Code Audit Banking Example

Submitted to:

Submitted by:

**Technical/
Business POC:**

Erik Santana

Date: June 14, 2022

Executive Summary

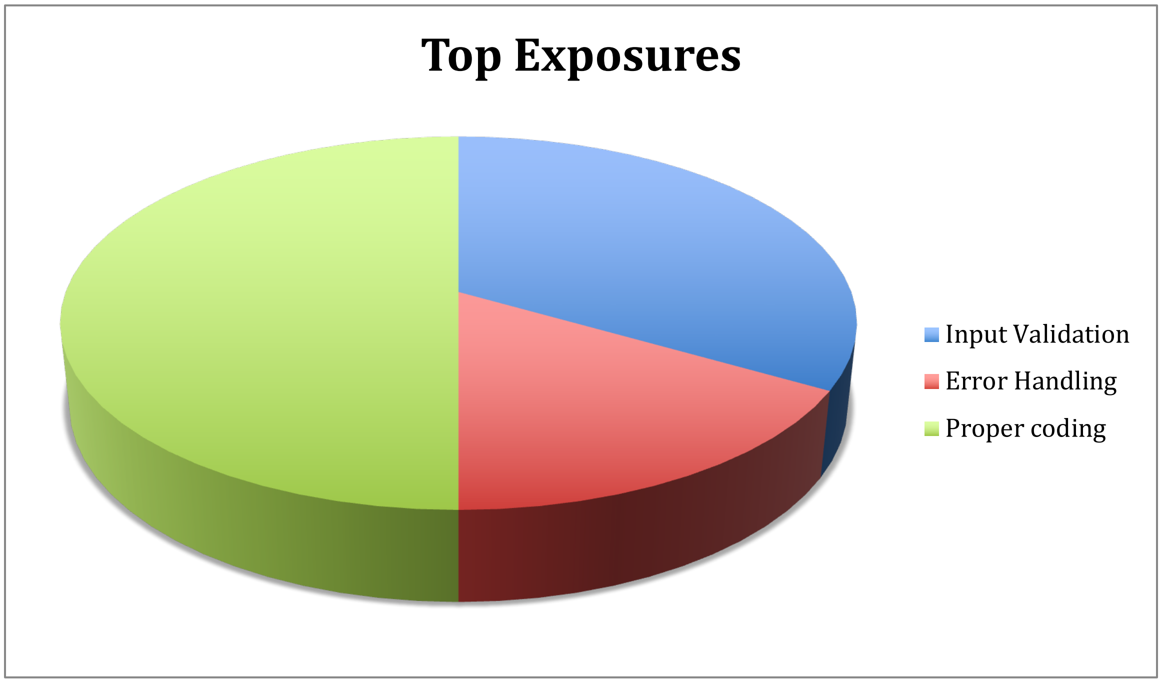
Within this report are the results of the code audit for banking site: The purpose of this report is to identify any security issues within the engagement scope of the analysis.

Engagement scope:

----- was tasked to analyze the recent code validations and compare the codes changes from the file named “pcreatrf.java” generated on and the latest file of the same name generated on What follows is a summary of the encountered vulnerabilities/exposures during the code audit:

Of particular attention, the ---- development team should handle the following issues promptly:

- Provide input validation of account funding, as this could prevent further banking account compromise.
- Ensure the proper code is in place for the recently implemented validations



Please be aware these issues are presented before you because they are the most outstanding in terms of your company’s security.

Important:

The supplied code is auto-generated with the program known as ----- is a cross-platform for developing web and mobile applications. A developer in describes an application in a high-level, mostly declarative language, from which native code is generated. It is important to note that can provide suggestions regarding the native Java code but not the code, as this custom code and not a standard.

Analysis

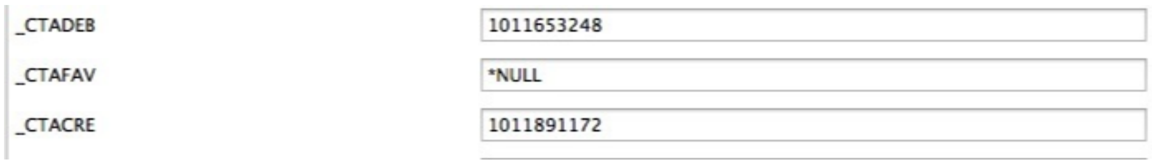
When developing applications, the most effective way of ensuring proper secure code is by using the "Defensive Programming" guidelines. This is the criteria used by during the code auditing:

- Making sure software behaves in a predictable manner despite unexpected inputs or user actions.
- Assuring that the source code comprehensible - the source code should be readable and understandable so it is approved in a code audit.
- Ensuring general quality - Reducing the number of software bugs and problems.

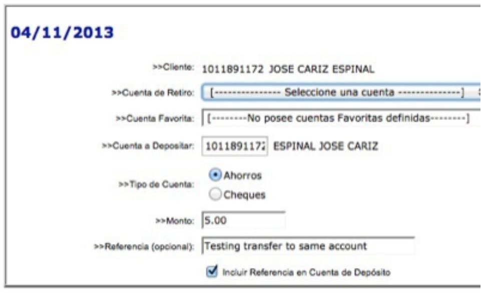
It is always recommended to prevent attacks as early as possible in the processing of the user's (attacker's) request. Input validation can be used to detect unauthorized input before it is processed by the application. conducted several input validation tests during the code auditing process.

Input validation tests

The following parameters were modified while attempting a transaction between a fake account number (1011653248) and a valid deposit account (1011891172):



_CTADEB	1011653248
_CTAFAV	*NULL
_CTACRE	1011891172



04/11/2013

>>Cliente:	1011891172 JOSE CARIZ ESPINAL
>>Cuenta de Retiro:	[--------------- Seleccione una cuenta ---------------]
>>Cuenta Favorita:	[--------No posee cuentas Favoritas definidas--------]
>>Cuenta a Depositar:	101189117 ESPINAL JOSE CARIZ
>>Tipo de Cuenta:	[✓ Ahorros] Cheques
>>Monto:	5.00
>>Referencia (opcional):	Testing transfer to same account
	✓ Incluir Referencia en Cuenta de Depósito

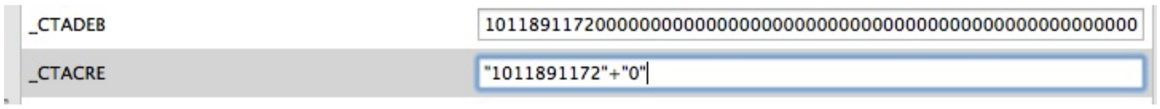
An insufficient funds alert is observed. This could indicate that the fake account was processed to the point of validating funds in it. This would have meant a bypass of the newly implemented account validators. **The expected response would have been a type 2 error: “Incorrect debit account number”.**

Recommendation

Account input validators need to be in place to determine whether the debit account is assigned to the actual user, before the verification actual funds in the account. A revision of the second account validator is recommended.

Error handling tests

The following parameters were modified while attempting a transaction between the same bank accounts. Zero padding was added to the debit account:



_CTADEB	101189117200000000000000000000000000000000000000000000000000
_CTACRE	"1011891172"+"0"

A Java exception error “Data Truncation” is observed. This type of exception appears when the MaxFieldSize is exceeded.

Recommendation

The Java interpreter handled the input correctly as it exceeded its maximum field size, but the exception stack trace is printed to the screen. This is a problem as it gives attackers a lot of information about the system and can lead to further attacks. The recommended action is to use error handlers for Java exceptions. For example, creating an xml page that handles specific Java errors with a generic message page error.jsp:

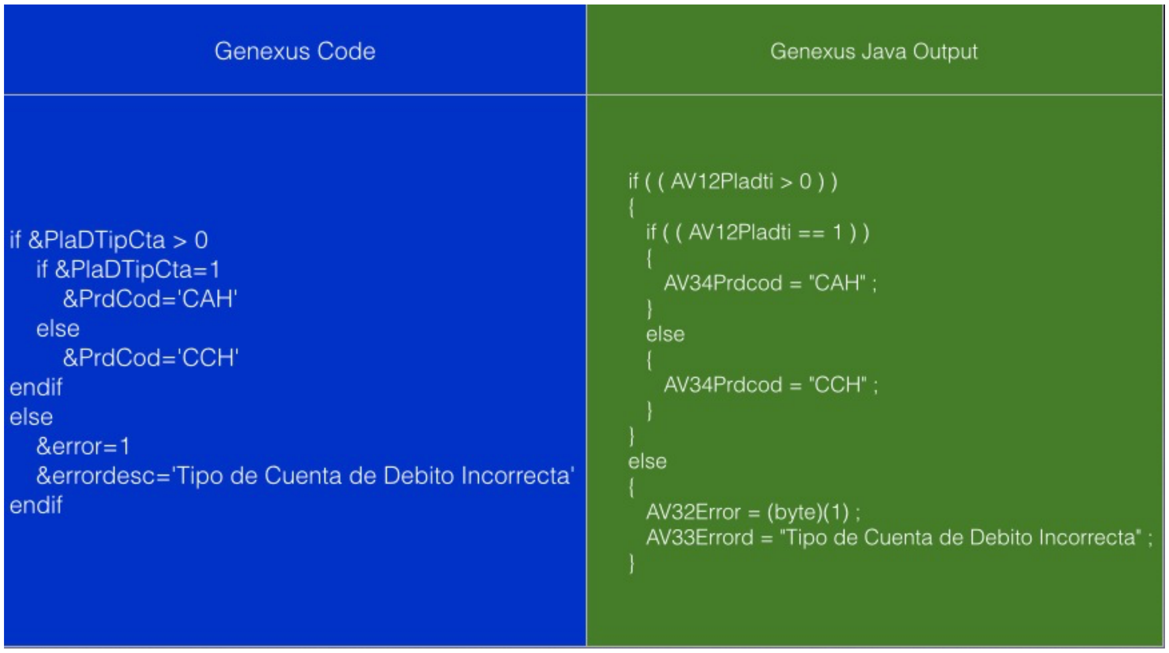
```
<error-page>
<error-code>404</error-code>
<location>/error.jsp</location>
</error-page>
<error-page>
<error-code>500</error-code>
<location>/error.jsp</location>
```

</error-page>
<error-page>
<exception-type>java.lang.Throwable</exception-type>
<location>/error.jsp</location>
</error-page>

Proper code

Three new data validations were found in the new code:

First Validation - Error Type = 1 "Incorrect Account Type"



Genexus Code

```
if &PlaDTipCta > 0
   if &PlaDTipCta=1
      &PrdCod='CAH'
   else
      &PrdCod='CCH'
endif
else
   &error=1
   &errordesc='Tipo de Cuenta de Debito Incorrecta'
endif
```

Genexus Java Output

```
if ( ( AV12Pladti > 0 ) )
{
   if ( ( AV12Pladti == 1 ) )
   {
      AV34Prdcod = "CAH" ;
   }
   else
   {
      AV34Prdcod = "CCH" ;
   }
}
else
{
   AV32Error = (byte)(1) ;
   AV33Errord = "Tipo de Cuenta de Debito Incorrecta" ;
}
```

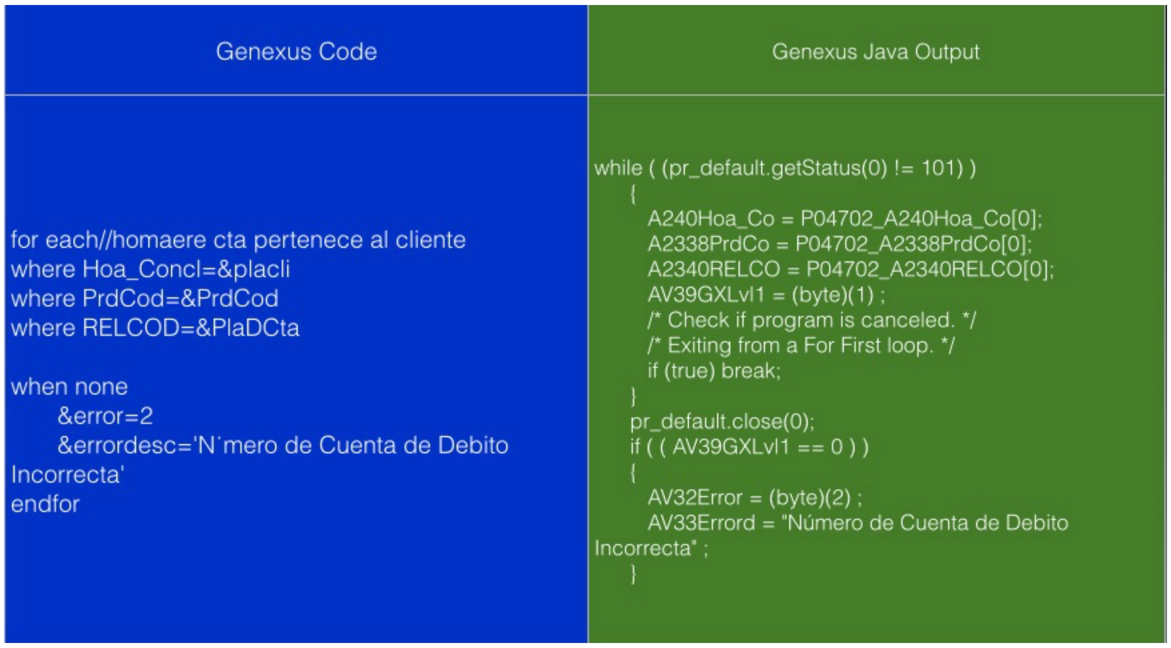
Recommendations:

Current code checks for an account type number greater than "0" and whether the account type is equal to "1" to assign it type "CAH", else it will be assigned type "CCH", no matter how large the positive number. A limit check should be in place to not accept account type number bigger than "X". For example

if ((AV12Pladti > 0 && AV12Pladti < 2))

This can help prevent buffer overflow attacks.

Second Validation - Error Type = 2 “Incorrect debit account number”



Genexus Code

```
for each//homaere cta pertenece al cliente
where Hoa_Concl=&placli
where PrdCod=&PrdCod
where RELCOD=&PlaDCta

when none
    &error=2
    &errordesc='N´mero de Cuenta de Debito
Incorrecta'
endfor
```

Genexus Java Output

```
while ( (pr_default.getStatus(0) != 101) )
   {
      A240Hoa_Co = P04702_A240Hoa_Co[0];
      A2338PrdCo = P04702_A2338PrdCo[0];
      A2340RELCO = P04702_A2340RELCO[0];
      AV39GXLvl1 = (byte)(1) ;
      /* Check if program is canceled. */
      /* Exiting from a For First loop. */
      if (true) break;
   }
   pr_default.close(0);
   if ( ( AV39GXLvl1 == 0 ) )
   {
      AV32Error = (byte)(2) ;
      AV33Errord = "Número de Cuenta de Debito
Incorrecta" ;
   }
```

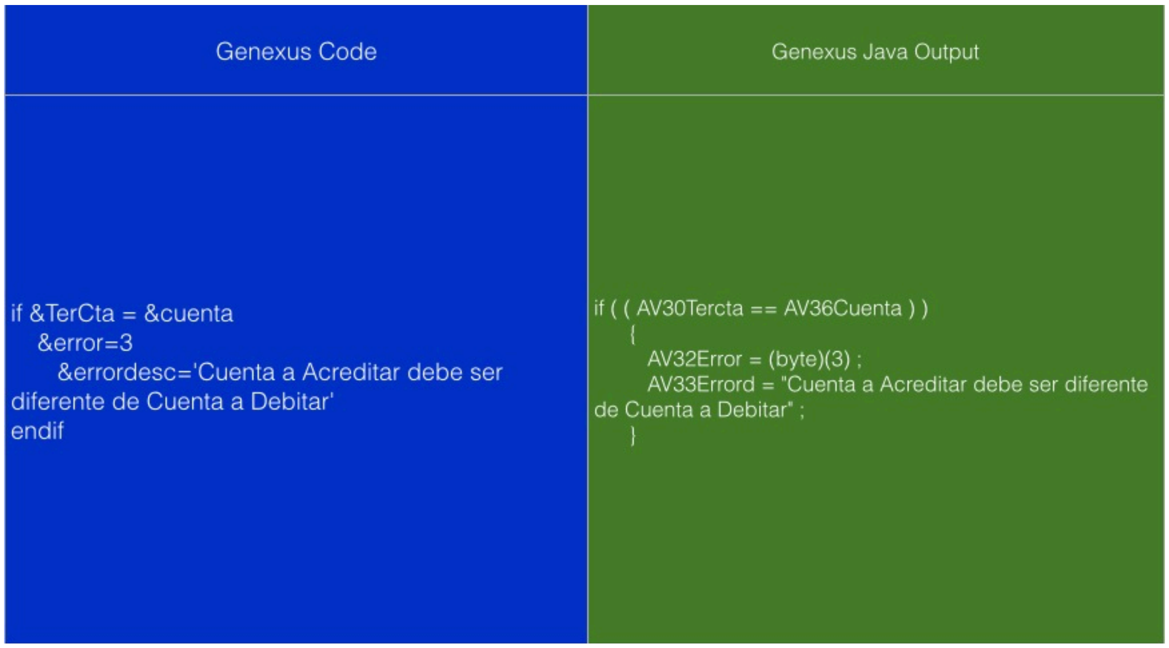
Recommendations:

According to the Java code, a while statement is gathering data from an array and assigning this data to a variable. When completed, then it breaks from the loop and continues the program execution. The recommended action for this validation seems to be independent if statements for each of the variables. For example:

if(A2340RELCO == P04702_A2340RELCO[0] && A2340RELCO != null)

This would allow for more precise handling of each IF statement, including any possible limits that can be implemented.

Third Validation - Error Type = 3 “Credit account must be different to debit account”



Genexus Code	Genexus Java Output

`if &TerCta = &cuenta`
`&error=3`
`&errordesc='Cuenta a Acreditar debe ser diferente de Cuenta a Debitar'`
`endif`
	`if ( ( AV30Tercta == AV36Cuenta ) )`
	`{`
	`AV32Error = (byte)(3) ;`
	`AV33Errord = "Cuenta a Acreditar debe ser diferente de Cuenta a Debitar" ;`
	`}`

Recommendations:

In the current code, an equality comparison is made to determine that the debit account is not the same as the depositing account:

AV30Tercta == AV36Cuenta

Secure Java programming guidelines support that for string/integer comparisons, the most effective way to this is by using a comparison function. For example:

AV30Tercta.equals(AV36Cuenta)

When using the “==“ operator for string comparison you are not comparing the contents of the variable but are actually comparing the memory address, if they are both equal it will return true and false otherwise. Whereas “equals” compares the contents. This will work in most situations but can vary, so it shouldn’t be depended on for proper comparisons. If the use of the “==“ is necessary, then the primitive integer value needs to be evaluated:

AV30Tercta.intValue() == AV36Cuenta.intValue();

Other recommendations:

The observable code had very few documentation lines. This could be a particularity of the application, but it is recommended to include detailed documentation as this can prevent future errors and can make code a lot more secure/error prone.