# Primera Guía de Código Seguro en BASE24

OCELOT | METABASE Q

# Índice

# Introducción

En la actualidad los sistemas de medios de pago se están enfrentando a una realidad en constante cambio. Nuevas formas de pago, nuevos canales para realizar compras y nuevas validaciones de la persona tarjetahabiente están forzando a los sistemas autorizadores (POS, ATM, *E-commerce, E-wallet*, etc) a ser más estrictos con las validaciones y reglas que deben de efectuar al momento de procesar una transacción, así como tener perfectamente definidos los parámetros que permitan o no que una transacción sea autorizada.

Durante este flujo, resultados inesperados pueden llevar a consecuencias catastróficas para las entidades financieras. Una simple confusión entre las reglas que aplican para la validación de una tarjeta de banda magnética contra una tipo *Contactless*, o una confusión en la parte que debe autorizar (*Switch* o Banco), puede ser suficiente para generar autorizaciones inesperadas que causen pérdidas millonarias a los Bancos o *Switches* bancarios.

Dentro de la revisión de las mejores prácticas de código seguro, se debe de incluir una revisión de errores lógicos en donde se haga una inspección minuciosa dentro del código de cada paso durante la validación y "ruteo" de una transacción. Esto requiere de un equipo experto en dichos procesos, con conocimiento tanto en el lenguaje TAL como en los sistemas de medios de pago (BASE24, Connex) y, desde luego, experiencia en la identificación de vulnerabilidades y ambientes que permitan identificar proactivamente los ataques, así como corregir a corto y largo plazo los huecos de seguridad presentes.

La presente guía tiene la intención de ser un precedente y sentar las bases para establecer mejores prácticas de programación en la escritura de código legado, específicamente en lenguaje TAL y sobre todo aquel que corre en sistemas de Transferencia Electrónica de Fondos (EFT, por sus siglas en inglés) como BASE24 y Connex. Los errores de código seguro que se presentan en esta guía son el resultado de un análisis exhaustivo realizado en el sistema BASE24, el cual ha sido modificado por entidades proveedoras diferentes a las propietarias del *Software* (ACI *Worldwide*). En muchas ocasiones el sistema es modificado por diferentes proveedores a lo largo de los años sin seguir ningún tipo de estándar en específico, dejándose guiar únicamente por la experiencia personal, la practicidad o rapidez, y en ocasiones, sin el conocimiento de las reglas propias del lenguaje TAL o de las reglas de transaccionalidad que establece el sistema BASE24.

Los errores que se muestran en esta guía están clasificados por categorías dependiendo del tipo de impacto que puedan tener y la criticidad de los mismos. Se plasman también las recomendaciones a seguir para evitar este tipo de errores y se incluyen diagramas para comprender de mejor manera el esquema modular que maneja BASE24, así como los módulos donde se puede presentar la vulnerabilidad. Igualmente se presenta un diagrama para entender de mejor manera el manejo de la memoria dentro del código escrito en TAL.

## Créditos

Equipo de Metabase Q que participó en la creación de esta primera guía, en orden alfabético:

Alejandro Gómez Bucio
Dana Paola Valle Arroyo
José Antonio Alcalá López

# *Bad Memory*

Errores que pueden provocar corrupción de la memoria, ocasionando rechazos, *Abends* o mal funcionamiento de los procesos.

| *Bad Memory* | *Bugs* | Riesgo e impacto |
|---|---|---|
| 1.0 | 1.1 *Bad Index* | Errores donde no se consideran todas las ocurrencias de una tabla, pudiendo descartar algunos elementos o invadir los límites de otros. En el primer caso, si hay una omisión de alguna ocurrencia, se puede afectar la transacción en proceso derivando en un rechazo, lo que afecta la **disponibilidad.**<br>En el caso de las lecturas mayores al número de ocurrencias cargadas en la tabla, el problema es que durante la lectura, en caso de no encontrarse el dato que se está buscando, la lectura seguirá hasta que se provoque un error de *Out of Bounds*. En otras palabras, se puede traer información que se encuentra disponible en la siguiente dirección de memoria, lo que puede ocasionar corrupción de datos de otras variables, provocando desde un rechazo hasta un *Abend* del proceso y afectando su **disponibilidad.** |
| | 1.2 *Bad Token* | Errores de validación del resultado en las operaciones realizadas con los *Tokens* pudiendo provocar rechazos, corrupción de memoria o caídas de los procesos, afectando la **disponibilidad** del servicio e **integridad** de la información. |
| | 1.3 *Bad Variables* | Errores relacionados con mal manejo de variables: corrupción de datos o validar los errores que regresan las rutinas y aun así continuar con el flujo de la transacción, pudiendo provocar rechazos por datos inválidos, afectando la **disponibilidad** del servicio. |

## Recomendaciones *Bad Index*

- En el caso de las lecturas mayores al número de ocurrencias cargadas en la tabla, el problema se presenta durante la lectura, ya que en caso de no encontrarse el dato que se está buscando, la lectura seguirá hasta que se provoque un error de *Out of Bounds*, ocasionando un *Abend* en el RTAU. También se deben de revisar los parámetros del LCONF indicados en el análisis para validar cuántos datos reales se pueden estar omitiendo en la lógica del programa.
- Otra buena práctica es usar un campo de índice dentro de la tabla que se vaya incrementando al momento de hacer la carga en memoria. Esto nos ayudará a tener visión de cuántas ocurrencias tenemos cargadas en memoria, sin importar que la tabla se haya definido con un tamaño mayor previendo el aumento de información o para su uso futuro.

## Recomendaciones *Bad Token*

- Dado el manejo que ya existe para los *Tokens* implementado por ACI, lo más recomendable es apegarse a estos estándares cada vez que se hace la manipulación de *Tokens*, validando el tipo de error y actuando en consecuencia al mismo; evitando el mal manejo de los apuntadores de memoria, de los mensajes en el *Log* en caso de una falla y por último, asegurarse que la información que se envía a las interfaces sea la correcta para evitar rechazos difíciles de diagnosticar.

## Recomendaciones *Bad Variables*

- La forma correcta y **la mejor práctica** para hacer movimientos de variables para evitar inconsistencias en los datos, pérdida de información importante para la transacción o *Buffer Overflow*, es **utilizar siempre** la utilería de TAL **$len**, que toma la longitud exacta del campo destino y que se mueve como máximo al campo destino. Por otro lado, es importante inicializar (limpiar) antes y después los *Buffers* en memoria.

# *Bad File*

Errores en las operaciones con archivos que pueden provocar mal funcionamiento de transacciones, corrupción de archivos y caídas en los procesos.

| *Bad File* | *Bugs* | Riesgo e impacto |
|---|---|---|
| 1.0 | 1.1 *Closing file* | Errores relacionados con el cierre de archivos después de su manipulación o después de ocurrir una falla, ocasionando que queden copias abiertas sin utilizar.<br>Dentro de BASE24 existe una tabla encargada de agregar o eliminar los archivos que utilizan los módulos. La cantidad de archivos que puede manejar este *File Table* está determinado por el Segmento de Procesamiento de Archivos (PFS, por sus siglas en inglés). Estos archivos se reconocen por un *File Number* (Número con el que se identifica un archivo dentro de los procesos de *Tandem*). Si un archivo no se cierra de manera explícita, este seguirá utilizando un espacio previamente designado dentro de la tabla, eventualmente causando un agotamiento de memoria. Al existir varias copias del mismo archivo que no fue cerrado, llegará un punto en el que se llene la tabla y se caiga el proceso. Esto puede derivar en una **denegación de servicio.** |
| | 1.2 LCONF *read* | Mala lectura de los parámetros o asignaciones en el archivo LCONF, causando obtención de información incompleta o errónea que afecte la transaccionalidad, lo que provoca **rechazos** o comportamiento extraño durante el proceso de la transacción.<br>El problema de este tipo de lectura, y que se ha vuelto un estándar en los RPQ/CSMs realizados por entidades proovedoras externas dentro del código, es que no sigue las mejores prácticas de codificación en TAL ni las establecidas por ACI para el manejo de este archivo. El riesgo es que cuando se lleguen a agregar parámetros iguales hasta la posición que se está leyendo y que, a partir de ahí cambie de valor, no se sabrá con certeza qué parámetro se leyó, pudiendo provocar desde **inconsistencias en los datos** hasta **negación de servicio** debido a **rechazos**. Ya que los parámetros solo se cargan al iniciar el proceso o ejecutar un *Warmboot*, la información traída incorrectamente estará todo el tiempo presente. |
| | 1.3 *Keypostion* | No se valida si se pudo posicionar el apuntador correctamente para la lectura en un registro, pudiendo provocar ***Abends*** en los programas, lo que afecta la **disponibilidad** del servicio.<br>Si se realiza *el Keyposition* sin validar si hubo error en la rutina o si el error es diferente de EOF (*End Of File*), significaría que sucedió algo con el archivo. Por lo tanto, habría un problema al momento de realizar el *Read* que no se está validando.<br>La severidad de esto y las consideraciones a tomar también dependen de la importancia de la información que contiene el archivo.<br>Ver sección **Información general sobre el manejo de archivos relacionada a la familia *Bad File***. |

| | | |
|---|---|---|
| | 1.4 *Read, Write, Open* | No se valida el resultado de estas operaciones en archivos, asumiendo que todo sucede de la manera en que se espera, pudiendo provocar *Abends*, corrupción de registros o bloqueo de registros, lo que afecta la **disponibilidad** del servicio. |

## Recomendaciones *Closing File*

- La recomendación es realizar el cierre de cada archivo que se abre previamente, o dependiendo del caso, agregar una rutina de *Warmboot* para este archivo. Así cada vez que se ejecute un *Warmboot* se reiniciará también el archivo en la *File Table*.

## Recomendaciones LCONF *Read*

- Se recomienda ajustar la lectura de los parámetros e ir de la mano con el estándar manejado por ACI.

## Recomendaciones *Keyposition*

- Cuando se hace un *Keyposition* se recomienda validar si hubo algún error. Dependiendo del número de error que arroje, si se detecta que es algo diferente a EOF, entonces se recomienda hacer un *File Info* para conocer el detalle del error o en su caso, meter una lógica para manejarlo y notificar la falla en el *Log* EMS.

## Recomendaciones *Read, Write, Open*

- Se recomienda que siempre que se realice un llamado a la rutina k30^open se evalúe el resultado de la variable donde retorna el error de la operación.
- Se recomienda que siempre que se realice un llamado a la rutina k60^writex se evalúe que el registro haya sido modificado o escrito correctamente.

## *PCI Violation*

Código escrito carente de un manejo de las mejores prácticas de seguridad, que podría impactar en la **confidencialidad** de los datos del tarjetahabiente siendo procesados, mismos que podrían ser guardados en disco sin ser enmascarados o encriptados (basado en el estándar PCI DSS 4.0).

| *PCI Violation* | *Bugs* | Riesgo e impacto |
|---|---|---|
| 1.0 | 1.1 *PCI Violation Security Control* 3.2 | Al enviar el número en claro, datos sensibles u otra información de la transacción como el número de secuencia, se está **violando la normativa PCI**. Se trata datos sensibles que, en caso de que llegaran a ser interceptados, se podría hacer mal uso de esa información provocando **fraudes**. Además de que la institución que tiene el sistema BASE24 pudiera incurrir en algún tipo de sanción por este tipo de fallas, el principalproblema es que queda en riesgo la **confidencialidad** de la clientela. |

## Recomendaciones

- La recomendación es enmascarar el número de tarjeta antes de guardarla en un *Log* u otro archivo en disco.

# *Abends*

Interrupciones inesperadas o no controladas en los procesos de BASE24 (conocido como “abendear”) que afectan la disponibilidad, pérdida de transacciones y la continuidad del negocio.

| *Abends* | *Bugs* | Riesgo e impacto |
|---|---|---|
| 1.0 | 1.1 Por *Token* | Se manda a **detener el proceso** cuando ocurre un error en alguna operación con los *Tokens* dentro del mensaje. Generalmente tienen que ver con problemas serios relacionados con la memoria. El problema radica en que los procesos de BASE24 deben de “**abendear**” solo en caso de que sea absolutamente necesario para evitar afectaciones en la **disponibilidad** del servicio. |
| | 1.2 Por corrupción de memoria | Se hace uso de apuntadores que no están correctamente inicializados, que apuntan a direcciones de memoria inválida o que apuntan a información diferente de la que se supone que debe de apuntar. Esto debido a que no se valida el resultado de rutinas que hacen operaciones con los mismos y se asume que la operación fue satisfactoria, continuando con el proceso de la transacción que puede provocar **truenes** difíciles de corregir, afectando la **disponibilidad** de los servicios e incluso hasta la **continuidad del negocio**. |
| | 1.3 Por carga de archivo | Se manda a **detener el proceso** cuando no se puede cargar correctamente en memoria un archivo **no esencial** para la operación del proceso. |

## Recomendaciones por *Token*

- Los programas de BASE24 deben de “abendear” en la minoría de los escenarios posibles para evitar la afectación del servicio y/o provocar encolamientos de transacciones. Lo que se recomienda es apoyarse de las herramientas con las que ya se cuentan en el programa.

## Recomendaciones por corrupción de memoria

- Cuando se hace la lectura de *Tokens* se recomienda validar si no hubo algún error fatal. Ya que, desde ahí, se puede identificar si hay un problema de corrupción de memoria o de mal formateo del mensaje. Esto evita que el error se convierta en una bola de nieve que genere una afectación mayor. Desde rechazos, hasta que el proceso se pueda detener por corrupción de memoria.

## Recomendaciones por carga de archivo

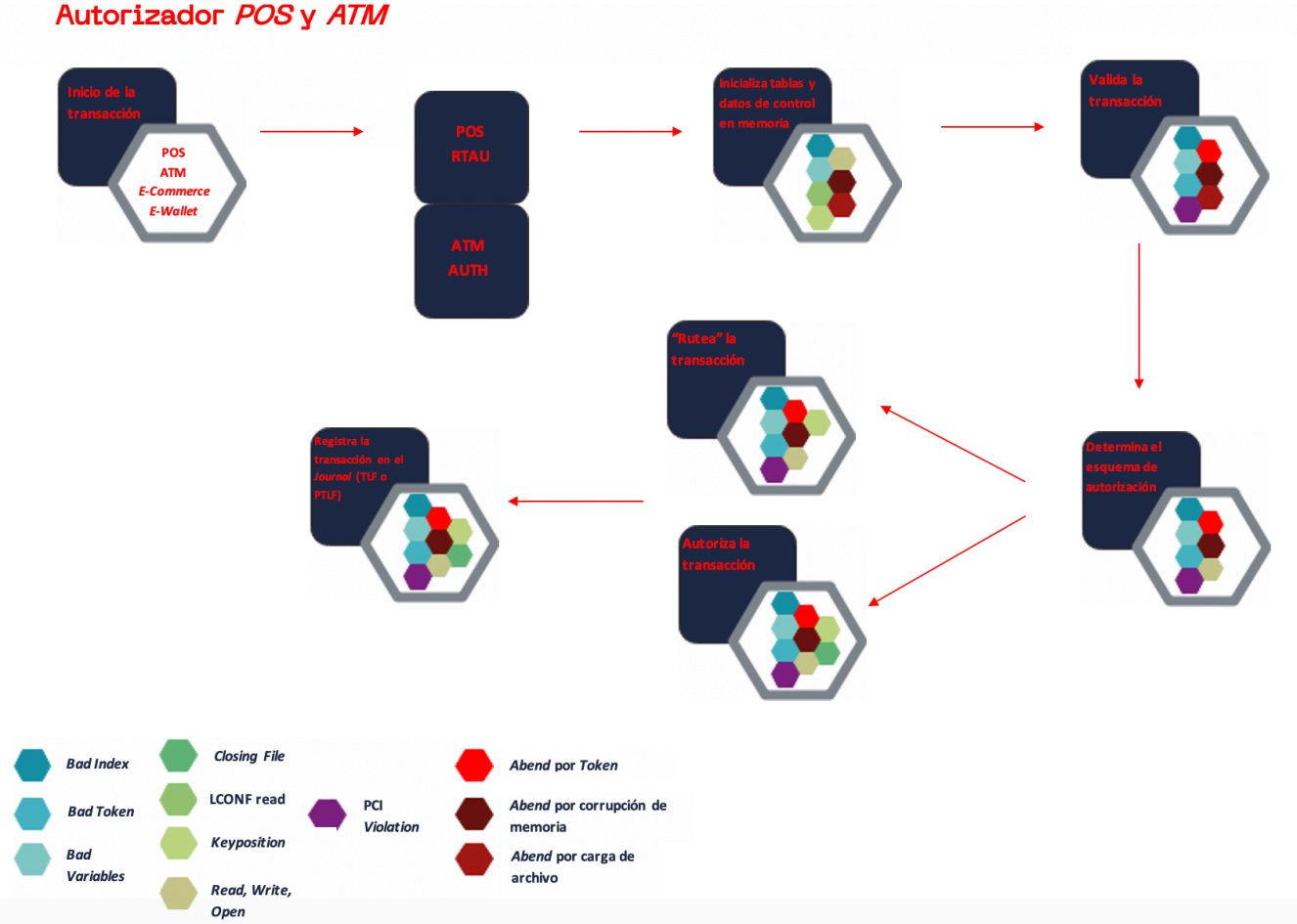
- Se recomienda que se elimine el *Abend* para que no se deje de ejecutar o cierre el programa.

Ahora que ya se han descrito las diferentes familias de *Bugs* que validamos durante la revisión, es muy importante ubicar su afectación dentro de los distintos flujos transaccionales, como se muestra en las siguientes secciones.

# Autorizador *POS* y *ATM*

Durante cualquier fase del proceso de autorización se pueden presentar varias de las fallas descritas en el siguiente diagrama, ya sea desde que se inicializa el proceso autorizador, hasta que se responde una transacción al *Device Handler*. Dentro de los autorizadores es donde el resultado de una falla puede ser más crítico y donde se requiere mayor atención al código implementado por terceras partes.

A continuación, se muestra un gráfico donde se describe el proceso del RTAU y AUTH de manera general y se brinda un breve contexto del tipo de afectación que puede haber dependiendo de la etapa.



## Inicializa tablas y datos de control en memoria

- Se pueden cargar o inicializar datos incorrectos en las tablas.
- Puede ocasionar procesamientos incorrectos de las transacciones basado en datos incorrectos.

## Valida la transacción

- Se producirá un rechazo inicial de la transacción, probablemente sin conocer mucho del motivo del rechazo o de la falla.
- Puede caerse el proceso autorizador desde el momento en que se inicializan archivos o se cargan datos en la memoria.

## Determina el esquema de autorización

- Puede afectar el método por el que se va procesar la transacción, pudiendo ocasionar que se pierda la transacción o que no se sepa a quién se debe de "rutear", hasta autorizaciones no debidas.
- Se pueden presentar caídas en el proceso de autorización.

## "Rutea" la transacción

- En caso de Nivel 1 y 2 de autorización puede enviarse información incorrecta al *Host* o la falla. Incluso puede provocar que se rechace la transacción o que se caiga el proceso de autorización sin que el *Host* sepa lo que sucedió.

## Autoriza la transacción basado en cuenta, institución, tarjeta y datos del *Track*

- En niveles 2 y 3 de autorización se puede ver comprometida la información de la persona tarjetahabiente al momento de manejar los datos sensibles que son expuestos en el EMS cuando se realizan las validaciones correspondientes. Se pueden provocar autorizaciones indebidas.

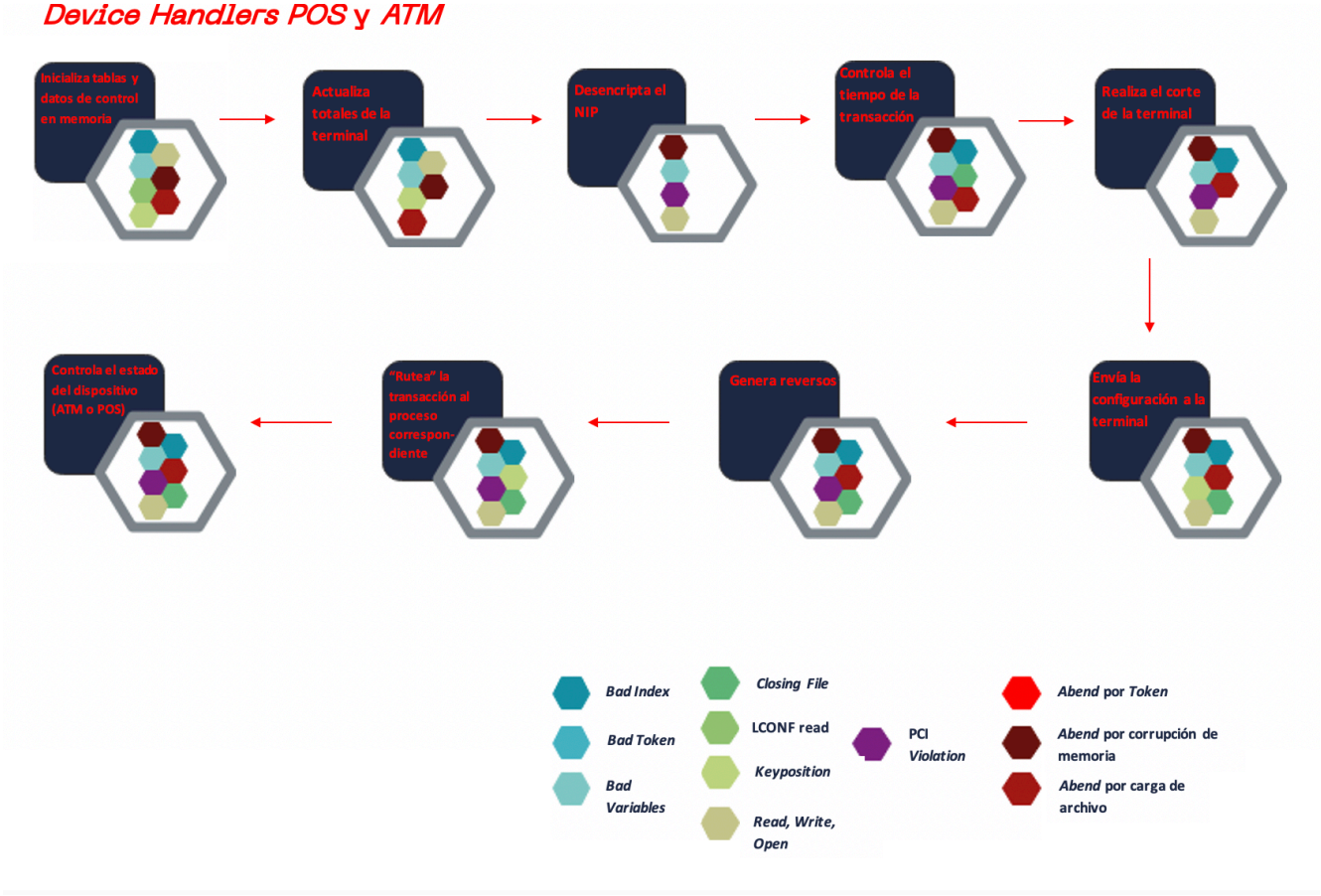
## Registra la transacción en el *Journal* (TLF o PTLF)

- Puede grabarse información incorrecta de los *Tokens* que se debe grabar en el *Journal*.
- Se puede grabar información que no sea consistente con el resultado final de la transacción.
- Se pueden corromper o bloquear los *Journals*.
- Pueden existir afectaciones en la compensación.

## *Device Handlers*

Los *Device Handlers* son la interfaz primaria entre el dispositivo (POS o ATM) y BASE24. Su principal función es realizar la conversión de los formatos de mensaje entre el dispositivo y BASE24. Dentro de estos procesos se pueden presentar también varias fallas de las descritas en el siguiente diagrama durante cualquier fase de su funcionalidad. Desde que recibe el mensaje en el dispositivo, hasta que envía una respuesta de regreso. En estos procesos hay que considerar que uno de los principales puntos de alta criticidad es que se puedan llegar a corromper los archivos de configuración de las terminales (ATM -> ATD o POS -> PTD), provocando serios problemas en la operación del negocio.

## *Device Handlers POS y ATM*



A continuación, se profundiza sobre las posibles consecuencias de que se produzca una falla en estos procesos.

## Inicializa tablas y datos de control en memoria

- Se pueden cargar o inicializar datos incorrectos en las tablas.
- Puede ocasionar procesamientos incorrectos de las transacciones, basados en datos incorrectos.

## Actualiza totales de la terminal

- Pueden ser actualizados valores incorrectos en los contadores detallados del dispositivo, afectando la contabilidad y posibles limitaciones establecidas en los servicios (límites de máximos o mínimos).

## Desencripta el NIP

- Al realizar cálculos incorrectos de datos cifrados se pueden rechazar transacciones por NIP incorrecto, ocasionado un mal servicio a la clientela.

## Controla el tiempo de la transacción

- Si se pierden estos valores por algún tipo de error, se puede perder el control de los reversos, afectando a la clientela  y dando una mala imagen.

## Realiza el corte de la terminal

- Si falla el corte o se realiza de forma incorrecta, ocasiona problemas a las transacciones en la fecha contable, lo que implica rechazos y mal servicio a la clientela.

## Controla el estado del dispositivo

- Al no tener bien controlado el estatus de cada elemento de los dispositivos (elementos de *Hardware* y *Software*), ofrece servicios que no finalizan en forma correcta por no tener el elemento con el estado correcto.

## “Rutea” la transacción al proceso correspondiente

- Puede ocasionar que se envíen transacciones a procesos incorrectos o no existentes, lo que genera un mal servicio en el dispositivo e inconsistencia de las transacciones.

## Genera reversos

- Debido al control con el dispositivo y sus elementos, si no se opera en forma correcta, impacta de manera muy importante un determinado tipo de transacciones que afectan directamente la cuenta de la o el cliente.
- Insatisfacción de la clientela.

## Envía la configuración a la terminal

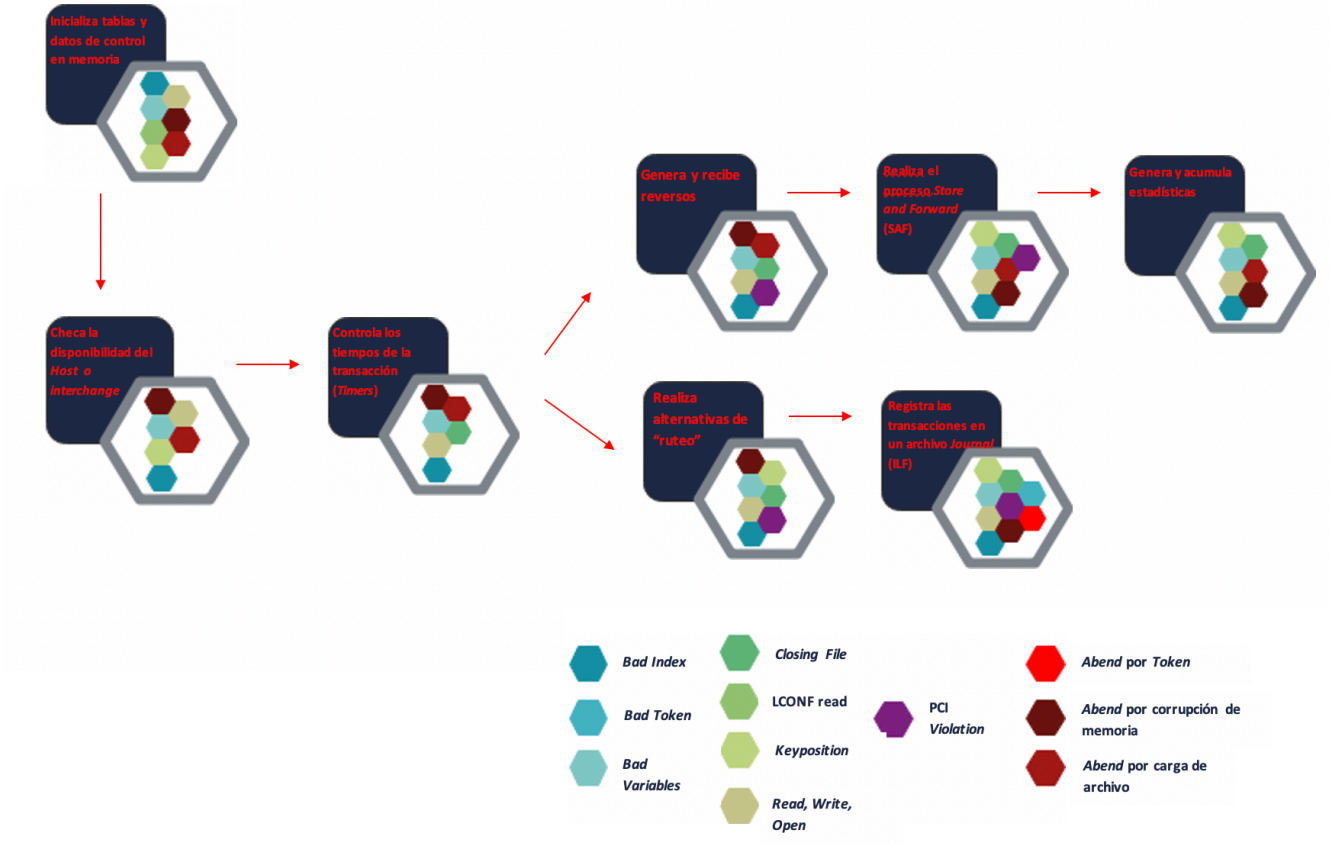
- Si no se tiene un correcto intercambio de control con el dispositivo, puede estar operando de forma incorrecta y procesando transacciones de forma errónea.

## *Host e Interchange Interface*

El Host Interface (ANSI o ISO) se encarga de controlar el tráfico de los mensajes para y desde un Host, así como realiza la conversión de los formatos de mensaje (interno y externo) entre los Hosts y BASE24.

El Interchange Interface se encarga de controlar el tráfico de los mensajes para y desde un Interchange, así como realizar la conversión de los formatos de mensaje (interno e Interchange) entre los Interchanges y BASE24. Dentro de estos procesos se pueden presentar también varias de las fallas descritas en el siguiente diagrama durante cualquier fase de su funcionalidad. Desde que recibe el mensaje de BASE24, hasta que contesta el mensaje de regreso. En este caso, la afectación más crítica producida por una vulnerabilidad en el proceso es que se puedan generar respuestas erróneas que se traduzcan en autorizaciones o rechazos inesperados.

## *Host e Interchange interface*



A continuación, se profundiza sobre las consecuencias de que se produzca una falla en estos procesos.

## Inicializa tablas y datos de control en memoria

- Se pueden cargar o inicializar datos incorrectos en las tablas.
- Puede ocasionar procesamientos incorrectos de las transacciones, basados en datos incorrectos.

## Checa disponibilidad del *Host* o *Interchange*

- Al no tener bien este control se pueden perder transacciones o dar respuestas incorrectas al servicio, en lugar de tomar posibles esquemas alternos de autorización.

## Controla los tiempos de la transacción (*Timers*)

- Puede ocasionar reversos no requeridos al perder el control de los tiempos limitados de las transacciones o respuestas como un mal servicio.

## Genera y recibe reversos

- Debido al control con el *Host*, si no se opera en forma correcta, genera un gran impacto para un determinado tipo de transacciones que afectan directamente la cuenta de la o el cliente.
- Insatisfacción de la clientela.

## Realiza el proceso *Store and Forward* (SAF)

- Este aspecto es para poder tener un equilibrio con el *Host*. Al fallar este control se tendrá pérdida de datos para lograr balances de información con el *Host*.

## Genera y acumula estadísticas

- Se afecta la información del comportamiento con el *Host* que permite realizar el monitoreo de que todo se procesa de manera eficiente con base en lo establecido.

## Realiza alternativas de “ruteo”

- Se puede estar aplicando procesamiento incorrecto de posibles alternativas de solución de las transacciones, dando un mal servicio.

## Registra las transacciones en un archivo *Journal* (ILF)

- Si no hay un registro correcto en los *Journal*, impacta directamente en la compensación con el *Host*.

## *From Host Maintenance*

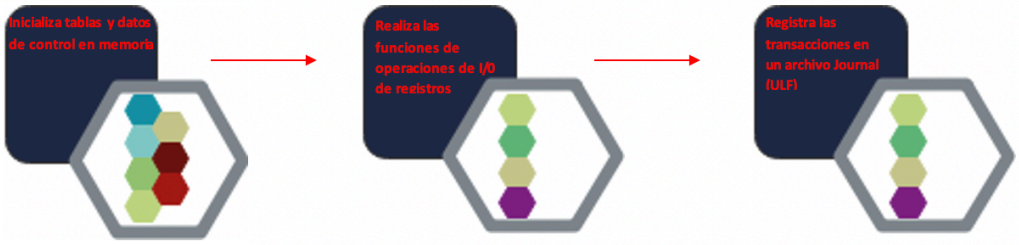
El From Host Maintenance (FHM) se encarga de habilitar a los usuarios para hacer actualizaciones en línea para los archivos de autorización BASE24 basado en archivos o tablas del Host.

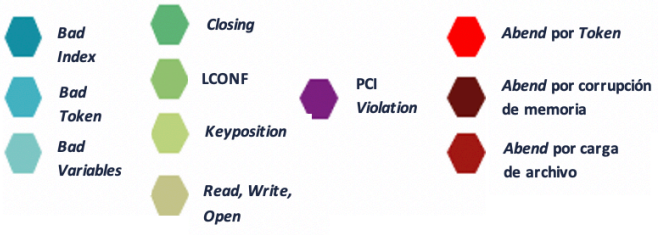
Las operaciones de actualización que se realizan con los archivos de la base de datos de BASE24 son: consulta, alta, baja y modificación de registros. Estas operaciones con archivos se realizan a través de mensajería tipo ISO con el Host y con el apoyo del módulo Host Interface.

Los archivos principales de la base de datos de BASE24 que operan con este esquema son los archivos de autorización: CAF (CardHolder File) o archivo de Tarjetas, PBF (Positive Balance File) o archivo de saldos de las cuentas y NEG (Negative File) o archivo de lista negra o boletín.

Dentro de estos procesos se pueden presentar también varias de las fallas descritas anteriormente durante cualquier fase de su funcionalidad. Desde que recibe el mensaje con la solicitud de mantenimiento de la base de datos en BASE24, hasta que se confirma con el mensaje de regreso. En este caso, la afectación más crítica producida por una vulnerabilidad en el proceso, es que se puedan generar inconsistencias en la base de datos de BASE24 al realizar operaciones incorrectas de actualizaciones.

## *From Host Maintenance*



A continuación, se profundiza sobre las consecuencias de que se produzca una falla en estos procesos.

## Inicializa tablas y datos de control en memoria

- Se pueden cargar o inicializar datos incorrectos en las tablas.
- Puede ocasionar procesamientos incorrectos de las actualizaciones de la base de datos basados en datos incorrectos e impactando a los procesos de autorización de las transacciones financieras.

## Realiza las funciones de operaciones de I/O de registros

- Se pueden estar realizando actualizaciones incorrectas de consulta, altas, bajas y actualizaciones; ocasionando corrupción de la base de datos y dando pérdida de integridad.

## Registra las transacciones en un archivo *Journal* (ULF)

- Si no hay un registro correcto en los *Journals*, impacta directamente en la integridad y control de cambios en la base de datos.

# Mejores prácticas para manejo de memoria

## Direccionamiento Directo

Aunque el direccionamiento directo está limitado a la cantidad de memoria que puede referenciar, es más eficiente y seguro que el direccionamiento indirecto. Por lo tanto, se debe de utilizar el direccionamiento directo de memoria siempre que sea posible.

Por ejemplo, supongamos que una rutina espera un parámetro de referencia que se utiliza en varios cálculos dentro de la rutina antes de regresar el resultado a quien le hizo la llamada. Una buena práctica sería mover el parámetro que está direccionado de manera indirecta a una variable en memoria local y después utilizar esa copia para realizar los cálculos. De esta manera se evita el realizar operaciones constantes con la memoria, reduciendo el riesgo de corrupción (*Bad Variable* o *Abend* por corrupción de memoria). Esto permite regresar el resultado final, ya validado al parámetro original. Aunque puede ocurrir un poco de *Overhead* en el proceso de copiar de un parámetro a una variable local y viceversa, el manejo es más seguro debido a que las referencias que utiliza esta variable ocupan direccionamiento directo.

## Direccionamiento Indirecto

Este considera arreglos, estructuras y apuntadores. La ventaja del uso de direccionamiento indirecto, en el caso de arreglos y estructuras, es que el compilador genera un apuntador, reserva el espacio necesario e inicializa el apuntador al comienzo de la estructura o del arreglo. Esto elimina la necesidad de manipulación de grandes cantidades de datos dentro de la memoria local; sin embargo, se debe de manejar con cuidado la manipulación de la información de arreglos o estructuras indirectas, ya que en caso de error o corrupción de la información, esto saldrá fuera del alcance de la rutina pudiendo afectar en otras partes del programa y provocando errores de disponibilidad (*Abend* por corrupción de memoria). Para utilizar apuntadores declarados, es necesario inicializar y manejar el espacio que utilizaran los datos al cual el apuntador estará apuntando.

## Direccionamiento Indirecto Extendido

Este tipo de direccionamiento se utiliza cuando el programa requiere un almacenamiento de información adicional al que ya tiene disponible en el segmento de datos de usuario. Con respecto al manejo de memoria para este tipo, el compilador se encarga de alojar y desalojar automáticamente el tamaño necesario en el segmento extendido. Por lo tanto, las consideraciones a nivel seguridad y manejo de la memoria para este tipo de direccionamiento son las mismas que las del direccionamiento indirecto. Asimismo, las operaciones que se realizan en el segmento extendido no son tan rápidas como aquellas que se utilizan en el segmento de personas usuarias.

Este tipo de direccionamiento se debe de usar solo cuando sea absolutamente necesario.

# Mejores prácticas para el manejo de archivos

La siguiente información pretende dar un contexto general del uso de los archivos por el sistema y programas, así como brindar consideraciones respecto al uso de memoria por parte de los mismos y posibles consecuencias resultantes de que se encuentre un *Bad File* dentro del algún programa.

El número máximo de archivos que pueden ser abiertos en un momento determinado depende del espacio disponible para los bloques de control:

- Bloques de control de acceso (ACBs)
- Bloques de control de archivos (FCBs)
- Bloques de control de apertura (OCBs)

El espacio disponible para los bloques de control está determinado, primeramente, por la memoria física del sistema. El máximo de espacio para los ACBs está determinado por el tamaño del Segmento de Procesamiento de Archivos (PFS). El tamaño del PFS disponible es de 32 MB en H-, J- y L-series.

Tomando en cuenta todo lo anterior, debemos analizar los posibles efectos que pueden tener dentro del código algún *Bug Bad File*. Por ejemplo, si se encuentran varios *Opens* dentro de un mismo archivo se puede ocasionar un *Deadlock*. Esto significa que el archivo puede quedar bloqueado usando el número de archivo asociado al siguiente *Open*. Inclusive, si llegan a ser tres o más *Opens* y ejecuta una operación de escritura en el archivo como un *Write* o *Update*, se podría estar asociando la operación con cualquiera de los otros *Opens*, lo que provoca que alguno de los archivos quede bloqueado, impidiendo el acceso al resto de archivos por medio de otros *File Numbers* y afectando la disponibilidad del proceso en cuestión.

De ahí la importancia de validar siempre el estatus de las operaciones que se ejecutan dentro de los archivos.

Otro aspecto que hay que considerar es que múltiples *Opens* en uno o diferentes archivos sin un *Close* pueden afectar otros procesos externos que hagan peticiones a estos procesos, como sucede en el caso de las herramientas de replicación o de "tokenización" de datos sensibles en los *Journals*, impidiendo que pueda realizar su tarea o provocando *Time-outs* o caídas de los procesos.

# ¿Quiénes somos?

## Metabase Q

Metabase Q es una empresa que brinda servicios administrados de ciberseguridad, enfocada en proteger a las organizaciones y empresas en América Latina de los ciberataques. Ofrecemos soluciones y servicios de ciberseguridad diseñados a la medida , con el objetivo de proteger a empresas de diversas industrias y tamaños. Nuestras capacidades de extremo a extremo incluyen nuestro servicios de consultoría, *CISO-as-a-Service*, nuestro servicios de innovación e integración tecnológica y servicios de seguridad ofensiva.

## Ocelot

**Ocelot, de Metabase Q**, es el equipo de seguridad ofensiva líder en América Latina. Hemos reunido al mejor talento del mundo en materia de simulaciones avanzadas de amenazas persistentes (APT, por sus siglas en inglés), sistemas de pago PoS y NFC, BASE24, cajeros automáticos, IoT e ICS para garantizar que podamos servirle como aliados en sus problemas más complicados. La inteligencia de amenazas, la investigación y las habilidades ofensivas de Ocelot impulsan las soluciones administradas de ciberseguridad de Metabase Q.

## Join us, join the revolution

contact@metabaseq.com
+52 55 2211 0920