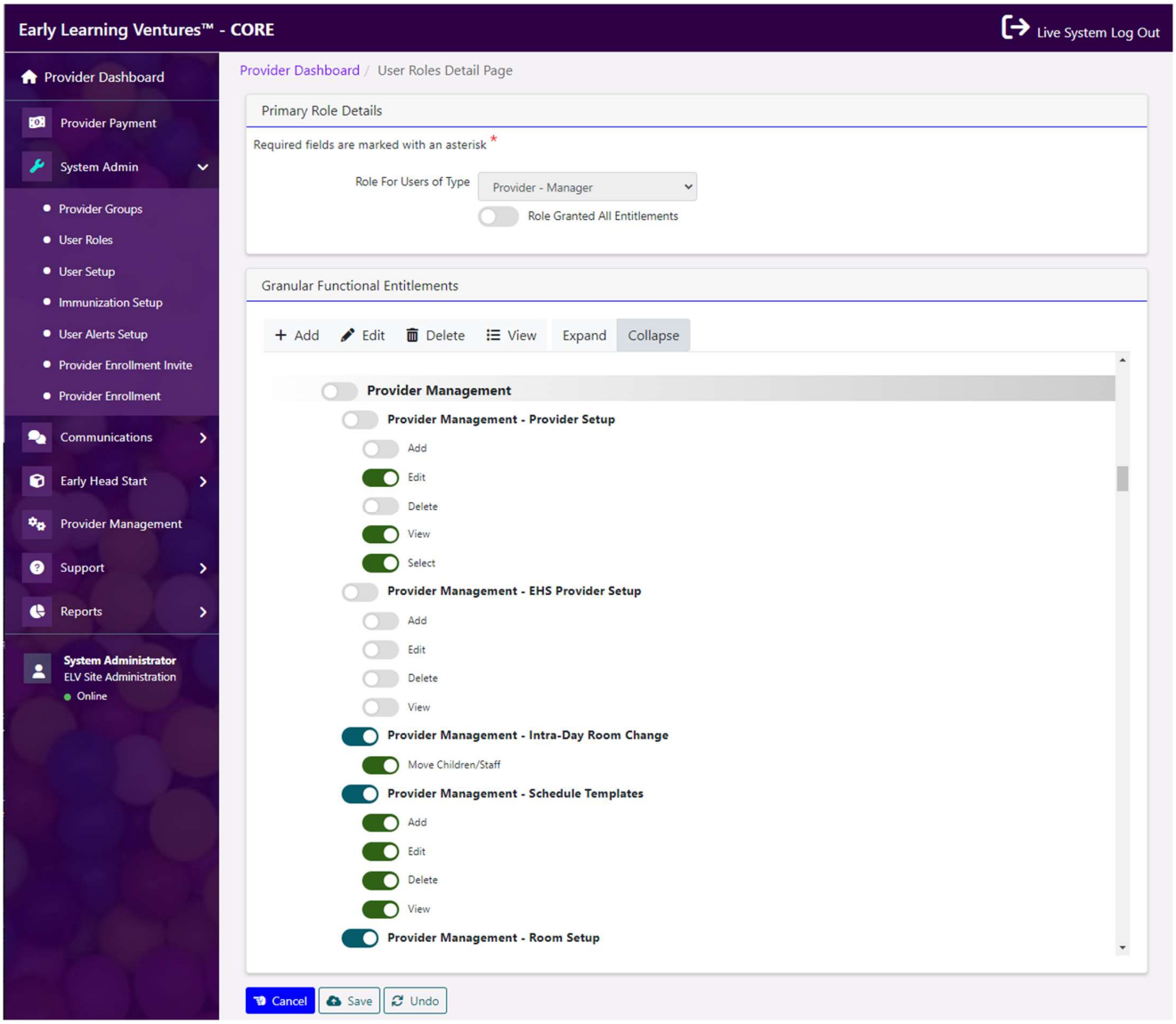
All current browsers will give you access to the details of the SSL server certificate that is in use.



User Roles

User Roles are foundational in terms of how functional access is granted to users of the ELV CORE System.

Within ELV CORE a series of “Roles” have been defined, and each user that is given access to ELV CORE is associated with a Role. The screen shot below is of the role “Provider – Manager” that is associated with all child care provider manager type users. The Role definition identifies what actions can be performed on every function within the system.



For child care provider users, the following Roles are currently defined:

- Provider – Manager
- Provider – Staff
- Provider – Teacher
- Provider – Teacher with Attendance
- Provider – Kiosk

User Accounts

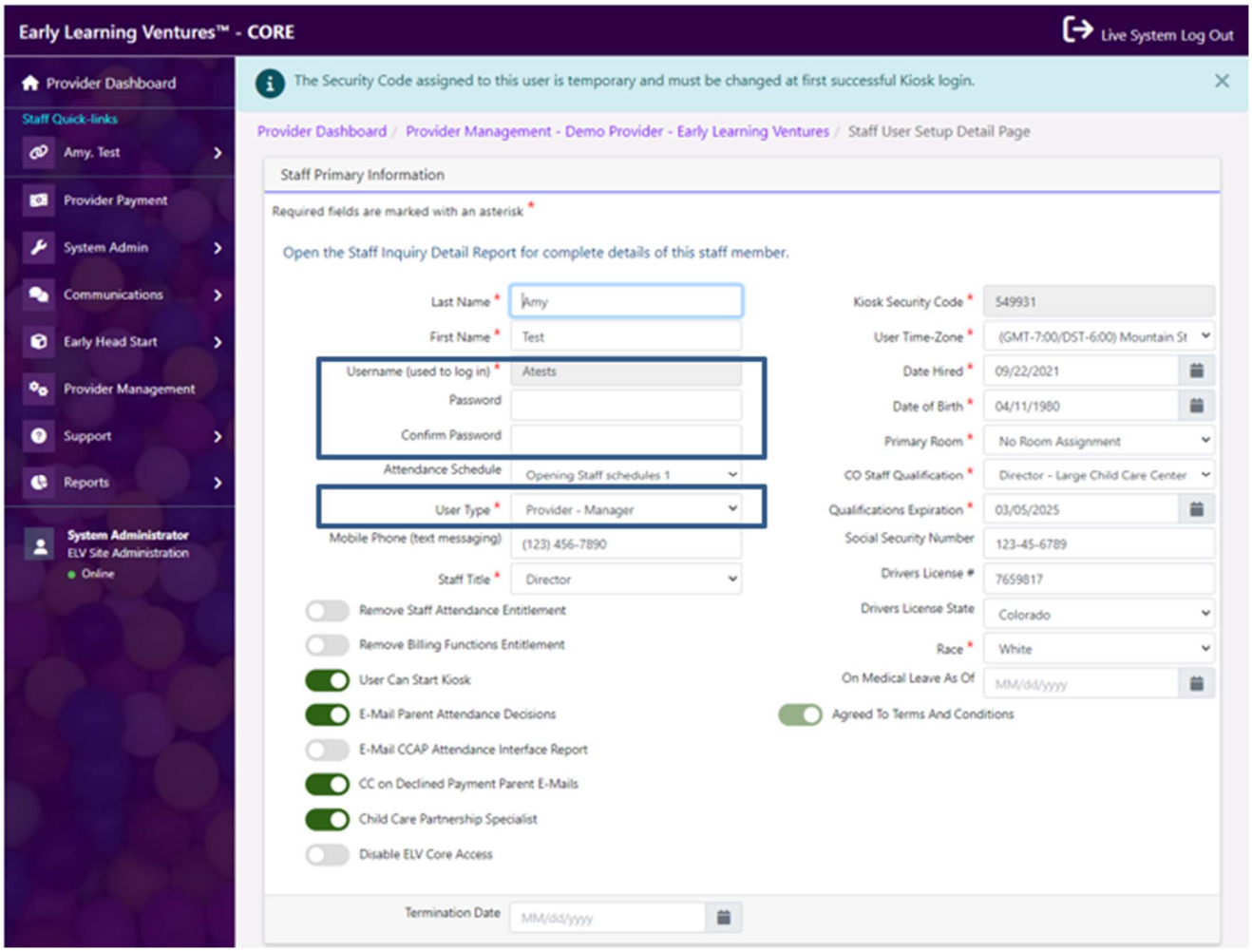
Child care provider user accounts are created using the Staff Setup function found under the Provider menu option.

The input fields highlighted on the screen shot below the user access into the system (username and password), and control the user's functional access within the system (User Type or Role).

When creating a new user, a unique username is required. The username must be unique across all child care providers using the ELV CORE system. The Password must be unique to this user only. The Confirm Password input field requires that the password be entered a second time so that the system can verify that the intended password was entered accurately.

When modifying a Staff User record the Username cannot be modified. The password however can be "Reset" by entering a new password for this user. The system will keep a history of passwords used by this user so that the new password cannot duplicate a password found in the password history. Currently only the three prior passwords are stored in the password history.

Note that when creating or resetting a user's password, the password will be automatically expired. When the user attempts to log into ELV CORE with the new/reset password the system will immediately force the user to change the password to a unique value that only the user knows. This prevents the user who created (or did a password reset) the Staff User account from knowing the user's password.



Also note Staff Users assigned to roles of "Provider – Staff" or "Provider – Teacher" will only have access to their own staff record. Only users of type "Provider – Manager" can create new users, or change the password for other users.

If the user is not a User Type of “Provider – Kiosk”, the “User Can Start Kiosk” must be checked to give the user authority to log into the Kiosk application (i.e., start up a kiosk).

Expired Password Change

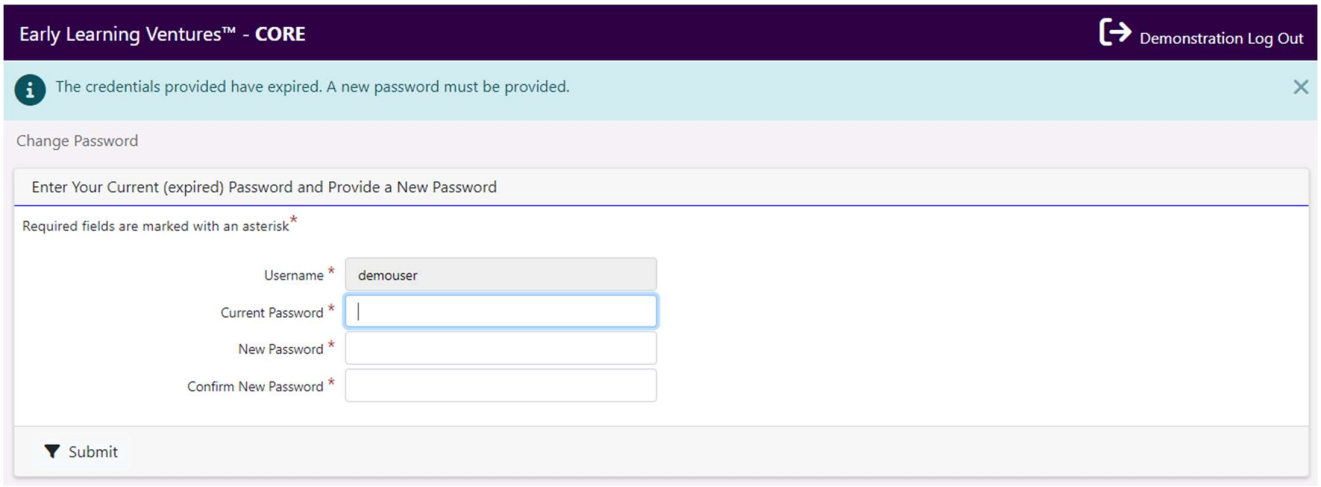
The Change Password page is presented based on three different conditions:

1. For new users accessing the system for the first time.
2. For users whose password has been reset.
3. For users whose password has expired.

When a new user logs into the system for the first time, the system will require that the user provide a new password that only they know.

When a user's password has been reset the system will require that the user provide a new password that only they know.

The system is configured with many password related parameters that allow ELV to configure the handling of passwords. One of those parameters has to do with the frequency at which passwords must be changed. Currently this parameter is set to 365 days, so users have one year to use a password before the system will require a new password.



The Change Password page requires that users enter the current password once, and then the new password twice (to verify input accuracy).

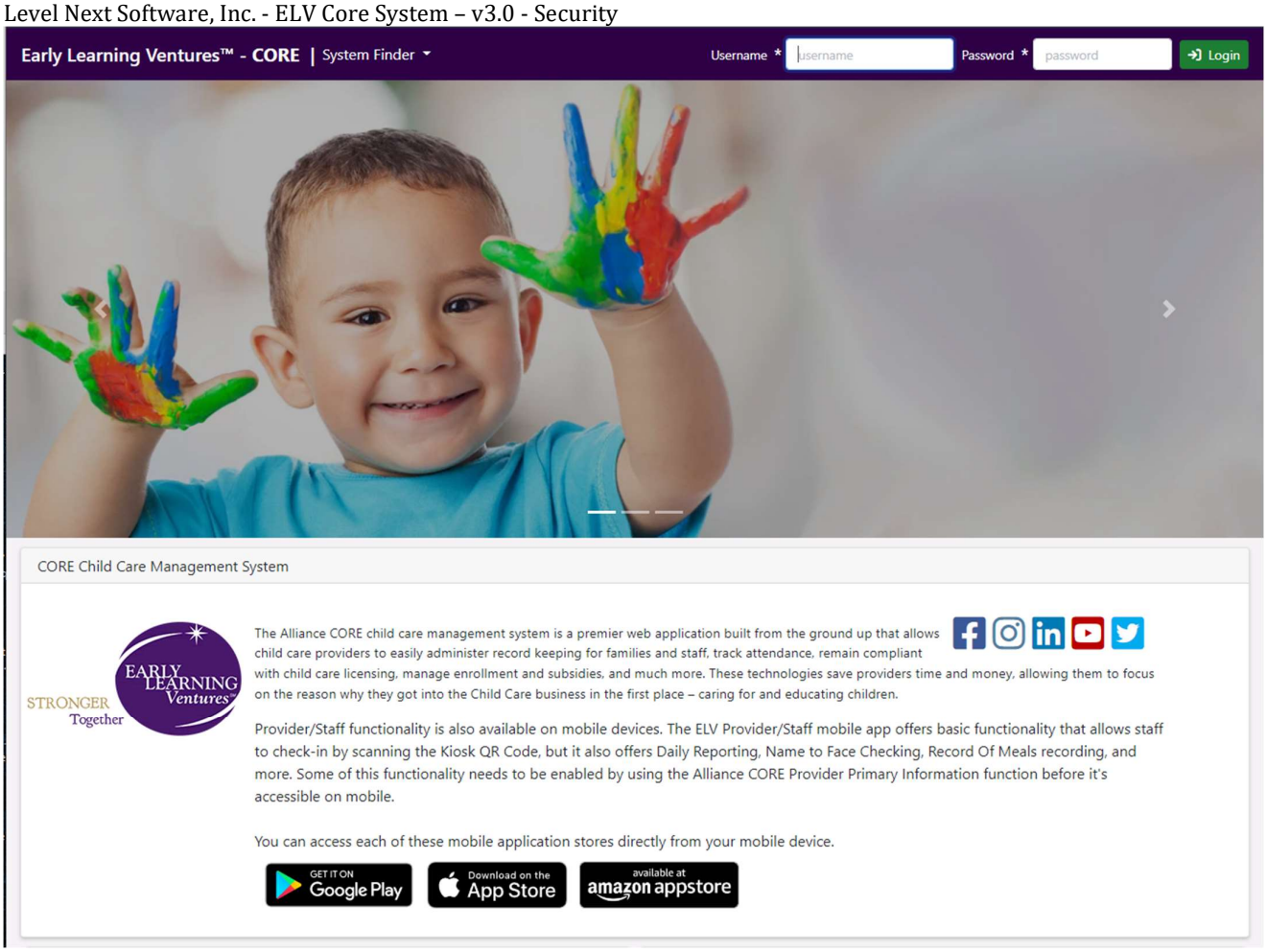
Note that for security minded users your password can be reset by you, at any time. See the section below labelled **Password Reset** for more information.

See the section Password Construction Rules for more detail.

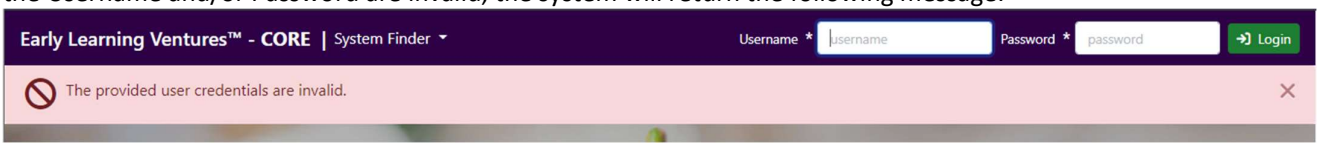
User Authentication

Now that Username and Passwords creation within the system has been covered, some discussion of how the system authenticates user credentials is in order. User Authentication is the process of collecting user credentials (username/password) and comparing those credentials against the known values stored within the ELV CORE system database.

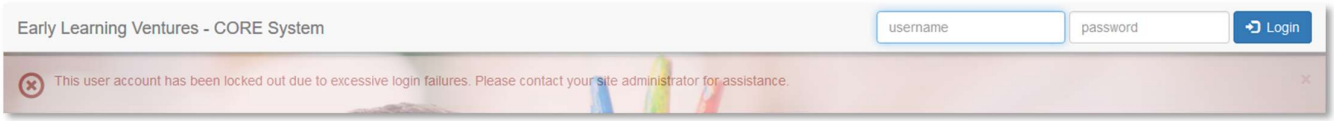
After the URL is entered into a browsers address bar, the ELV CORE system login page is displayed:



Login requires that users provide a valid Username and Password. The two input fields at the top of the above page are used to enter the Username first, followed by the Password. When the Login button is clicked, the credentials are passed from the client browser to the ELV CORE server over SSL, and the server will validate the credentials. If the Username and/or Password are invalid, the system will return the following message:



The system will not give hints regarding which credential is invalid. If a user enters the correct Username and an incorrect Password more than 5 times in a row, the user account will be locked and the password will need to be reset but a user with valid credentials.



The process of validating credentials involves taking the provided user credentials, encrypting the password (using the password itself and another private value), and then comparing the encrypted result with the encrypted password stored in the ELV CORE database.

The encryption process is one-way, meaning that the ELV system cannot decrypt a password that is stored within the ELV database. This means that the only way to gain access to the system using another user's username is to know their password, or to use a brute force attacks attempting a new password each time. However, the system locks the account after 5 failed attempts to prevent brute force attacks.

If the Username and Password are determined to be valid, and the account is not disabled (locked), the user is navigated to the ELV Home page. From this point forward the functionality available to the user within ELV CORE is governed by assigned User Role.

Session Timeout

When a user is successful logged into the ELV CORE system, the system assigns the user a session which contains information collected during login. The system also starts a count-down clock to monitor user inactivity. The count-down close starts with 35 minutes, and is reset to 35 minutes each time the user navigates between pages.

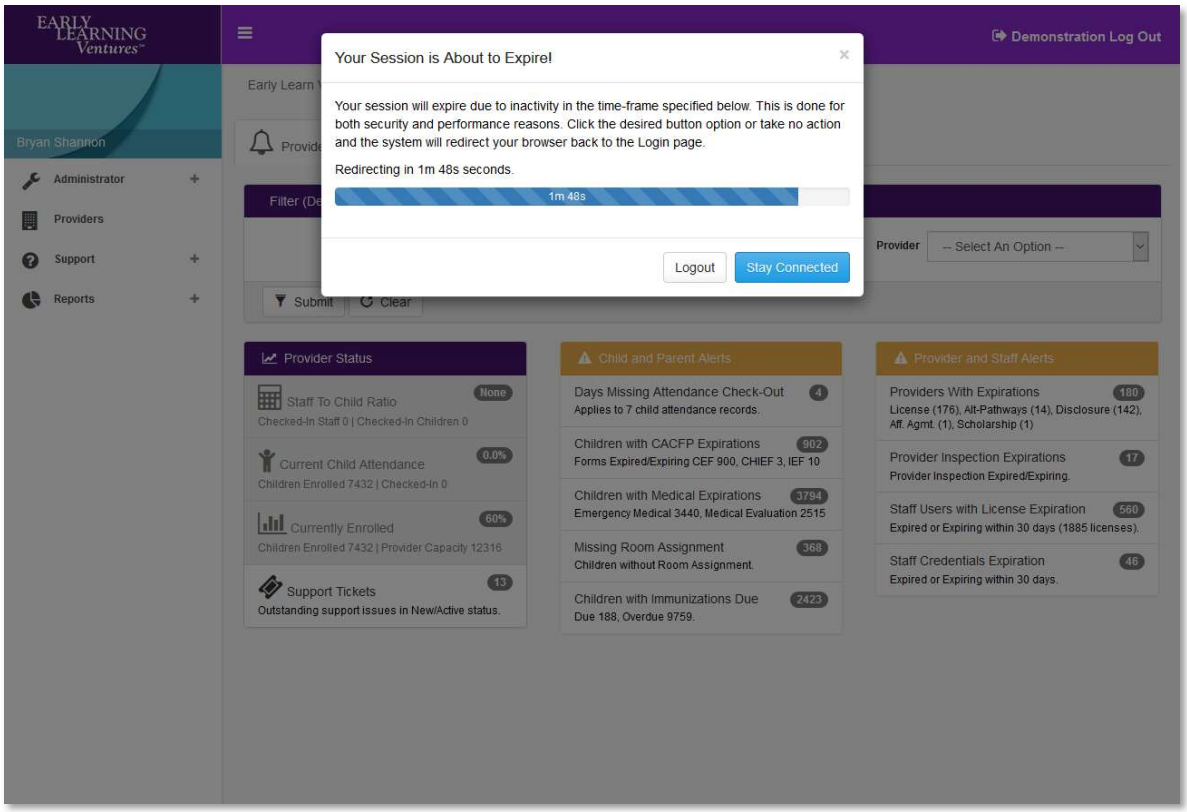
If the user's browser does not interact with the ELV CORE server for a period of 35 minutes, the ELV CORE system will destroy the user's session and redirect the user's browser to the login page.

This is done for two reasons:

- A browser that is logged into the ELV CORE system potentially left unattended for a prolonged period of time is considered to be insecure.
- If users are not actually using the system it is a good idea to free up the memory being utilized by the user's session, rather than leaving inactive (unused) sessions active.

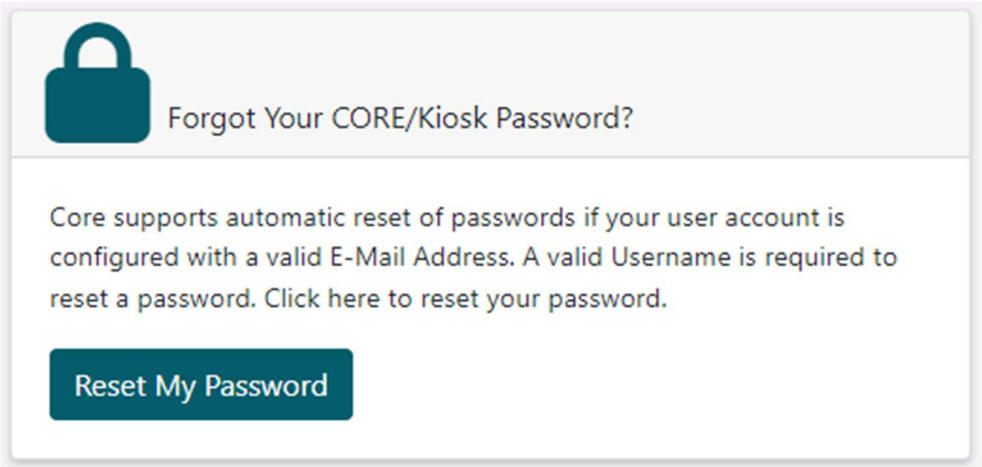
The ELV CORE system has a system timeout set for 35 minutes. At minute 30 a modal dialog box will be displayed within the user's browser window indicating that their session is about to expire. It will display a two-minute count-down timer, and will offer two button options to either Log Out, or to Stay Logged In.

If the user clicks the "Stay Logged In" button, their session will be refreshed and the 35-minute count-down clock will restart. If they click Log Out the system will immediately destroy the user's session and direct the user to the login page. If the user does nothing after two minutes the system will destroy the user's session, and redirect to the login page. This dialog would appear as follows:

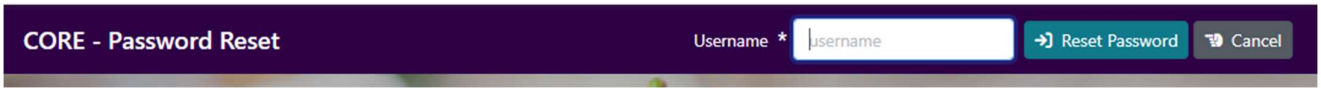


Password Reset

The ELV CORE login page has a section found in the body of the page that appears as follows:



To reset your password simply click on the "Forgot Your Password?" section. When you click on this section the login page will change at the top to display the CORE – Password Reset navigation bar:



Enter your user identifier and click the Reset Password button. If the Username entered is valid the system will:

- Reset the password associated with the user account
- Send an e-mail message to the e-mail address associated with the user account
- Re-display the standard login page

The e-mail message will contain a temporary password that must be changed the first time the user logs into the system.

Click the Password Reset navigation bars Cancel button if you do not wish to reset the password. The standard login navigation bar will be re-displayed.

Password Construction/Rules

The rules regarding the handling/structure of passwords is for the most part parameterized. These parameters exist in a system file called “early-learning.properties”. This property file contains many properties (key/value pairs), the following are the properties related to usernames and passwords.

```
# user identifier configuration (required for UserName/Password authentication)
username.require.alpha.uppercase=false
username.require.alpha.lowercase=false
username.require.numeric=false
username.special.characters.required.count=0
username.special.characters=!@#$%^&*()_+
username.minimum.length=6
username.allow.space=false
username.same.consecutive.character=true

failed.login.attempts.before.lockout=5
reenable.lockout.accounts.after.minutes=60

# password configuration (required for UserName/Password authentication)
password.require.alpha.uppercase=false
password.require.alpha.lowercase=false
password.require.numeric=false
password.special.characters.required.count=0
password.special.characters=!@#$%^&*()_+
password.minimum.length=8
password.allow.space=false
password.same.consecutive.character=true
password.can.equal.username=false
# Number of passwords to keep in PasswordHistory (0 to keep all)
password.history.to.keep.count=1
# Number of days that passwords will be valid before they must be changed (must be 1 or greater).
password.valid.for.days=365
password.valid.for.days.admin=365
password.minimum.length.admin=8
# Set to zero to disable auto-expiration of inactive accounts.
password.inactive.auto.expire.after.days=365
# How many days is a temporary password valid for?
password.temporary.password.valid.for.days=365
```

This document will not go into the details of each of these key/value pairs (which are documented elsewhere), however these parameters can be modified to control the structure of usernames and passwords, the number of days that a password is valid before the system requires it to be changed, etc. For example, if we wanted passwords to be a minimum length of 8, to be composed of: both upper and lower case letters, numeric values, and at least (n) special characters we could change the properties as follows:

```
password.require.alpha.uppercase=true
password.require.alpha.lowercase=true
password.require.numeric=true
password.special.characters.required.count=1
```

We could also indicate that the password expires after 30 days:

```
password.inactive.auto.expire.after.days=30
password.temporary.password.valid.for.days=30
```

As of this writing, passwords are valid (do not expire) for 365 days (a full year), must be at least 8 characters. Five invalid login attempts will lock user accounts. Locked accounts are unlocked after 60 minutes of being locked.

Note: These rules would not constitute strong passwords. There was concern at the time of conversion from the legacy CORE system that having more stringent rules, or forcing passwords to change at a greater frequency would be a problem for most child care providers.

Application Security

The ELV CORE application is secured by requiring an authenticated user session before requests can be made from the various application functions.

When a user is authenticated the user's session is established, and a portion of the session information contains the entitlements granted to the user. The entitlements are extracted based on the users Role (User Type), and are used to construct the user's menu within the ELV CORE user interface. Any functions/features to which the user is not entitled will not be accessible via the user interface.

Regardless of the functions listed within the menu, the system has to prevent users from making requests (typing a URL into the browser address bar) from those portions of the system to which the user is not entitled. To accomplish this the ELV CORE system uses a feature of the Spring MVC framework, the Controller method annotation "@Secured".

For instance, when a request is made to display the Child Setup list page, the ChildController java code that would be executed is the following:

```
private final String ENTITLEMENT_KEY = EntitlementConstants.PROVIDER_CHILD;
…
@Secured(ENTITLEMENT_KEY)
@RequestMapping(value = UriConstants.REQUESTMAPPING_SUFFIX_LISTPAGE, method = RequestMethod.GET)
public ModelAndView displayListPage(HttpSession session) {
        …
}
```

The @Secured annotation however will require that before the code can be executed that Spring must validate that the user is entitled to access this system feature. The @Secured annotation will call java code that compares the Provider Child entitlement against the users granted entitlements. If the user is not authorized for this feature Spring MVC will throw an error (Not Authorized).

The only portions of the system that are not annotated with @Secured are related to Login In and Log Out code that will need to execute against unauthenticated users.

Security HTTP Response Headers

Web Configuration

All web based Alliance CORE applications are configured with fundamental protections offered by the Spring framework:

```
<!-- Enable Cross Site Request Forgery Protection -->
<security:csrf disabled="false"/>
<!-- This controls the HTTP headers that are returned in the response to the browser -->
<security:headers>
      <security:cache-control />
      <security:content-type-options />
      <security:hsts />
      <security:frame-options policy="SAMEORIGIN"/>
      <security:xss-protection />
</security:headers>
```

Cross-Site Request Forgery (CSRF)

Before we discuss how Spring Security can protect applications from CSRF attacks, we will explain what a CSRF attack is. Let's take a look at a concrete example to get a better understanding.

Assume that your bank's website provides a form that allows transferring money from the currently logged in user to another bank account. For example, the HTTP request might look like:

```
POST /transfer HTTP/1.1
Host: bank.example.com
Cookie: JSESSIONID=randomid; Domain=bank.example.com; Secure; HttpOnly
Content-Type: application/x-www-form-urlencoded

amount=100.00&routingNumber=1234&account=9876
```

Now pretend you authenticate to your bank's website and then, without logging out, visit an evil website. The evil website contains an HTML page with the following form:

```
<form action="https://bank.example.com/transfer" method="post">
<input type="hidden"
       name="amount"
       value="100.00"/>
<input type="hidden"
       name="routingNumber"
       value="evilsRoutingNumber"/>
<input type="hidden"
       name="account"
       value="evilsAccountNumber"/>
<input type="submit"
       value="Win Money!"/>
</form>
```

You like to win money, so you click on the submit button. In the process, you have unintentionally transferred $100 to a malicious user. This happens because, while the evil website cannot see your cookies, the cookies associated with your bank are still sent along with the request.

Worst yet, this whole process could have been automated using JavaScript. This means you didn't even need to click on the button. So how do we protect ourselves from such attacks?

Synchronizer Token Pattern

The issue is that the HTTP request from the bank's website and the request from the evil website are exactly the same. This means there is no way to reject requests coming from the evil website and allow requests coming from the bank's website. To protect against CSRF attacks we need to ensure there is something in the request that the evil site is unable to provide.

One solution is to use the Synchronizer Token Pattern. This solution is to ensure that each request requires, in addition to our session cookie, a randomly generated token as an HTTP parameter. When a request is submitted, the server must look up the expected value for the parameter and compare it against the actual value in the request. If the values do not match, the request should fail.

We can relax the expectations to only require the token for each HTTP request that updates state. This can be safely done since the same origin policy ensures the evil site cannot read the response. Additionally, we do not want to include the random token in HTTP GET as this can cause the tokens to be leaked.

Let's take a look at how our example would change. Assume the randomly generated token is present in an HTTP parameter named _csrf. For example, the request to transfer money would look like this:

```
POST /transfer HTTP/1.1
Host: bank.example.com
Cookie: JSESSIONID=randomid; Domain=bank.example.com; Secure; HttpOnly
Content-Type: application/x-www-form-urlencoded

amount=100.00&routingNumber=1234&account=9876&_csrf=<secure-random>
```

You will notice that we added the _csrf parameter with a random value. Now the evil website will not be able to guess the correct value for the _csrf parameter (which must be explicitly provided on the evil website) and the transfer will fail when the server compares the actual token to the expected token.

Cache Control

In the past Spring Security required you to provide your own cache control for your web application. This seemed reasonable at the time, but browser caches have evolved to include caches for secure connections as well. This means that a user may view an authenticated page, log out, and then a malicious user can use the browser history to view the cached page. To help mitigate this Spring Security has added cache control support which will insert the following headers into your response.

```
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
```

Content Type Options

Historically browsers, including Internet Explorer, would try to guess the content type of a request using content sniffing. This allowed browsers to improve the user experience by guessing the content type on resources that had not specified the content type. For example, if a browser encountered a JavaScript file that did not have the content type specified, it would be able to guess the content type and then execute it.

> There are many additional things one should do (i.e. only display the document in a distinct domain, ensure Content-Type header is set, sanitize the document, etc) when allowing content to be uploaded. However, these measures are out of the scope of what Spring Security provides. It is also important to point out when disabling content sniffing, you must specify the content type in order for things to work properly.

The problem with content sniffing is that this allowed malicious users to use polyglots (i.e. a file that is valid as multiple content types) to execute XSS attacks. For example, some sites may allow users to submit a valid postscript document to a website and view it. A malicious user might create a postscript document that is also a valid JavaScript file and execute a XSS attack with it.

HTTP Strict Transport Security (HSTS)

When you type in your bank's website, do you enter mybank.example.com or do you enter https://mybank.example.com? If you omit the https protocol, you are potentially vulnerable to Man in the Middle attacks. Even if the website performs a redirect to https://mybank.example.com a malicious user could intercept the initial HTTP request and manipulate the response (i.e. redirect to https://mibank.example.com and steal their credentials).

Many users omit the https protocol and this is why HTTP Strict Transport Security (HSTS) was created. Once mybank.example.com is added as a HSTS host, a browser can know ahead of time that any request to mybank.example.com should be interpreted as https://mybank.example.com. This greatly reduces the possibility of a Man in the Middle attack occurring.

> In accordance with RFC6797, the HSTS header is only injected into HTTPS responses. In order for the browser to acknowledge the header, the browser must first trust the CA that signed the SSL certificate used to make the connection (not just the SSL certificate).

X-Frame-Options

Allowing your website to be added to a frame can be a security issue. For example, using clever CSS styling users could be tricked into clicking on something that they were not intending (video demo). For example, a user that is logged into their bank might click a button that grants access to other users. This sort of attack is known as Clickjacking.

> Another modern approach to dealing with clickjacking is to use Section 21.1.7, "Content Security Policy (CSP)".

There are a number ways to mitigate clickjacking attacks. For example, to protect legacy browsers from clickjacking attacks you can use frame breaking code. While not perfect, the frame breaking code is the best you can do for the legacy browsers.

A more modern approach to address clickjacking is to use X-Frame-Options header:

```
X-Frame-Options: DENY
```

The X-Frame-Options response header instructs the browser to prevent any site with this header in the response from being rendered within a frame. By default, Spring Security disables rendering within an iframe.

You can customize X-Frame-Options with the frame-options element. For example, the following will instruct Spring Security to use "X-Frame-Options: SAMEORIGIN" which allows iframes within the same domain:

```
<http>
```

```
        <!-- ... -->

        <headers>
                <frame-options
                policy="SAMEORIGIN" />
        </headers>
</http>
```

X-XSS-Protection

Some browsers have built in support for filtering out reflected XSS attacks. This is by no means foolproof, but does assist in XSS protection.

The filtering is typically enabled by default, so adding the header typically just ensures it is enabled and instructs the browser what to do when a XSS attack is detected. For example, the filter might try to change the content in the least invasive way to still render everything. At times, this type of replacement can become a XSS vulnerability in itself. Instead, it is best to block the content rather than attempt to fix it. To do this we can add the following header:

```
X-XSS-Protection: 1; mode=block
```

This header is included by default. However, we can customize it if we wanted. For example:

```
<http>
        <!-- ... -->

        <headers>
                <xss-protection block="false"/>
        </headers>
</http>
```

Java Persistence Architecture (JPA)

Java Persistence API is a source to store business entities as relational entities. It shows how to define a PLAIN OLD JAVA OBJECT (POJO) as an entity and how to manage entities with relations.

How Applications Become Vulnerable to SQL Injection?

Injection attacks work because, for many applications, the only way to execute a given computation is to dynamically generate code that is in turn run by another system or component. If in the process of generating this code we use untrusted data without proper sanitization, we leave an open door for hackers to exploit.

This statement may sound a bit abstract, so let's take look at how this happens in practice with a textbook example:

```
public List<AccountDTO>
  unsafeFindAccountsByCustomerId(String customerId)
  throws SQLException {
    // UNSAFE !!! DON'T DO THIS !!!
    String sql = "select "
      + "customer_id,acc_number,branch_id,balance "
      + "from Accounts where customer_id = '"
      + customerId
      + "'";
    Connection c = dataSource.getConnection();
    ResultSet rs = c.createStatement().executeQuery(sql);
    // ...
}
```

The problem with this code is obvious: **we've put the *customerId*'s value into the query with no validation at all**. Nothing bad will happen if we're sure that this value will only come from trusted sources, but can we?

Let's imagine that this function is used in a REST API implementation for an *account* resource. Exploiting this code is trivial: all we have to do is to send a value that, when concatenated with the fixed part of the query, change its intended behavior:

```
curl -X GET \
  'http://localhost:8080/accounts?customerId=abc%27%20or%20%271%27=%271' \
```

Assuming the *customerId* parameter value goes unchecked until it reaches our function, here's what we'd receive:

```
abc' or '1' = '1
```

When we join this value with the fixed part, we get the final SQL statement that will be executed:

```
select customer_id, acc_number,branch_id, balance
  from Accounts where customerId = 'abc' or '1' = '1'
```

Probably not what we've wanted…

JPA Criteria API

Since explicit JQL query building is the main source of SQL Injections, we favor the use of the JPA's Query API, when possible.

Let's rewrite our JPA query method to use the Criteria API:

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Account> cq = cb.createQuery(Account.class);
Root<Account> root = cq.from(Account.class);
cq.select(root).where(cb.equal(root.get(Account_.customerId), customerId));

TypedQuery<Account> q = em.createQuery(cq);
// Execute query and return mapped results (omitted)
```

Here, we've used more code lines to get the same result, but the upside is that now **we don't have to worry about JQL syntax**.

Another important point: despite its verbosity, **the Criteria API makes creating complex query services more straightforward and safer.**

Java Persistence Query language (JPQL)

If we are not using the JPA Criteria API, we are using JPQL.

```
@Modifying
@Query("Update ChildAttendanceModel set childDisplayName = :childDisplayName where childKy =
:childKy and childDisplayName <> :childDisplayName")
void updateChildDisplayName(

      @Param(ColumnConstants.CHILD_DISPLAY_NAME) String childDisplayName,
      @Param(ColumnConstants.CHILD_KY) Integer childKy);
```

Because the query utilizes placeholders for the query values, JPQ can interrogate each parameter and assure that the resulting query is constructed to defend against any injection attempt.

One Exception

The only place in the Alliance CORE system were JPA is not being utilized, is in Dynamic Reporting. This needs to be evaluated to determine if 1) it can be hardened to ensure that injection is not possible, or 2) if JPA can be used in place of query building.

Database Security

To access the ELV CORE database the system needs to have access to a database specific user identifier and password (this is not referring to an ELV user's username/password, this is unique to the database).

Only the ELV CORE system knows the credentials required to log into the ELV Core system database.

Within the ELV database, all database objects are owned by the schema "dbo", and all database objects (tables, views, stored procedures, functions) grant required access to the database login used by the ELV system.

There are actually two different database logins used by the ELV CORE system, one used by the system excluding reporting (for read/write access), and another used for reporting only with read-only access.

Database Backup

Two maintenance plans are defined on the database server that handles

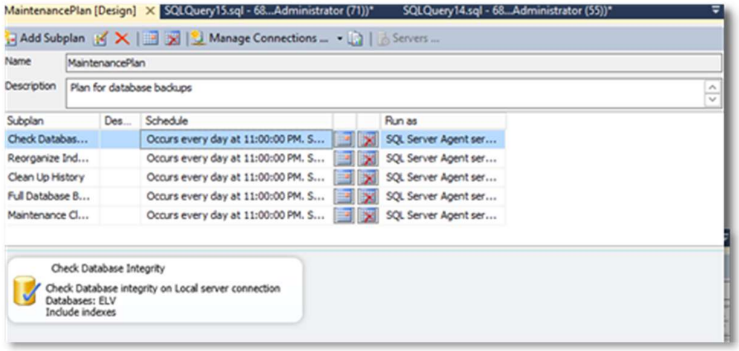
1) Full database backup, and
2) Transaction logging (incremental database backups).

Full database backup is done nightly at 11:00PM MST. Transaction logging (or incremental backups) are done every 30 minutes throughout the day. A Transaction Log backup contains a log of the transactions that have occurred within the last 30 minute interval. To restore a database you must restore the last full database backup, followed by all transaction log backups that occurred after the last full database backup.

Full Database Backup Plan

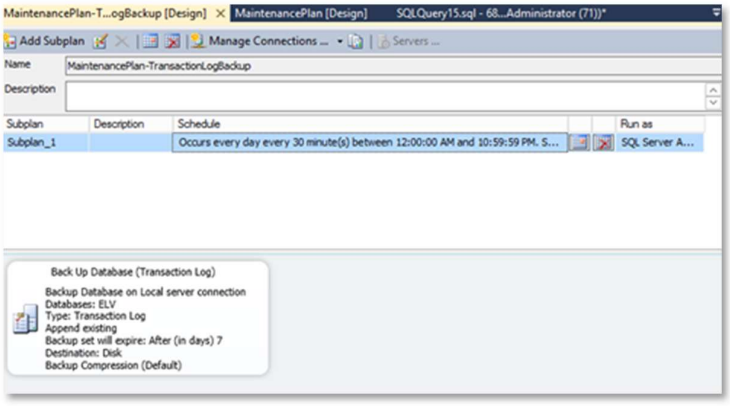
The full database backup plan is composed of five steps:

1) Check Database Integrity
2) Reorganize Indexes (reorganize/optimize table indexes)
3) Clean Up History (remove job history older than 1 week old)
4) Full Database Backup
5) Cleanup Database Backups (remove backups older than 1 week old)



Transaction Log Backup Plan

The only task in the Transaction Log backup plan is to create a backup of the transaction log (those transactions that have occurred in the last 30 minutes).

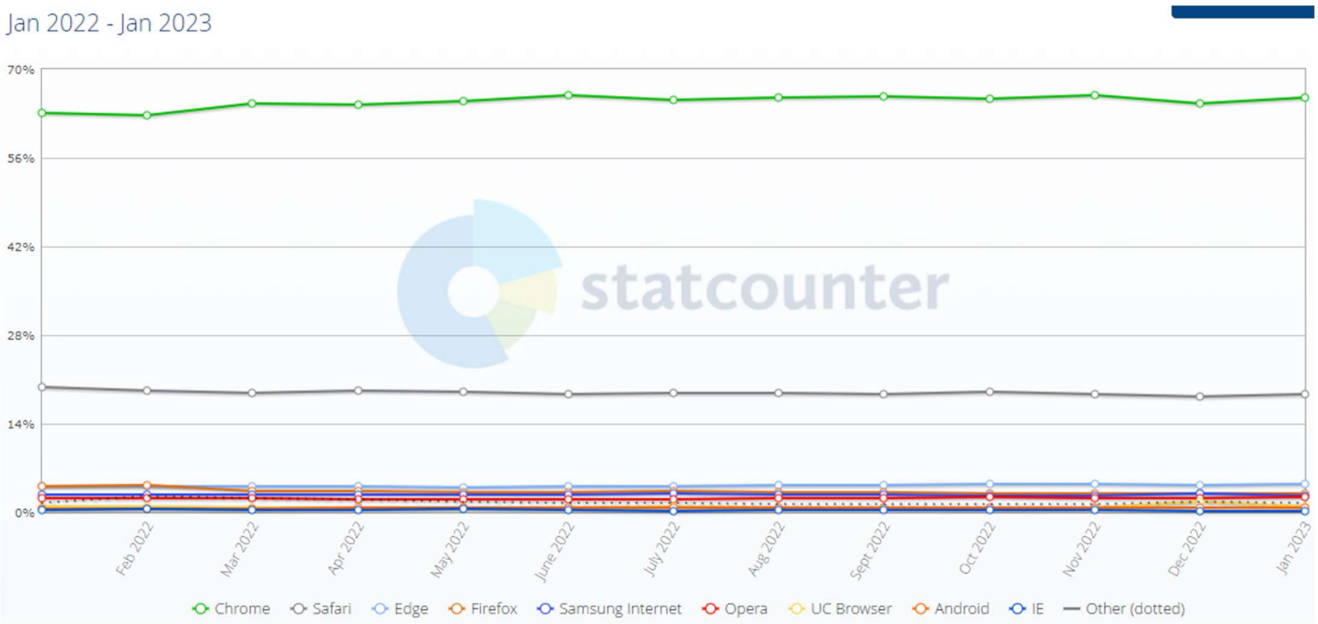


Because transaction logging occurs every 30 minutes, this is the maximum window of data loss that child care providers could experience due to a database loss (server crash).

Supported Browsers

The ELV CORE system has been tested with IE11, Microsoft Edge, and most current versions of Chrome and Firefox. Safari was not tested with but is used regularly by child care providers that access ELV CORE using iPad tablets.

Below are the browser statistics for using in Jan 2023 within the United States.



We recommend using Chrome, Edge or FireFox for best performance. No version of Internet Explorer prior to version 11 is supported by Microsoft.

It's important that you are setup to accept automatic updates on your browser for Chrome, Firefox or Edge so the most recent security features and functionality are available to you.

Also, we should discourage child care providers from using browser password managers.

Multiple Logins

The ELV system is configured to allow the same user to log into the system multiple times. The most secure user session strategy allows only 1 user session per user, and when log in occurs the system would log out any existing user session for that user.

```
<!-- Limit the number of sessions a user can have to 15 -->
<security:session-management invalid-session-url="/concurrency"
        session-fixation-protection="migrateSession">
        <security:concurrency-control  max-sessions="15" error-if-maximum-exceeded="false"
                expired-url="/concurrency"  session-registry-alias="sessionRegistry" />
</security:session-management>
```

However the ELV Core system allows multiple sessions per user, because child care provider managers (on occasion) require this type of access, but primarily because a child care provider can have multiple Kiosk's running at one time.

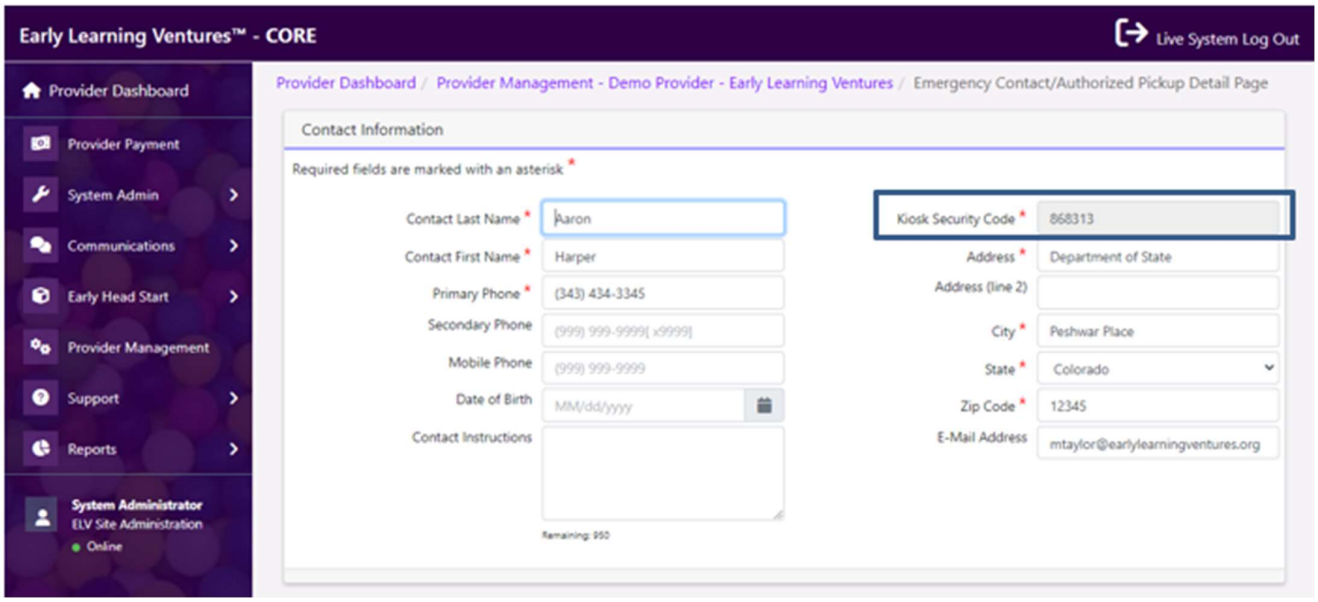
From a single computer, if a user wants to log into ELV multiple times they must use multiple browsers (ex: Chrome and IE11, FireFox and Chrome, etc). A browser holds a cooking called a JSessionID that is passed to the server with each request. If multiple browser tabs (in the same browser) are logged into the ELV system then the system will not functional correctly.

If however users access the ELV system multiple times from the same computer using two different browser types, the system will function normally because each browser will have a unique JSessionID value.

Parent Security Codes

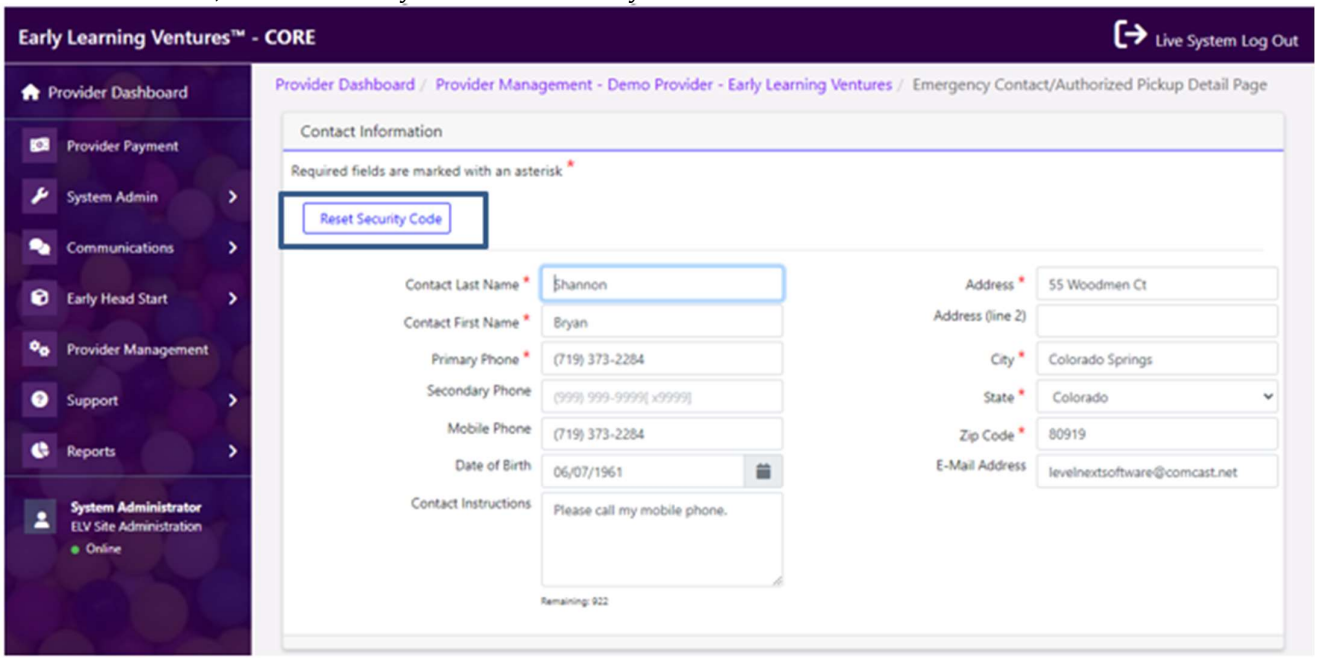
Within ELV CORE a function called Emergency Contact/Authorized Pickup is used by child care providers to define parents, guardians, and others that will drop-off/pick-up children at day care, act as emergency contacts, or both. Any person that is authorized to drop-off/pick-up children is issued a security code within this function.

When adding a new authorized pickup person, the system generates a temporary "Kiosk Security Code" (highlighted in red). This temporary code is then given to the parent/guardian or other authorized individual.



The authorized pickup person then uses the Security Code to log into the child care provider Kiosk, where they can check children in or out. However, because the security code is temporary, the Kiosk will force the authorized pickup person to change the security code to a value only they know before they can check children in for the first time.

Once the security code is changed by the authorized pickup person, the child care provider no longer knows the code. The Emergency Contact/Authorized Pickup function now shows a button at the top of the page which the child care provider can click if for any reason a new temporary security code needs to be assigned to the authorized pickup person.



The Kiosk will then force the authorized pickup person to change the security code to a value that only they know.

Authorized Pickup persons should be encouraged to:

1) Not use commonly known values like phone number.
2) Use a mix of alphabetic, numeric and special characters in making up a security code.
3) To keep their security code to themselves.

Up to 10 characters can be used when creating a security code, and a minimum of 6 are required.