|  |  |  |
| --- | --- | --- |
|  |  |  |

Secure Coding – **Fase IV**

Pruebas de Vulnerabilidades

GRUPO 4

Presentado por Grupo FivePlus

Álvaro López

Guillermo Molina León

Felipe Giraldo

Hernan Tenjo

Ehrlichmann Casas

2014

## TABLA DE CONTENIDO

TABLA DE CONTENIDO 2

1 INTRODUCCION 3

2 CICLO DE VIDA DEL CONOCIMIENTO DE LAS AMENAZAS Y CONTROLES 4

3 HALLAZGOS APLICACIÓN GRUPO 4 (EL NUESTRO) 5

3.1 Verificación y análisis de nuestros ejecutables (desde el punto de vista de un atacante). 6

4 HALLAZGOS APLICACIÓN GRUPO 1 9

4.1 Ejercicio de Decodificación 14

5 HALLAZGOS APLICACIÓN GRUPO 8 15

5.1 Aplicación C: 15

5.2 Aplicación Java: 20

6 HERRAMIENTAS UTILIZADAS 24

7 LECCIONES APRENDIDAS 25

# INTRODUCCION

El presente documento corresponde al entregable de la cuarta fase para el proyecto de nuestro grupo 4 y del grupo 1 el cual fue asignado por Martin Ochoa.

El documento se enmarca en realizar pruebas de caja negra, blanca y decodificar y analizar los programas en C y Java de otros grupos.

El alcance de esta fase de trabajo es reconocimiento, mapeo, descubrimiento y explotación de vulnerabilidades de la aplicación**.**

# CICLO DE VIDA DEL CONOCIMIENTO DE LAS AMENAZAS Y CONTROLES

Para iniciar nuestra exploración en los ataques a aplicaciones y sus posibles salvaguradas y contramedidas emplearemos unos pasos los cuales son: Conocimiento del objetivo, Mapeo de las posibles vulnerabilidades, Descubrimiento de las mismas (no sea falsos positivos) y por último Explotar dichas vulnerabilidades.



Gráfica tomada del libro The Web Application Hacker´s Handbook – Dafydd Stuttard, Marcus Pinto

* Primero realizamos unas pruebas de caja negra sobre la aplicación donde deberiamos haber controlado y/o arreglado las vulnerabilidades descubiertas por nosotros y por el grupo 5.
* En el análisis de caja negra es posible que surjan vulnerabilidades de los nuevos desarrollos e integraciones.
* El siguiente análisis es de caja blanca donde se verificará el código de los programas en PHP, JAVA y C respectivamente, en forma manual y automática.
* Por último se deberá decodificar los programas del grupo 8 y grupo 1 en C/C++ y Java.

# HALLAZGOS APLICACIÓN GRUPO 4 (EL NUESTRO)

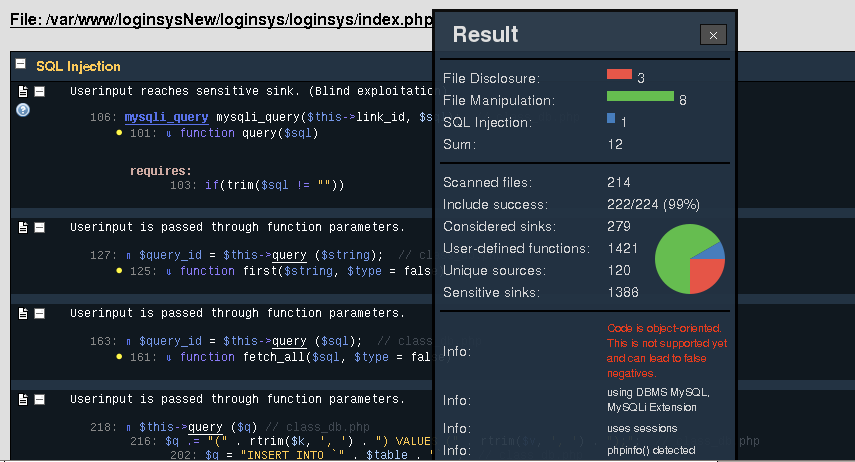
La aplicación se migro y se configuro con las siguientes caracteristicas:

* Se tiene un servidor sin interfaz gráfica ni componentes que puedan generar riesgos en el ambiente a la aplicación, se cerraron los puertos que no estan en uso.
* Se instalo un apache y un motor MySql con usuarios privilegiados y sus contraseñas robustas y definiendo usuarios con permisos limitados para el uso de los servicios por parte de las aplicaciones construidas.
* Se deshabilito el puerto 80 y se habilito todo el trafico por el 443 con https.
* Se crearon reglas de filtrado de paquetes a nivel de kernel para controlar el tráfico de información a través de los puertos.

Se detectaron las siguientes vulnerabilidades, despúes de realizar las pruebas de caja negra y blanca en forma manual y/o automática:

|  |  |  |
| --- | --- | --- |
| **VULNERABILIDAD** | **RIESGO** | **CONTROL** |
| Las Páginas de seguridad se dejan en cache, no existe un control de cache. En las páginas de <https://192.168.69.136/loginsys/theme/css/login.css>, https://192.168.69.136/loginsys/admin | Medio | <META HTTP-EQUIV='Cache-Control' CONTENT='no-cache'> |
| El atributo AUTOCOMPLETE no ha sido deshabilitado de todas las pantallas de la aplicación.  <https://192.168.69.136/loginsys/register.php>  https://192.168.69.136/loginsys/admin/login.php | Bajo | Use el atributo AUTOCOMPLETE='OFF' |
| La Ip del servidor es pública https://192.168.69.136/ | Medio | Se debe colocar un alias en el Apache |
| Las Cookies esta configuradas sin el Secure Flag, lo cual se puede acceder todavía a las cookies sin cifrar. PHPSESSION.  https://192.168.69.136/loginsys/lib/captcha.php | Medio | Asegurar que las cookies tengan el secure flag. |
| Se utiliza sprintf en transactionManager.c | Alto | Utilice snprintf |
| Se puede explotar un buffer overflow con fixed size global buffer en transactionManager.c | Alto |  |
| El uso de atoi en transactionManager.c no se controla y puede ser sobrecargado para exceder el rango de las variables | Medio | Se puede salvar o almacenar un valor sin signo |
| El uso de fopen transactionManager.c no esta controlado y puede utilizarse para redireccionar. | Alto | Se puede forzar a abrir determinado tipo de archivos o en forma controlada. |
| Se utiliza strncpy en transactionManager.c | Bajo | Utilizar strccpy o controle el carácter \0 |
| En loadFile.php se manipula archivo y no se ve un control sobre las entradas | Bajo | Utilizar API de control de listas blancas. |
| En el archivo config.php.ini | Medio | No dejar implicitas las contraseñas ni usuarios privilegiados |

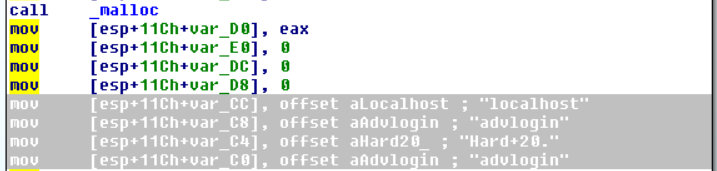
La aplicación RIPS es una excelente herramienta para pruebas de caja blanca a PHP, pero nos generó varios falsos positivos los cuales los corroboramos.



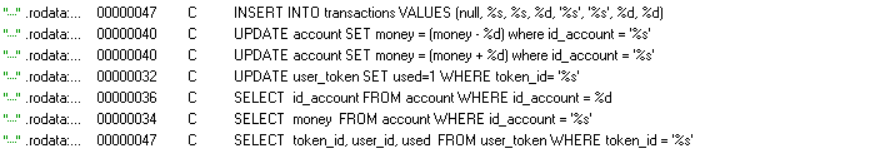
## Verificación y análisis de nuestros ejecutables (desde el punto de vista de un atacante).

Se examine el código ejecutable del programa batch de ejecución de transacciones (Herramienta IDA-PRO):

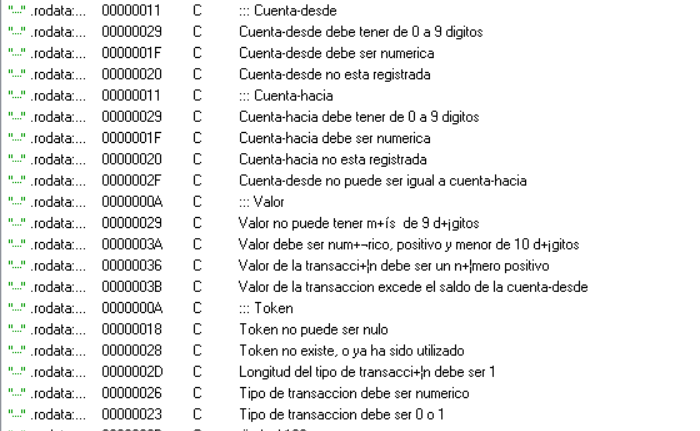
1. El usuario y password de la base de datos están quemados y son visibles:



1. Las instrucciones de SQL son totalmente visibles:



1. Las validaciones realizadas sobre el input son totalmente visibles



1. El procedimiento de creación de la llave AES es transparente al observar la rutina getAESKey que utiliza una llave quemada y el algoritmo SHA-256



# HALLAZGOS APLICACIÓN GRUPO 1

La aplicación se configuro para navegar por el puerto 443 mediante https, pero se dejo en el mismo servidor de samurai WTF el cual tiene muchas vulnerabilidades de ambiente, las cuales no públicaremos y nos dedicaremos solo a las de la aplicación.

Se verificaron los 21 módulos de PHP en sus diferentes subdirectorios MVC se encontró:

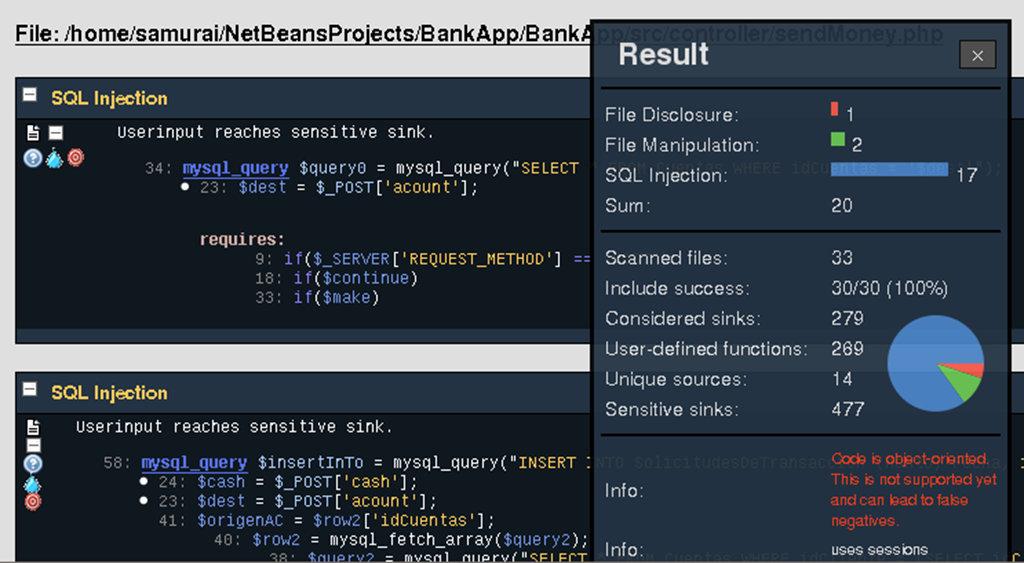
* Manejo de listas blancas
* Escape de caracteres especiales
* Validadores de XSS, Sql Injection y CSRF (por ejemplo ver checklogin.php)

Se verifico el estandar y verisón del SSL utilizado para verificar posibles vulnerabilidades conocidas:

* Version: 2
* Serial Number: -4294967295
* Signature Algorithm: sha1WithRSAEncryption
* Issuer: /C=CO/ST=Bogota/L=BOG/O=BankApp/OU=SecureTeem/CN=127.0.0.1/BankApp//emailAddress=bbnkapp@gmail.com
* Todas las cuentas por defecto tienen la contraseña por defecto
* No pudimos observar que la aplicación tuviera control de acceso
* Cuando un usuario se registra le abona un monto al usuario … regala plata
* Se pudo ingresar a las tablas

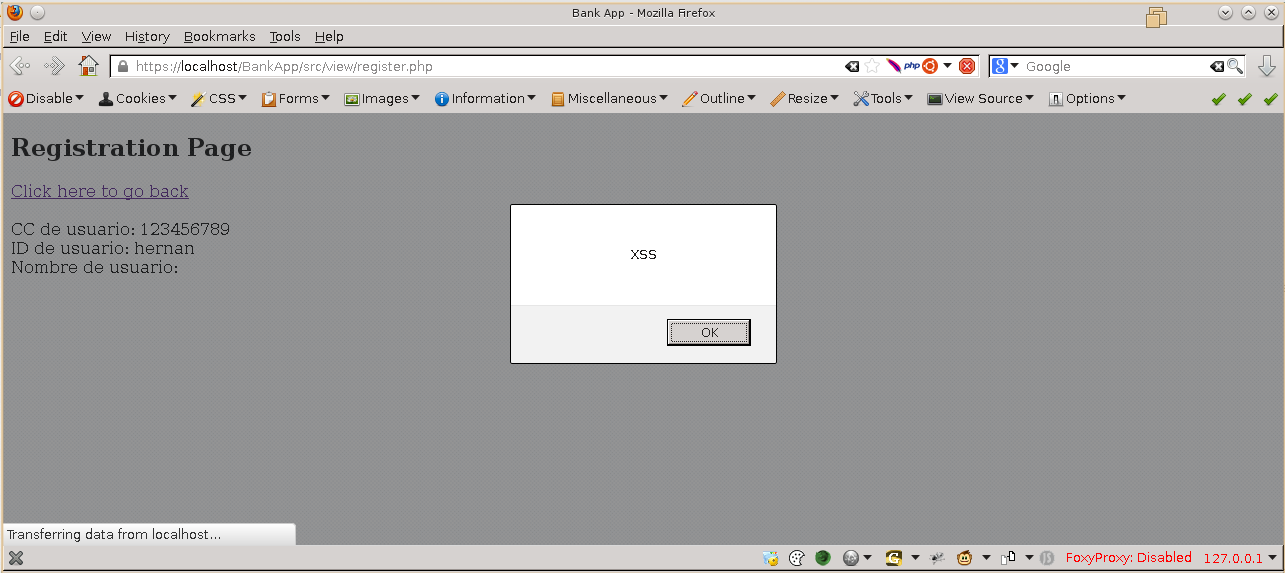
Se detectaron las siguientes vulnerabilidades, despúes de realizar las pruebas de caja negra y blanca en forma Manual y/o automática en las :

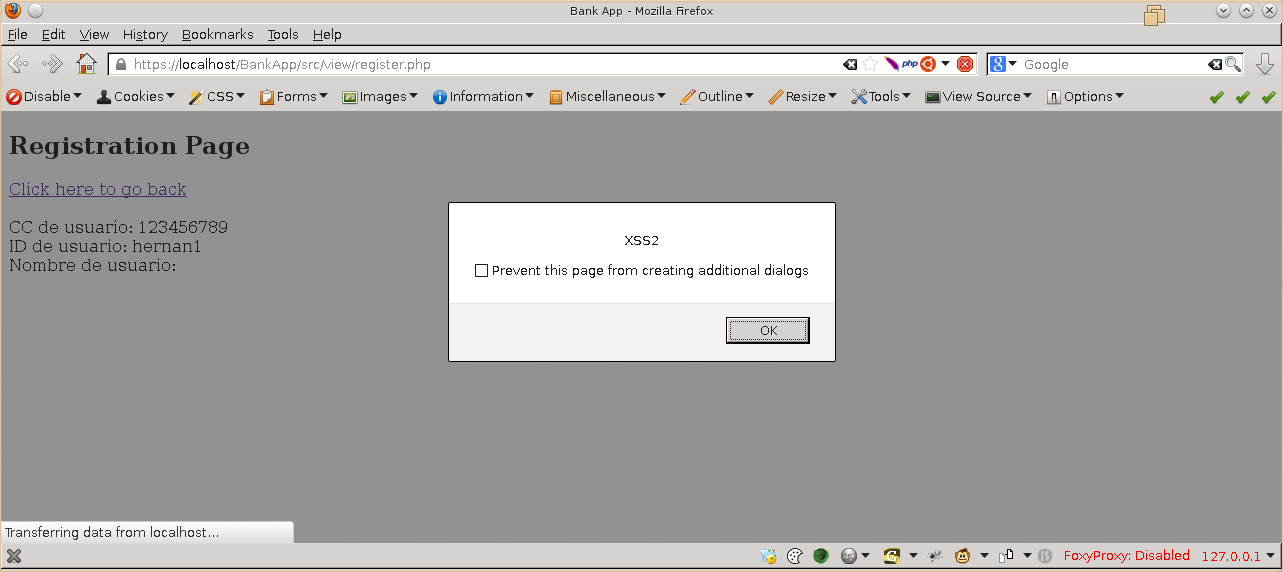
|  |  |  |
| --- | --- | --- |
| **VULNERABILIDAD** | **RIESGO** | **CONTROL** |
| Las Páginas de seguridad se dejan en cache, no existe un control de cache. En las páginas de <https://localhost/BankApp/src/view/register.php>  https://localhost/BankApp/src/view/login.php | Medio | <META HTTP-EQUIV='Cache-Control' CONTENT='no-cache'> |
| El atributo AUTOCOMPLETE no ha sido deshabilitado de todas las pantallas de la aplicación. | Bajo | Use el atributo AUTOCOMPLETE='OFF' |
| Los usuarios y contraseñas por defecto siguen originales | Alto | Cambiar las contraseñas de Mysql y Servidor |
| Cuando la aplicación BankApp/BankApp/src/controller/checklogin.php pasa los parámetros al correo, estos pueden ser redireccionados a un servidor de correo remoto | Alto | Verificar las entradas y proteger la transmisión, puede ser con PGP o cualquier sistema de seguridad de correos. |
| El fopen en BankApp/BankApp/src/controller/send.php no esta controlado y puede utilizarse para redireccionar | Alto | Se puede forzar a abrir determinado tipo de archivos o en forma controlada. |
| Deja el código de Mysql quemado(ver línea 27 del archivo checklogin.php) | Alto | Parametrizar las variables sensibles |
| Aun cuando se utilizan las librerías de escapado de MySql, El código puede recibir inyección dado que es posible inyectar la consulta (línea 29 checklogin.php). | Alto | Cambiar forma de construir y ejecutar las consultas a la BD. |
| No se validan entradas en el registro de usuarios | Alto | Se deberían restringir los datos que se pueden ingresar a los campos. |
| No se valida la fortaleza de las contraseñas | Alto | Se aconseja verificar reglas mínimas para la construcción de claves, para minimizar los riesgos de ataques de fuerza bruta. |
| Las salidas verificadas en el registro permiten la inyección de XSS | Alto | Se deberá escapar toda la información ingresada por el usuario en el momento de la salida y no solamente en el ingreso. |
| Realizando pruebas de caja blanca, no se encontró el uso de PreparedStatements | Alto | Se aconseja transformar las consultas para que manejen los binding propuestos por los PreparedStatements y así evitar las inyecciones en un 100% |
| Se encontraron rutas quemadas dentro del código php que podrían dar info adicional. | Bajo | Sacar las rutas físicas del código ya sea a la BD o a algún ambiente con un mínimo de seguridad. |
| Al parecer los TAN se encuentran guardados de forma plana en la BD, pero algo falla en la transferencia. | Alto | Se deberá guardar encriptada la información. |
| Se encontraron credenciales planas en la BD (claves, usuarios, etc) | Alto | Se deberá guardar encriptada la información. |
| Se encontró que la aplicación puede ser cargada desde un frame (Framing). | Bajo | Se aconseja agregar las directivas necesarias o los encabezados js que eviten la carga del sitio desde un dominio diferente. |

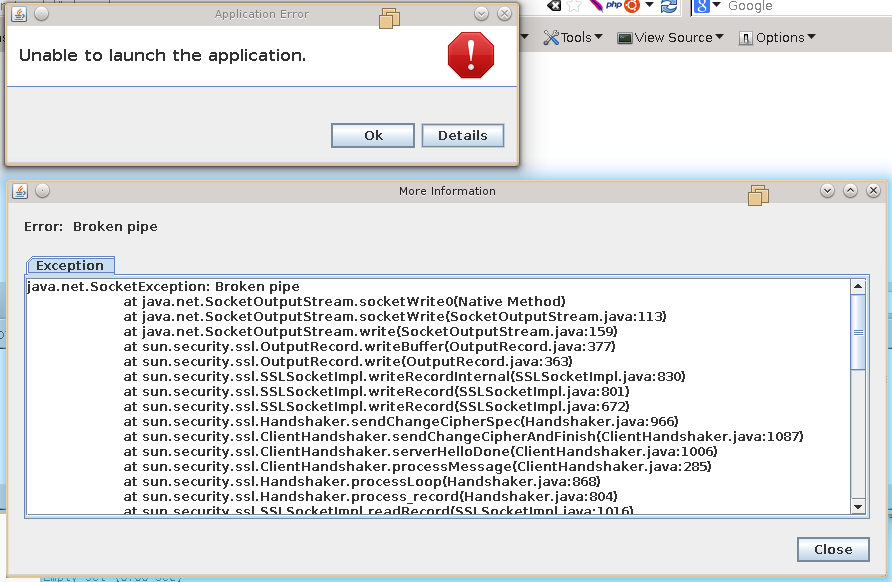


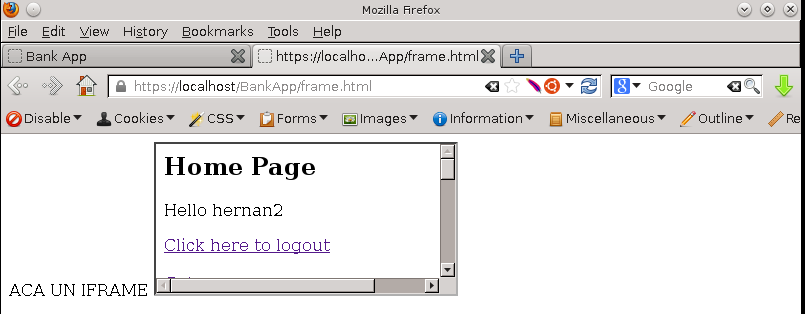
|  |  |  |
| --- | --- | --- |
| Top 10 OWASP | | |
| No. | **Descripción** | **Estado** |
| 1. | Inyección | Vulnerable |
| 2. | Pérdida de autenticación y gestión de sesiones | OK |
| 3. | XSS | Explotado |
| 4. | Referencia directa insegura a objetos | OK |
| 5. | Configuración de seguridad incorrecta | Explotado |
| 6. | Exposición de datos sensibles | Explotado |
| 7. | Ausencia de control de acceso a funciones | OK |
| 8. | CSRF | Vulnerable |
| 9. | Uso de componentes vulnerables | OK |
| 10 | Redirecciones y reenvíos comprometidos | OK |











Mejoras de seguridad encontradas, según recomendaciones generales:

* Uso de protocolo seguro https, para todo el sitio
* Utilización del flag secure en las cookies.
* Utilización del flag httpOnly en las cookies
* No se visualizaron links de acceso directo a objetos

## Ejercicio de Decodificación

Al descompilar la aplicación JAR, encontramos:

* La APIKey está totalmente visible: "AIzaSyB8azikXJKi\_NjpWcVNJVO0dGFG1WuNJlg"
* Para la creación de contenido y prueba de los mensajes, se utiliza la semilla:

c.addRegId("APA91bFqnQzp0z5IpXWdth1lagGQZw1PTbdBAD13c-UQ0T76BBYVsFrY96MA4SFduBW9RzDguLaad-7l4QWluQcP6zSoX1HSUaAzQYSmI93hMJQUYEdRLpBOpmAGkjckuoVFt6icRjgXKuOpqZEedWUvVsKOqRruXuAe3mDbkimcNAMpc7XMF8M");

* Se tiene una constante para el serial:

private static final long serialVersionUID = 0x44f4195ab35e7f24L;

* Para el cifrado del identificador de la sesión utiliza MD5 plano.
* Los mensajes de error son bien utilizados. Sin embargo, se preferiría algo más estándar y no tan específico para no dar pistas de lo que se está haciendo.
* Debido a la estructura utilizada en el código, y al tener la APIKey 'quemada' dentro del código, es perfectamente posible realizar una aplicación que genere códigos válidos para transacciones fraudulentas.

Posibles Errores ubicados:

* Se utiliza un método de cifrado débil (MD5)
* Almacena información sensitiva (APIKey/REgID) sin cifrado alguno del lado del cliente (en el jar)
* No utiliza anotaciones para proteger el código
* No se aprecian medidas implementadas para recobrarse de errores de sistema
* No hay ofuscación del código, lo que lo hace totalmente legible y replicable.

Buenas Prácticas:

* Minimiza la visibilidad de las variables

# HALLAZGOS APLICACIÓN GRUPO 8

Dado que el grupo 1 no disponía de programas en C, se pidió la autorización a Martín Ochoa de evaluar otro grupo en este punto para lo cual nos fue asignado el grupo 8.

Este grupo tomo medidas para asegurar que el acceso a la información fuera lo más complicado posible. Utilizan algoritmos de cifrado como Base64 y RSA, e incluyen un certificado para guardar la llave de cifrado.

Sin embargo, no se aprecian medidas que hagan que la aplicación se recupere de fallos graves sin comprometer la privacidad de los datos. Por otro lado el código esta sin ofuscar.

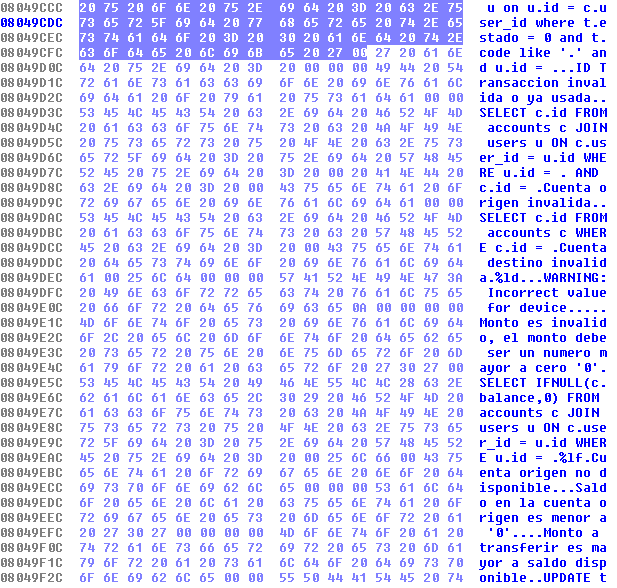
## Aplicación C:

Los datos de conexión a la base de datos están explícitos, como veremos más adelante:

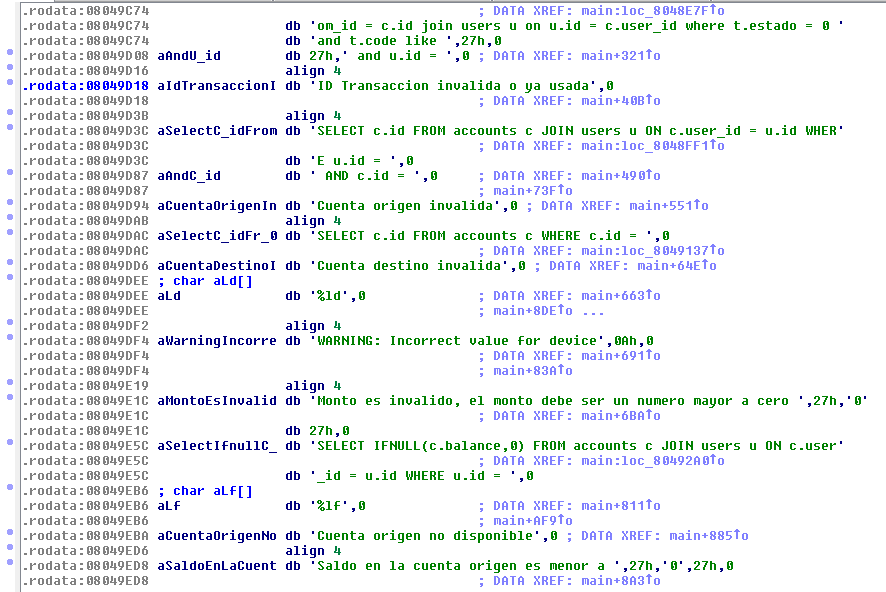
* servidor: localhost
* usuario: root
* clave: 29ft230ft
* DB: securecoding



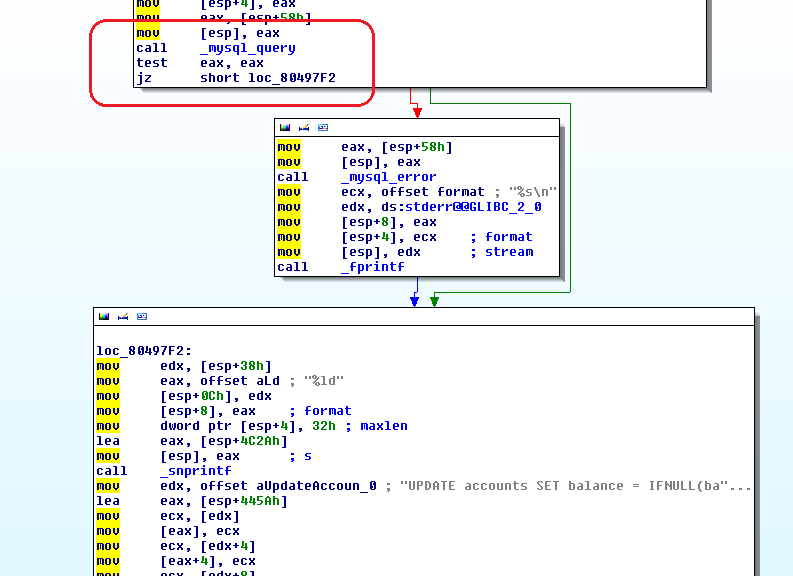
Las consultas a la base de datos utilizadas dentro de la aplicación, se pueden ver claramente.



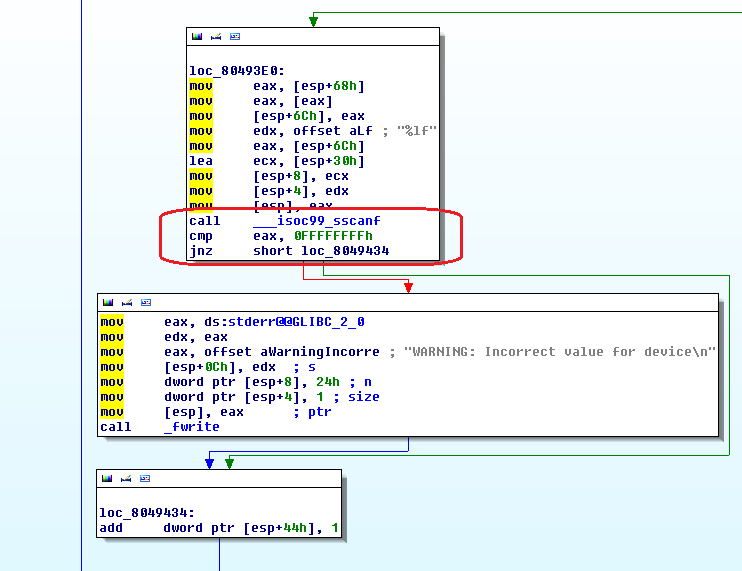
En el análisis realizado con la herramienta **IDA Pro**, se puede apreciar el uso de funciones no seguras como printf, fgets, strcat, strcopy, lo cual puede llevar a ataques de Buffer Overflow, lo anterior si el Malloc utilizado no está lo suficientemente bien protegido.



Las comparaciones de los resultados obtenidos desde la base de datos, son realizados de manera directa, lo que permitiría, mediante la modificación del archivo compilado (ej. con un editor hexadecimal), el aprobar transacciones que normalmente no se deberían aprobar, mediante el cambio de una instrucción JNE (o una JZ) por una JMP. E igualmente enviar inyecciones ciegas al código.



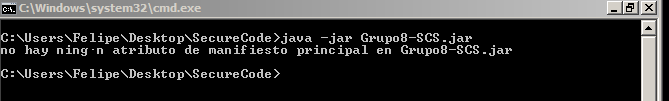
Igualmente, aplica también para los valores generados por la SmartCard.



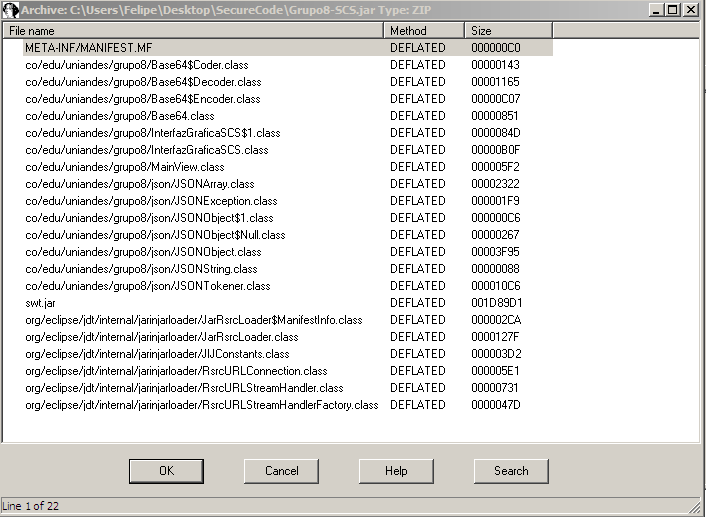
Creemos que no se utilizaron buenas prácticas como Prepared Statements para el procesamiento de la información que va a ingresar a base de datos, lo cual es muy probable un ataque de inyección.

## Aplicación Java:

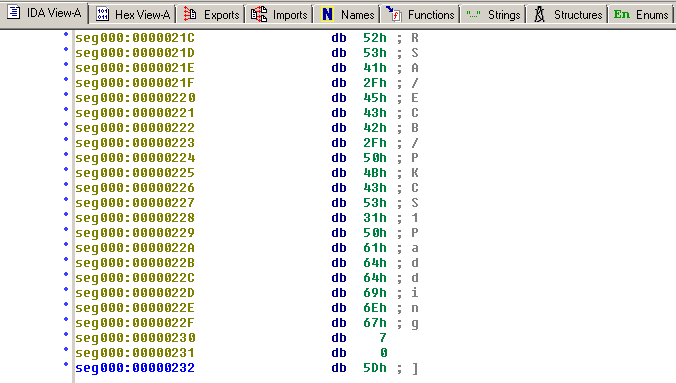
No ue posible poner en funcionamiento la aplicación Java del grupo 8, sin embargo se realizo análisis de caja blanca y se decompilo el jar entregado, para lo cual obtuvimos los siguientes resultados:

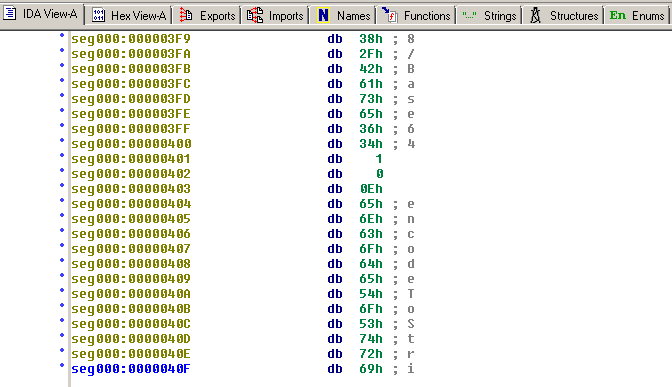


* Estructura de la aplicación.

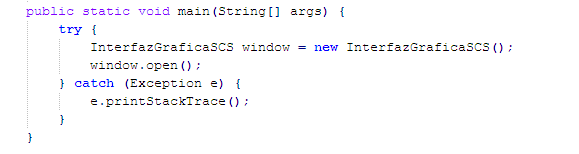


* Algoritmo usado para proteger los datos desde el punto de vista de “confidencialidad”. Se usa la clase Base64 proporcionada por las librerias de apache. También se puede visualizar con la herramienta IDA que usan RSA.

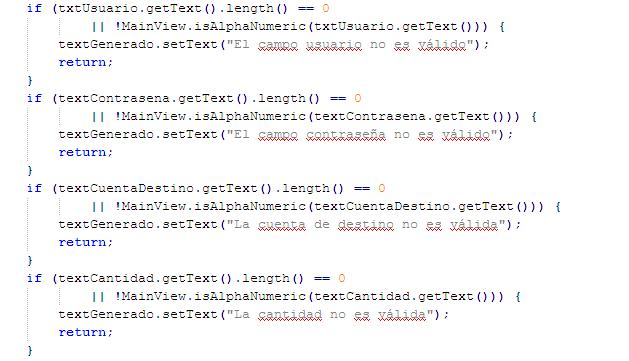




* Cadenas usadas que permiten tomar puntos de referencia en el código.
* Manejo inadecuado de errores y excepciones cuando la aplicación falla.



* Validaciones faltantes cuando los campos con datos ingresados por el usuario son nulos.



# HERRAMIENTAS UTILIZADAS

Herramientas utilizadas y detalladas en anteriores fases.

* nMap
* Sql Map
* ZAP
* Burpsuite
* Dirbuster

Nuevas Herramientas como complemento en esta fase

* **PDM,** scans Java source code. Es una herramienta de análisis de código estático para vulnerabilidades en JAVA.
* **RIPS,** Es una herramienta de análisis de código estático para vulnerabilidades en PHP.
* **RATS,** Es una herramienta de análisis de código estático para vulnerabilidades en C/C++, PHP entre otros.
* **IDA Pro**, es una herramienta de desambladora y depuradora de código (disassembler and debugger) utilizada para ingeniería inversa, recursiva, permite su manipulación en el disassembler, es bastante extensible mediante plugins y scripts y es multiplataforma. Lo más interesante e importante es que soporta varios ISA.
* **DJ Java Decompiler**, Herramienta que produce los programas fuentes de un archivo .class.

# LECCIONES APRENDIDAS

1. Las pruebas de caja negra nos dan una perspectiva desde el punto de vista de un atacante el cual no tiene contacto ni acceso directo a nuestros servidores. Con esto estamos representando un gran porcentaje de atacantes y se toman sus perspectivas mediante esta prueba.
2. Mediante las pruebas de caja blanca, no solo podemos determinar fallas de seguridad sino mala programación que de una u otra manera, impactando en el rendimiento de la misma. Como pudimos corroborarlo con la herramienta PDM, la cual nos indico muchas falencias o malos usos de las variables que sabiendolas explotar podríamos generar vulnerabilidades a partir de mala programación.
3. Aprender a decompilar el código y entenderlo es un arte, muchos de nosotros no lo habiamos realizado y menos con el objetivo de vulnerar al mismo.
4. En el uso de herramientas especializadas como IDA Pro se requiere bastante expertis y comprender el código víctima. Recomendamos primero navegar por la aplicación victima y comprenderla, para luego si poder tener puntos de ataque o mensajes como pivots de guías en la comprensión del código descompilado.
5. Es importante asegurar la infraestructura base de la aplicación, pues en diferentes escenarios se toman todos los controles pertienentes para evitr diferentes tipos de ataques a nivel del software construido, pero se descuidan las aplicaciónes en las cuales nuestros sistemas se soportan, por ejemplo servicios de almacenamiento, de contenido Web y controles sobre el sistema operativo.
6. Definitivamente, cuando se realiza programación a destajo, sin la planeación adecuada, se incurre en situaciones qe comprometen la seguridad de la infraestructura, olvidadndo detalles de seguridad para priorizar detalles funcionales.