GeneXus y OWASP TOP 10

Guía de desarrollo GeneXus

TABLA DE CONTENIDO

Sección : I. Introducción

Objetivo

El presente documento tiene como objetivo identificar los aspectos de seguridad a tener en cuenta desarrollando con GeneXus, para cumplir con el OWASP Top 10.

Tomando en cuenta cada uno de los ítems del OWASP TOP 10, recopila las funcionalidades realizadas automáticamente por GeneXus, los aspectos a tener en cuenta en el desarrollo y las consideraciones de despliegue del sistema.

Autores

Este documento es mantenido y actualizado por el Grupo de Seguridad de GeneXus Consulting (Pablo Alzuri, Guillermo Skrilec y Gerardo Canedo).

¡Comentarios, mejoras y precisiones son bienvenidos! Ponemos a disposición el mail seguridad@genexusconsulting.com como medio de comunicación.

Versiones

Versión	Fecha	Editor	Actualización
1.0	13/11/2013	Gerardo Canedo	Release Inicial.
1.1	15/11/2013	Guillermo Skrilec	Ajustes formato y redacción.

Sección : II. GeneXus y OWASP Top 10

A1-Injection

Referencia

https://www.owasp.org/index.php/Top_10_2013-A1

Descripción

Las fallas debido a inyecciones suceden cuando una aplicación envía datos no confiables a un intérprete. Este tipo de fallas abundan en existencia, en particular en aplicaciones legadas.

Usualmente se encuentran en intérpretes como SQL, LDAP, XPath, comandos del sistema operativo, XML, listas delimitadas, etc.

Inyección de SQL

Una vulnerabilidad de inyección de SQL consiste en la inserción de una consulta SQL a través de una entrada de datos desde el cliente de una aplicación. Un exploit exitoso de una inyección SQL puede producir la lectura de datos sensibles, la modificación de datos (Insert, Update, Delete), la ejecución de operaciones administrativas en la base de datos, la obtención de archivos existentes en sistema que aloja al DMBS y en algunos casos, la ejecución de comandos del sistema operativo.

La forma de realizar estos ataques es por medio de una entrada de datos del sistema. Se ingresan caracteres que un atacante puede modificar una sentencia SQL del propio sistema o ejecutar una propia.

Modelado de Amenazas

La inyección de SQL permite a los atacantes falsificar su identidad, modificar datos, generar situaciones de repudio en las cuales se pueden invalidar transacciones o modificar balances, permitir la divulgación de información privilegiada, destruir información o la toma del servidor de base de datos.

Acciones realizadas por GeneXus

GeneXus genera todas las consultas a la base de datos utilizando consultas parametrizadas. Esta es una de las formas recomendadas por OWASP para evitar las inyecciones SQL.

En el caso del generador RUBY, dadas las restricciones actuales de la tecnología con MySQL, además de utilizar consultas parametrizadas, se realiza la codificación de los parámetros.

Acciones a realizar por los desarrolladores

Comando SQL

Si se utiliza el comando SQL, y en dicho comando se utiliza una entrada de datos, se podría estar frente a una vulnerabilidad de Inyección SQL.

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner (código 103).

Mitigaciones

Permitir a GeneXus generar la Consulta SQL

En algunos casos, es posible reemplazar la sentencia SQL y permitir a GeneXus generarla. En ese caso, se recomienda utilizar el DBRet para generar los DataView necesarios para incorporar dichas tablas al modelo GeneXus.

Suplantar el comando SQL por un stored procedure

De ser posible, se recomienda quitar la sentencia SQL y en su lugar crear un Stored Procedure. Este Stored Procedure es utilizado desde GeneXus por medio de un External Object (del tipo Stored Procedure). Utilizando este mecanismo garantiza el uso de parámetros tipados.

Se debe notar que este mecanismo no es viable en caso de sentencias de estructura variable.

Sanitizar las Entrada

En caso de tratarse de una sentencia de estructura muy variable, se debe tratar a toda entrada como amenaza. Como mitigación, se deben sanitizar cada entrada utilizando rutinas de escape de comandos de control (OWASP ESAPI por ejemplo). Existen versiones para Java, .Net Framework y Ruby). Es importante notar que no se deben implementar rutinas propias de sanitización.

Uso de External Objects

Dependiendo de su cometido, los external objects poseen implementaciones variables. Se deben estudiar caso a caso para determinar la existencia de vulnerabilidades y su implementación.

Acciones a realizar en el despliegue de la aplicación

El usuario de conexión a la base de datos configurado en la parametrización de GeneXus, debe poseer la menor cantidad de privilegios posible. Dicho usuario no debe poseer la capacidad de ejecutar sentencias DDL sobre el esquema, no debe de poder leer esquemas fuera de los necesarios. A su vez, no se debe de utilizar super usuarios como sa, dba, admin, o el correspondiente para el DBMS utilizado.

Inyección de XML

Una vulnerabilidad de inyección de XML consiste en la inserción de documentos o elementos XML a través de una entrada de datos desde el cliente de una aplicación. Un exploit exitoso de una inyección XML puede producir la modificación de datos sensibles o el acceso a la información de forma indebida.

La forma de realizar estos ataques es por medio de una entrada de datos del sistema, la cual es incorporada en un documento XML sin realizar su codificación.

Modelado de Amenazas

La inyección de XML puede permitir a un atacante escalar en sus privilegios, modificar datos y en algún caso particular, visualizar información.

Acciones realizadas por GeneXus

Utilizando los SDTs, al realizar la operación *ToXML*, GeneXus codifica el contenido de todos sus campos.

En cuanto a la lectura y escritura de archivos XML de forma manual, los tipos de datos *XMLWriter* y *XMLReader* realizan la codificación y decodificación de los valores de los elementos y atributos utilizados.

Acciones a realizar por los desarrolladores

Conversión de XML a SDTs

Al concatenar strings para luego cargar un SDT utilizando la primitiva *FromXML*, se deben sanitizar las entradas de usuario que se concatenan. Esto es debido a que un atacante puede inyectar un elemento XML válido y sobrescribir el valor del elemento original.

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner (códigos 113 y 122). De forma manual, se debe detectar si el string a ser cargado por medio de la primitiva *FromXML* posee alguna entrada de información del usuario.

Mitigaciones

Transformar meta caracteres a su representación de dato.

Las ocurrencias de caracteres como >, <, &, etc. que pueden ser interpretados como meta caracteres de XML deben ser codificados. Para ello se puede utilizar el método *System.Security.SecurityElement.Escape* incluido en .Net Framework o el método *org.owasp.esapi.Encoder.encodeForXML* de la biblioteca ESAPI en Java.

XML Writer

Si se utiliza el método *WriteRawText* del tipo de dato *XMLWriter*, las entradas ingresadas por usuarios pueden contener meta caracteres de XML que deben ser sanitizados.

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner. De forma manual, se debe detectar si las entradas de datos que se utilizan en el método *WriteRawText* son ingresadas por un usuario y si fueron codificadas para generar el XML.

Mitigaciones

Transformar meta caracteres a su representación de dato.

Las ocurrencias de caracteres como >,<,&, etc. que pueden ser interpretados como meta caracteres de XML deben ser codificados. Para ello se pueden utilizar el método *System.Security.SecurityElement.Escape* incluido en .Net Framework o el método *org.owasp.esapi.Encoder.encodeForXML* de la biblioteca ESAPI en Java.

XML Rearder

Al utilizar el tipo de dato *XMLReader*, el XML a ser procesado puede contener elementos repetidos ingresados por medio de una inyección de XML. Generalmente se toma en cuenta el valor del segundo elemento (siendo este generalmente el inyectado).

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner.

Mitigaciones

Se debe utilizar la propiedad *ValidationType* y especificar el formato del XML. Esto genera que se valide el documento XML contra la estructura que debe tener. En caso de existir elementos duplicados (no válidos), no se procesa el XML evitando así la inyección.

Inyección de JSON

Una vulnerabilidad de inyección de JSON consiste en la inserción caracteres de control de JSON a través de una entrada de datos desde el cliente de una aplicación. Un exploit exitoso de una inyección JSON puede producir la modificación de datos sensibles o el acceso a información de forma indebida.

La forma de realizar estos ataques es por medio de una entrada de datos del sistema, la cual es incorporada en un JSON sin realizar su codificación.

Modelado de Amenazas

La inyección de JSON puede permitir a un atacante visualizar, modificar y eliminar datos.

Acciones realizadas por GeneXus

Utilizando los SDTs, al realizar la operación *ToJson*, GeneXus codifica el contenido de todos sus campos.

Acciones a realizar por los desarrolladores

Conversión de JSON a SDTs

Al concatenar strings para luego cargar un SDT utilizando la primitiva *FromJSON*, se deben sanitizar las entradas de usuario que se concatenan. Esto es debido a que un atacante puede inyectar un elemento JSON válido y sobrescribir el valor del elemento original.

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner. De forma manual, se debe detectar si el string a ser cargado por medio de la primitiva *FromJSON* posee alguna entrada de usuario.

Mitigaciones

Transformar meta caracteres a su representación de dato.

Las ocurrencias de caracteres cómo “, \, etc. que pueden ser interpretados como meta caracteres de JSON deben ser codificados. Para ello se pueden utilizar el método *System.Web.Script.Serialization.JavaScriptSerializer.Serialize* incluido en .Net Framework o el método *org.owasp.esapi.Encoder.encodeForJavascript* de la biblioteca ESAPI en Java.

Inyección de LDAP

Una vulnerabilidad de inyección LDAP es un ataque utilizado para atacar aplicaciones web que utilizan sentencias LDAP generadas a partir de entradas de datos. Un exploit exitoso de una inyección LDAP puede producir una escalada de privilegios, la modificación y visualización de información almacenada en el directorio LDAP. En caso de ser un directorio compartido por varias aplicaciones, una vulnerabilidad de un sistema puede repercutir en otros.

Modelado de Amenazas

La inyección LDAP permite a los atacantes falsificar su identidad, modificar datos, generar situaciones de repudio en las cuales se pueden invalidar transacciones, permitir la divulgación de información privilegiada, destruir información o el acceso a otros sistemas que comparten el mismo directorio LDAP.

Acciones realizadas por GeneXus

Al obtener los atributos de un ítem de LDAP, GeneXus sanitiza los caracteres utilizados en los campos User y Password, no permitiendo operadores lógicos ni paréntesis.

Al utilizar el método *GetAttribute*, sus atributos se encuentran pre-estructurados. Esto inhibe la posibilidad de ejecutar inyecciones. De la misma forma, no permite realizar consultas LDAP complejas.

Acciones a realizar por los desarrolladores

El desarrollador no debe realizar acciones adicionales.

Acciones a realizar en el despliegue de la aplicación

Se recomienda realizar la conexión al servidor LDAP sobre un canal seguro. En caso contrario, las credenciales de LDAP pueden ser visualizadas en su transporte.

Inyección de Comandos del sistema Operativo

Una vulnerabilidad de inyección de comandos de sistema operativo ocurre cuando un atacante logra ejecutar comandos del sistema operativo anfitrión de la aplicación web de forma discrecional. Se produce cuando se utilizan comandos de sistema operativo a partir de entradas de datos. Un exploit exitoso de una inyección de comandos puede producir una escalada de privilegios o la toma del servidor en cuestión por parte del atacante.

Modelado de Amenazas

La inyección de comandos del sistema operativo permite al atacante manipular la aplicación, controlar el servidor anfitrión y posiblemente afectar la información que gestiona el sistema.

Acciones realizadas por GeneXus

GeneXus no realiza acciones contra la inyección de comandos del sistema operativo.

Acciones a realizar por los desarrolladores

Validación del formato de los parámetros de llamado al comando Shell

Al utilizar la función Shell con entradas de datos ingresadas por usuarios, se pueden ejecutar comandos arbitrarios. Esto se puede lograr por medio del uso de separadores de comando como “;” en UNIX o “&” en Windows.

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner. De forma manual, se debe detectar si el string a utilizar en el comando Shell posee alguna entrada de usuario.

Mitigaciones

Validar tipos de datos del comando y de sus parámetros

Por medio de expresiones regulares y de tipos de datos (Numeric, Boolean, File, Folder, etc.) validar los parámetros que formarán parte de la invocación al sistema operativo

Codificar caracteres especiales

Ciertos caracteres especiales como “, \, /, & o ; deben ser codificados para que no sean interpretados como metadatos, sino como texto. Finalmente, a cada parámetro se lo puede enmarcar con comillas (“).

Acciones a realizar en el despliegue de la aplicación

Se recomienda ejecutar el servidor de aplicaciones con un usuario con mínimos privilegios.

Inyección de SMTP

Una vulnerabilidad de inyección de SMTP ocurre cuando un atacante logra modificar los comandos STMP enviados desde el servidor web al servidor de correo de forma arbitraria. Un exploit exitoso de permite al atacante enviar mails de forma discrecional.

Modelado de Amenazas

Una inyección de SMTP permite realizar phishing, propagar spam, generando situaciones de repudio sobre los emails enviados por el sistema.

Acciones realizadas por GeneXus

GeneXus brinda los tipos de datos para el envío de mails vía SMTP.

Acciones a realizar por los desarrolladores

El desarrollador no debe realizar acciones adicionales.

Acciones a realizar en el despliegue de la aplicación

Se recomienda realizar un hardening adecuado del servidor de correo y una revisión periódica de los logs generados por el mismo

Inyección de Código Fuente

Una vulnerabilidad de inyección de código ocurre cuando un atacante logra ejecutar código arbitrario (interpretado o compilado) utilizando para ello una entrada del sistema.

Modelado de Amenazas

La inyección de código permite al atacante manipular la aplicación, controlar el servidor anfitrión y afectar la información que gestiona el sistema. También puede permitir la obtención de usuarios y contraseñas de la infraestructura utilizada.

Acciones realizadas por GeneXus

El código nativo generado por GeneXus contempla la validación de la entrada en la carga dinámica de clases.

Acciones a realizar por los desarrolladores

Validar los tipos de archivo subidos por medio de Blobs

Utilizando las funcionalidades de Blobs, se podrían subir distintos tipos de archivos, en particular aquellos cuyo formato es interpretado por el servidor de aplicaciones.

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner. De forma manual, se debe analizar la existencia de reglas que validen contra una lista aceptada de extensiones los archivos subidos al servidor.

Mitigaciones

Validar extensiones aceptadas en los blobs

Utilizando listas de extensiones de archivos, aceptar o rechazar un archivo cargado por medio de un blob.

Es preferible utilizar listas positivas (listas blancas) a listas negativas (listas negras). Las listas blancas permiten conocer lo autorizado, mientras que las listas negras reflejan lo no autorizado en un momento dado, pudiendo crearse una nueva extensión peligrosa en el futuro.

Validar el parámetro que indica el programa en las llamadas dinámicas

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner. De forma manual, se debe analizar si la variable que representa al programa a ser llamado proviene de una entrada de datos de un usuario.

Mitigaciones

Utilizar llamadas dinámicas sólo si es necesario

De ser posible, es recomendable realizar llamadas indicando los programas efectivos a ser invocados.

Validar el programa llamado dinámicamente

Se debe verificar si el programa llamado dinámicamente puede ser indicado por un usuario por medio de una entrada de datos. En dicho caso, se debe validar que dicho programa es válido y no ingresado por el usuario.

Acciones a realizar en el despliegue de la aplicación

Configurar el servidor de aplicaciones para no ejecutar código u objetos cargados dinámicamente.

Inyección de Logs

La inyección en registros (logs) ocurre cuando un atacante logra escribir (o eliminar) en un registro del sistema líneas arbitrarias que no permiten realizar una traza de los eventos que efectivamente sucedieron en el sistema.

Modelado de Amenazas

La inyección en los registros pueden afectar a los sistemas de monitoreo de eventos, así como enmascarar eventos de seguridad ocurridos en el sistema.

Acciones realizadas por GeneXus

GeneXus no brinda funcionalidades de log de forma incorporada. Una forma de realizarlo, es utilizando una tabla de la base de datos para almacenar log. En este caso, no es posible generar una inyección de registros o eliminar registros ya ingresados.

En Java, se puede utilizar el log por defecto del servidor de aplicaciones por medio del modificador *status* del comando *msg*.

Acciones a realizar por los desarrolladores

Caracteres de fin de línea, espacios en blanco excesivos, inyección de HTML y caracteres de separación

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner. De forma manual, se debe analizar si las variables a enviar al log que son producto de entradas de usuario son sanitizadas.

Mitigaciones

Codificar las entradas de usuario

Considerando las particularidades del log a ser utilizado, se deben codificar las entradas de usuario para que no interfieran con su formato. Distintos aspectos a codificar son : fin de línea, cantidad de espacios en blanco excesivos que parecen una nueva línea, código HTML, caracteres de separación (|, ; u otros).

Inyección de Fechas y Horas

En el caso que en el formato de la línea de log se utilice un timestamp, se debe considerar la posibilidad que la entrada ingresada por el usuario posea una fecha.

Mitigaciones

Utilizar un Log auto-numerado

Se deben numerar correlativamente las líneas. De esa forma, las líneas inyectadas serán detectadas por la falta de correlación de números y no serán confundidas.

Acciones a realizar en el despliegue de la aplicación

Configurar los permisos en los archivos de log de forma que el servidor de aplicaciones no pueda eliminar registros ya creados.

A2-Broken Authentication and Session Management

Referencia

https://www.owasp.org/index.php/Top_10_2013-A2

Descripción

Usualmente se desarrollan mecanismos de autenticación y/o manejo de sesión propios, pero desarrollar los mismos correctamente es muy difícil. Como resultado dichos mecanismos tienen defectos en áreas tales como: salida del sistema, manejo de contraseñas, tiempos de espera, recuérdame, preguntas secretas, actualización de la cuenta, etc. Encontrar dichos defectos suele ser difícil pues cada implementación es única.

Autenticación

Es el proceso en el cual se verifica que una persona o entidad es quien dice ser. Usualmente es implementada enviando un nombre de usuario (o identificador) y uno o más ítems que solo el usuario debería conocer.

Modelado de Amenazas

Defectos en el mecanismo de autenticación permite a alguien atacar a algunas o incluso todas las cuentas, cuando el atacante tuvo éxito puede hacer todo lo que la víctima puede hacer. Las cuentas privilegiadas en los sistemas son usualmente blanco de este tipo de ataques.

Acciones realizadas por GeneXus

GeneXus X Evolution 2 agrega el módulo GAM (GeneXus Access Manager). GAM es el módulo de seguridad incluido con GeneXus, el mismo implementa autenticación y autorización basado en roles (RBAC).

GAM soporta los siguientes esquemas de autenticación:

- Local – Las credenciales de usuario se almacenan en el repositorio de datos GAM.
- Externa – En este caso se consume un servicio web SOAP o un programa externo. Dicho programa define si el usuario autentica o no, el mismo servicio retorna los roles que tiene asociados el usuario.
- Facebook – La autenticación se realiza en el sitio de Facebook.
- Twitter – La autenticación se realiza en el sitio de Twitter.
- Google – La autenticación se realiza en el sitio de Google.

Acciones a realizar por los desarrolladores

Implementar modulo propio siguiendo todas las mejores prácticas de la industria en materia de seguridad para el desarrollo del mismo (por ejemplo no almacenar las contraseñas sino un hash salteado de las mismas).

Formas de detección

Para detectar esta vulnerabilidad se debería hacer una revisión del código del módulo de autenticación. Para apoyar en la detección de problemas se puede utilizar GeneXus Security Scanner, no obstante es necesaria la revisión manual de código. La revisión manual del código

es necesaria pues problemas lógicos (o inclusive de requerimientos) pueden generar vulnerabilidades explotables en dicho modulo.

Problemas comunes en los mecanismos de autenticación

Fortaleza de las contraseñas

Una contraseña se considera fuerte cuando es difícil de adivinar por medio de una atacante. La evaluación de la fortaleza de una contraseña debe tener en cuenta su largo y complejidad.

Acciones realizadas por GeneXus

GAM (GeneXus Access Manager) implementa políticas de seguridad en las cuales podemos configurar:

- Tiempo máximo entre cambio de contraseñas (días)
- Tiempo mínimo de espera entre cambio de contraseñas (minutos)
- Largo mínimo de las contraseñas
- Mínima cantidad de caracteres numéricos en la contraseña
- Mínima cantidad de caracteres mayúscula en la contraseña
- Mínima cantidad de caracteres especiales en la contraseña
- Número máximo de contraseñas en la historia

Acciones a realizar por los desarrolladores

Configurar las propiedades de políticas de fortaleza de contraseñas de la empresa.

Para esta tarea se tienen los siguientes casos:

1. Se almacenan las contraseñas en la base de datos del GAM, entonces las políticas de fortaleza de contraseñas deben ser definidas en el GAM.
2. Se almacenan las contraseñas en un gestor externo (ej.: LDAP), entonces las políticas de fortaleza de contraseñas deben ser definidas en el gestor externo.
3. Se almacenan las contraseñas en un módulo propio, entonces se debe implementar en el módulo los controles para las políticas de fortaleza de contraseñas.

Mecanismo de recuperación de contraseña inseguro

Si la funcionalidad de recuperación de contraseñas está mal implementada puede comprometer las credenciales de uno o hasta todos los usuarios.

Acciones realizadas por GeneXus

- GAM (GeneXus Access Manager) brinda una implementación de referencia de la funcionalidad de recuperación de contraseñas.

Acciones a realizar por los desarrolladores

- Si se utiliza GAM, se debe revisar la implementación de referencia de la funcionalidad de recuperación de contraseñas. El objetivo de esta revisión es adaptar dicha funcionalidad a las políticas de recuperación de contraseñas de la empresa.
- Si no se utiliza GAM, se debe implementar la funcionalidad de recuperación de contraseñas siguiendo las políticas de recuperación de contraseñas de la empresa. Se sugiere tomar como guía el documento OWASP - Forgot Password Cheat Sheet.

Autenticación multifactor

Autenticación de múltiples factores es cuando se usa más de un factor de autenticación para iniciar sesión o procesar una transacción:

- Algo que sabes (ej.: datos de la cuenta o contraseñas)
- Algo que tiene (ej.: tarjeta de coordenadas)
- Algo que está (ej.: huellas digitales)

La autenticación multifactor es requerido en contextos donde necesitamos niveles de seguridad altos (ej.: transferencia bancaria).

Acciones realizadas por GeneXus

GeneXus no implementa funcionalidades de autenticación multifactor.

Acciones a realizar por los desarrolladores

- Implementar en un módulo propio autenticación multifactor.

Mensajes de error

Los errores reportados en el proceso de autenticación deben ser genéricos, esto es para evitar la posibilidad de enumeración e usuarios. En particular debemos tener cuidado de retornar errores genéricos en las funcionalidades de login y recordar contraseña.

Acciones realizadas por GeneXus

- GAM (GeneXus Access Manager) brinda una implementación de referencia de las funcionalidades de login y recuperación de contraseñas. En dichas funcionalidades se retorna un mensaje genérico. A modo de ejemplo ante un error en el login retorna: “El usuario o la contraseña no es correcta. (GAM18)”.

Acciones a realizar por los desarrolladores

- Devolver mensajes genéricos en las funcionalidades que puedan brindar a un atacante información sobre la existencia o no de un determinado usuario en el sistema.

Uso de HTTPS

La página de inicio de sesión y todas las páginas autenticadas posteriores se debe acceder en exclusiva a través de HTTPS estricto.

Acciones realizadas por GeneXus

GeneXus no presenta restricciones para el uso de HTTPS estricto, su configuración es únicamente una propiedad del modelo GeneXus.

Acciones a realizar por los desarrolladores

No utilizar llamadas con el protocolo HTTP fijado a fuego, esto puede evitar la compatibilidad con HTTPS.

Acciones a realizar en el despliegue de la aplicación

- Desplegar la aplicación mediante HTTPS estricto.
- Verificar el certificado utilizado para HTTPS.
- Verificar la configuración HTTPS del servidor web.

Manejo de sesión

Es el proceso en el cual el servidor mantiene el estado de una entidad con la cual esta interactuando. Esto es necesario para que el servidor sepa cómo reaccionar frente a los siguientes pedidos. Las sesiones son mantenidas en el servidor utilizando un identificador de sesión, el identificador de sesión es pasado entre el cliente y el servidor en cada pedido (y respuesta). Los identificadores de sesión deben ser únicos por usuario y difíciles de predecir.

Modelado de Amenazas

Este tipo de defectos permite a alguien atacar a algunas o incluso todas las cuentas, cuando el atacante tuvo éxito puede hacer todo lo que la víctima puede hacer. Las cuentas privilegiadas en los sistemas son usualmente blanco de este tipo de ataques.

Acciones realizadas por GeneXus

- GeneXus utiliza el manejo de sesión web del lenguaje generado. Esto es una fortaleza, pues el manejo de sesión web de los lenguajes está altamente probado y existen recomendaciones claras para la configuración del mismo.
- Implementa seguridad en los pedidos AJAX, lo cual se configura con la propiedad Ajax Requests Security property. Esto permite cifrar los parámetros de los pedidos AJAX, cuando esta propiedad está configurada en “High”, todos los parámetros son cifrados usando una clave asociada con la sesión web.
- Implementa seguridad en los llamados a objetos GeneXus expuestos por URL, lo cual se configura con la propiedad Encrypt parameters. Esto permite cifrar los parámetros de los pedidos HTTP/HTTPS, cuando esta propiedad está configurada en “Session Key”, todos los parámetros son cifrados usando una clave asociada con la sesión web.

Acciones a realizar por los desarrolladores

- Verificar que todos los parámetros están cifrados (Security Scanner #100).
- En GeneXus X Evolution 1 verificar que la propiedad Ajax Request Security esta en High (Security Scanner #106).
- En GeneXus X Evolution 2 verificar que la propiedad Javascript Debug Mode Property está en No (Security Scanner #106).
- Establecer un tiempo de espera adecuado para la aplicación y complementarlo con un manejo adecuado de la propiedad On Session Timeout property.

Formas de detección

Para detectar el uso no recomendado de las configuraciones de GeneXus se puede utilizar GeneXus Security Scanner. No obstante se debe hacer un análisis manual del manejo de sesión para asegurar que el mismo no es vulnerable a ataques.

Problemas comunes en la implementación el manejo de sesión

No requerir re-autenticación para funciones sensibles

Para mitigar CSRF y el secuestro de sesión, es importante exigir las credenciales del usuario antes de actualizar la información de cuenta confidencial (ej.: contraseña, correo electrónico, etc.), o antes de las operaciones sensibles, como una transferencia electrónica de dinero.

Acciones realizadas por GeneXus

GeneXus no implementa funcionalidades de re-autenticación.

Acciones a realizar por los desarrolladores

Implementar la re-ejecución de algún mecanismo de autenticación antes de efectuar operaciones sensibles.

Fortaleza de los identificadores de sesión

Los identificadores de sesión deben cumplir ser aleatorios y no predecibles.

Acciones realizadas por GeneXus

Utiliza los identificadores de sesión propios de cada lenguaje, con esto garantiza no tener problemas con la entropía y predictibilidad.

Acciones realizadas por los desarrolladores

No es necesario realizar ninguna acción.

Acciones a realizar en el despliegue de la aplicación

- Se recomienda no utilizar el nombre por defecto de la cookie utilizada para manejo del ID de sesión, los mismos suelen ser muy descriptivos y brindan a un atacante información innecesaria (ej.: JSESSIONID (J2EE), ASP.NET_SessionId (ASP .NET), etc.).
- Los identificadores de sesión deben ser suficientemente largos, por lo menos deben tener un largo de 128 bits (16 bytes).

Evitar exposición de identificadores de sesión

Una forma habitual de secuestro de sesión, es que un atacante malicioso intercepte el identificador de sesión de un usuario autenticado.

Acciones a realizar en el despliegue de la aplicación

- De ser posible utilizar cookies no persistentes para intercambio del identificador de sesión, la cookie de sesión debe tener las propiedades:
    - SECURE – Indica a los navegadores que deben enviar esta cookie solamente a través de conexiones seguras (HTTPS).
    - HTTP-ONLY – Indica a los navegadores que esta cookie no debe ser accesible por scripts (ej.: JavaScript o VBScript).
    - DOMINIO – El dominio de una cookie, le indica al navegador que la misma es solamente enviada a dicho dominio o subdominios.
    - PATH – El path de una cookie, le indica al navegador que la misma es solamente enviada a dicha aplicación dentro del dominio o subdominios.
- Si no es posible utilizar cookies para el intercambio de identificadores de sesión, se debe hacer un estudio exhaustivo de cómo se debe configurar el mecanismo elegido.
- Deshabilitar en el servidor web los mecanismos de intercambios de identificadores de sesión no admitidos para la aplicación.

Fijación del identificador de sesión

Una forma habitual de secuestro de sesión, es tratar de fijar el identificador de sesión de nuestra víctima.

Acciones realizadas por GeneXus

GeneXus no implementa funcionalidades contra la fijación de sesión.

Acciones a realizar por los desarrolladores

En el proceso de login, antes de comenzarlo, hacer un *destroy* de la sesión web para asegurarse que le estamos ofreciendo un nuevo identificador de sesión al usuario.

Una visión más genérica del mismo control es, cambiar el identificador de la sesión después de cualquier cambio del nivel de privilegios.

Expiración de la sesión

Un error habitual, el cual facilita ataques sobre el manejo de sesiones es olvidar la expiración manual o automática de la misma.

Acciones realizadas por GeneXus

GeneXus caduca la sesión que contiene la clave se autenticación AJAX en un tiempo parametrizable.

Acciones a realizar por los desarrolladores

En el proceso de logout destruir la sesión web.

Acciones a realizar en el despliegue de la aplicación

Configurar la expiración automática de la sesión web, esto se puede configurar por tiempo inactivo o tiempo absoluto desde su creación.

A3-Cross-Site Scripting (XSS)

Referencia

https://www.owasp.org/index.php/Top_10_2013-A3

Descripción

Los ataques de Cross Site Scripting (XSS) son un tipo de inyección, que por su relevancia son estudiados de forma independiente. En ellos, el atacante logra utilizar una aplicación web para enviar código malicioso (generalmente en la forma de un script) al navegador del usuario final.

Las vulnerabilidades que permiten este tipo de ataque son relativamente frecuentes y pueden ocurrir en cualquier parte que la aplicación utilice una entrada de un usuario para generar una salida, sin su correspondiente validación o codificación.

Existen distintas tipos de Cross Site Scripting (almacenado, reflejado o basado en DOM), siendo todos ellos igual de peligrosos.

Modelado de Amenazas

Mediante el XSS, un atacante puede causar una variedad de problemas a los usuarios finales (desde mensajes molestos a comprometer sus actividades en el sistema). En los casos más severos, el atacante puede apropiarse de la identidad del usuario. Otros riesgos potenciales son la instalación de software troyano, la divulgación de información almacenada en el equipo del usuario, el reenvío del usuario a otro sitio web o la modificación de la información presentada en el sitio.

Acciones realizadas por GeneXus

Validación de las entradas en tipo dato y largo.

GeneXus realiza de forma automática la validación de los distintos tipos de datos de los atributos o variables que son ingresados al sistema. En el caso de tipos de datos que posean un dominio de valores predefinido, un largo máximo o un formato específico, se realizan dichas validaciones antes de poder ser utilizados en la lógica de negocio. Esto elimina la capacidad de ingresar valores por fuera del dominio o rango especificado; evitando errores y caracteres inválidos.

También se brinda un mecanismo por el cual se permite especificar una expresión regular contra la cual se validarán todas las entradas de datos especificadas sobre el campo o variable en cuestión.

Codificación del contenido incluido en las páginas basado en el contexto (HTML, como valor de atributo, JavaScript, JSON, CSS, URL, etc.)

GeneXus codifica automáticamente todos los datos de usuarios que GeneXus incluye en las páginas web que presenta a sus clientes. Para realizar esta codificación, toma en cuenta el contexto sobre el cual se utiliza; ya sea como parte del cuerpo de un documento HTML, como un atributo de un elemento de HTML, un valor en JavaScript, una respuesta JSON a un servicio REST, parte de una URL o una regla en un estilo CSS.

Es importante notar que el desarrollador puede optar por deshabilitar la codificación de entradas. En este caso, el desarrollador debe de codificar la salida apropiadamente.

Acciones a realizar por los desarrolladores

Validación de formato en textos enriquecidos

Algunos campos deben aceptar entradas en las cuales se presentan datos y metadatos. El ingreso de HTML en un editor es un ejemplo. Se ingresa el contenido y su formato. Dependiendo del contenido que se desea utilizar, los metadatos a permitir.

En el caso de optar por utilizar un subconjunto de los metadatos disponibles, se recomienda un enfoque de lista blanca (enumerar los elementos aceptados) contra un enfoque de lista negra (enumerar los elementos prohibidos). Esto permite evitar los vectores de amenazas conocidos actualmente así como los que se descubran posteriormente.

Formas de detección

Esta vulnerabilidad puede ser detectada observando el código o los requerimientos del sistema. Se debe verificar que en las entradas de texto enriquecido se encuentran exclusivamente metadatos válidos.

Mitigaciones

Sanitizar el valor ingresado.

Dado el valor de un campo con formato enriquecido, se inspecciona el mismo en búsqueda de metadatos que son válidos para el lenguaje pero no se desean utilizar. En caso de encontrarse estos metadatos, se convierten en texto simple.

Retornar un error al detectar una entrada inválida.

Dado el valor de un campo con formato enriquecido, se inspecciona el mismo en búsqueda de metadatos que son válidos para el lenguaje pero no se desean utilizar. En caso de encontrarse estos metadatos, se retorna un error indicando el metadato no soportado.

Codificación de código escrito por el desarrollador

GeneXus brinda la posibilidad al desarrollador de escribir código HTML propio. Si bien existen otras opciones (como la utilización de User Controls), ésta alternativa en ampliamente utilizada. En este caso, también es responsabilidad del desarrollador codificar las entradas de datos de usuario de forma que éstas no interfieran con su código HTML.

Cabe notar que la validación de formato no es exclusiva de HTML. Existen casos en los cuales se utilizan lenguajes de marcas personalizados (envíos de mails personalizados por ejemplo). En estos casos, también se deben codificar los indicadores de metadatos.

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner.

De forma manual, se debe identificar los siguientes ítems:

- Textblocks con formato HTML

- Variables con formato HTML
- Atributos con formato HTML
- Valor de la propiedad Default HTML Format
- Uso del tipo de datos HTTPResponse
- Form.HeaderRawHTML
- Form.JScriptSrc property

Mitigaciones

Evaluar si el código HTML o de scripting es realmente necesario

No siempre la escritura de código HTML por parte del desarrollador es estrictamente necesaria. Se puede haber originado en una versión anterior de GeneXus o en una decisión de un desarrollador donde no fue tomado en cuenta el aspecto de seguridad. Se plantea como primera mitigación evaluar si efectivamente el código HTML personalizado es necesario, o el mismo puede ser suplantado por programación GeneXus.

Reemplazar el código HTML por un User Control

En caso de ser posible, se plantea utilizar un User Control (con su consiguiente prueba de seguridad) en vez de utilizar código HTML directamente en los objetos GeneXus. Aparte de las facilidades de reúso de los User Controls, también definen una separación entre datos y comportamiento.

Codificar la salida

En caso de escribirse dinámicamente HTML, JavaScript, CSS u otro lenguaje interpretado por un navegador web, se deben seguir las reglas de programación de OWASP para la prevención de XSS:
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

En caso de otros lenguajes, como puede ser un lenguaje de plantilla de emails (con metadatos identificados con el símbolo %), se deben codificar las ocurrencias de los caracteres especiales antes de ser procesados.

Acciones a realizar en el despliegue de la aplicación

Uso de políticas de seguridad de contenidos

En un enfoque de seguridad en profundidad, se propone configurar una política de seguridad de contenido (Content Security Policy) para indicarles a los distintos navegadores las funcionalidades y dominios con los cuales la aplicación puede utilizar.

Configuración de cookies de Sesión como HTTPOnly

Se recomienda configurar el marcador de *HTTPOnly* a aquellas cookies que no deben ser accedidas desde código JavaScript (como la cookie identificadora de la sesión del usuario). Esto permite, en caso de un exploit XSS exitoso, mitigar el posible impacto del mismo.

A4-Insecure Direct Object References

Referencia

https://www.owasp.org/index.php/Top_10_2013-Insecure_Direct_Object_References

Descripción

Este problema se puede dar cuando tenemos usuarios en el sistema que tienen acceso parcial al mismo. Dicho usuario puede variar parámetros en la URL para intentar acceder a objetos para los cuales no tiene permiso.

Las aplicaciones usualmente utilizan el nombre o clave de algún objeto para generar las páginas web. Cuando dichas aplicaciones no verifican que el usuario este autorizado para acceder al objeto, el resultado es un defecto de referencia insegura al objeto. Se puede probar fácilmente manipulando los parámetros para verificar que la autorización fue implementada correctamente.

Objetos con interfaz web

Los objetos con interfaz web generados por GeneXus son:

- Web Panels (se consideran los Prompts un caso particular de los Web Panels)
- Transacciones
- Procedimiento Main con protocolo de llamado HTTP
- Web componente con URL Access en true
- Reportes

En los mismos debemos tener cuidado que la variación en los parámetros de los mismos, no permita a un usuario acceder a datos para los cuales no tiene permiso.

Modelado de Amenazas

Todos los datos que son referenciados de forma insegura pueden estar comprometidos, el atacante modificando parámetros en la URL pude acceder a los mismos. El impacto que puede tener esta vulnerabilidad en nuestro negocio, es el impacto de que los datos comprometidos sean revelados, más el impacto en la imagen pública del conocimiento de la vulnerabilidad.

Acciones realizadas por GeneXus

- GeneXus posibilita el cifrado de los parámetros enviados en la URL, eso se configura con la propiedad Encrypt parameters. La misma se define a nivel del modelo GeneXus y sobrescrita por cada objeto.
- GAM (GeneXus Access Manager) verifica en forma automática la autorización al acceder a los diferentes tipos de objetos con interfaz web.

Acciones a realizar por los desarrolladores

- Verificar que los parámetros estén cifrados en todos los objetos, para esta verificación se puede utilizar el Security Scanner (Security Scanner #100).
  En los objetos que no se tienen parámetros cifrados, se debe verificar la necesidad de no cifrar y que se tengan contramedidas adecuadas para evitar esta vulnerabilidad.
- Si se utiliza GAM, se debe verificar que todos los objetos verifique autorización.

- Si no se utiliza GAM, se debe implementar correctamente la autorización de los objetos.

Acciones a realizar en el despliegue de la aplicación

- Cambiar clave de cifrado de los parámetros.

Formas de detección

- Ejecución del GeneXus Security Scanner para verificar el cifrado de los parámetros.
- Ejecución de la herramienta de testing automatizado URLChecker (la misma es capaz de ejecutar un pedido valido, con la configuración de un determinado usuario, esto nos sirve para detectar problemas en el mecanismo de autenticación y autorización).

Llamadas AJAX

En las llamadas AJAX se debe tener cuidado que la variación en los parámetros de los mismos, no permita a un usuario acceder a datos para los cuales no tiene permiso.

Modelado de Amenazas

Todos los datos incluidos en suggest y combos dinámicos pueden ser pedidos por medio de llamadas AJAX en GeneXus. Si no se toman contemplaciones al respecto, los mismos pueden ser accedidos por medio de un atacante.

Acciones realizadas por GeneXus

- Si se está en GeneXus X Evolution 1, se tiene la propiedad *Ajax Request Security*, la misma permite manejar el cifrado de los parámetros en los pedidos AJAX. El valor recomendado para esta propiedad es “High”, dicho valor es el valor por defecto de la propiedad. Cuando esta propiedad está configurada en “High”, todos los parámetros se cifran mediante una clave asociada a la web session. Cuando la misma expira o si no es válida, se mostrará un mensaje de error indica que el usuario actualice la página para obtener una nueva sesión.
- Si se está en GeneXus X Evolution 2, se tiene la propiedad *Javascript Debug Mode*, la misma permite generar todos los JS y CSS estándar comprimidos. Cuando esta propiedad está definida en “No”, se ejecutan las llamadas AJAX de manera segura.

Acciones a realizar por los desarrolladores

- Si el desarrollador tiene llamadas AJAX propias o utiliza algún User Control que tenga llamadas AJAX, es responsabilidad del mismo asegurar las mismas.

Formas de detección

- Ejecución del GeneXus Security Scanner para la propiedad *Ajax Request Security* o *Javascript Debug Mode*.

Generación insegura de archivos temporales

Los archivos temporales que queden disponibles dentro de la aplicación web, pueden potencialmente ser accedidos por medio de un atacante cambiando los parámetros en la URL.

Modelado de Amenazas

Todos los archivos que son generados dentro de la aplicación web pueden ser referenciados de forma insegura y pueden estar comprometidos. El atacante modificando los parámetros en la URL pude acceder a los mismos. El impacto que puede tener esta vulnerabilidad en nuestro negocio, es el impacto de que los datos comprometidos sean revelados, más el impacto en la imagen pública del conocimiento de la vulnerabilidad.

Acciones realizadas por GeneXus

- GeneXus separa los contenidos públicos y privados, para esto debemos configurar cuidadosamente las propiedades *PrivateTempStorage* y *PublicTempStorage*; para profundizar en este tema ver A5.

Acciones a realizar por los desarrolladores

- No se deben generar archivos con información sensible dentro de la aplicación web, si se generan archivos con información sensible, los mismos deben ser generados fuera de la aplicación web y se debe brindar acceso al usuario por medio de un procedimiento GeneXus. Dicho procedimiento GeneXus debe hacer el control de acceso antes de retornar al usuario el archivo. Es importante resaltar que dicho procedimiento no puede recibir el path del documento a devolver, debe recibir un identificador que permita inferir el path del mismo en función de datos almacenados en el servidor (ej. tabla que mapeo: one time password y path), esto es para evitar la vulnerabilidad conocida como path traversal.

Formas de detección

Para detectar generación de archivos por intermedio de GeneXus dentro de la aplicación web, se puede utilizar GeneXus Security Scanner. No obstante, se debe hacer un análisis manual para detectar todos los casos de generación de archivos dentro de la aplicación web.

A5-Security Misconfiguration

Referencia

https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration

Descripción

Parte de la seguridad requiere una configuración segura definida e implementada para la aplicación, frameworks, servidores, bases de datos y plataformas. Todas estas configuraciones se deben definir, implementar y mantener, dado que la mayoría de los componentes utilizan valores predeterminados que no son seguros. Esto también incluye mantener todo el software actualizado.

Modelado de Amenazas

Los errores de configuración pueden comprometer el sistema, los datos y cualquier otro recurso. Utilizando credenciales por defecto o de prueba, se pueden modificar datos y generar situaciones de repudio, las cuales pueden invalidar transacciones y hacer perder confianza en la veracidad de la información del sistema.

Acciones realizadas por GeneXus

GeneXus no realiza acciones sobre la configuración del ambiente de producción

Acciones a realizar por los desarrolladores

Propiedades de la KB GeneXus

Algunas propiedades de la base de conocimiento GeneXus pueden deshabilitar controles en el código generado.

Formas de detección

Se deben inspeccionar manualmente el valor de las propiedades en la KB que generan los programas de test y producción

Mitigaciones

Establecer los siguientes valores para las propiedades:

- *Javascript Debug Mode*: No

Acciones a realizar en el despliegue de la aplicación

Mantener actualizado el software de base (OS, Web/App Server, DBMS)

Es común que con el paso del tiempo se identifiquen vulnerabilidades en los distintos componentes del stack tecnológico en el que se soporta la aplicación. Vulnerabilidades en estos componentes de software ponen en peligro la aplicación.

Formas de detección

Se deben comparar las versiones de los distintos componentes con las últimas liberaciones por parte de los proveedores. A su vez, se deben instalar las actualizaciones y parches de seguridad liberados.

Mitigaciones

Implementar y ejecutar políticas de actualización de software, determinando las versiones seguras y los planes de actualización de los distintos componentes de software.

Instalar en los servidores de producción únicamente las funcionalidades necesarias.

Tener presente software no necesario en los servidores puede introducir vulnerabilidades de éstos productos en los sistemas.

Formas de detección

Inventariar el software instalado en los distintos servidores y comparar dicho inventario con el software necesario para la ejecución.

Mitigaciones

Se recomienda desinstalar o eliminar todo aquel servicio o software presente en los servidores que no sea necesario.

Configurar permisos correctamente sobre los directorios de la aplicación web.

Es recomendable establecer permisos mínimos y suficientes sobre el sistema de archivos de cada servidor que forma parte del sistema.

Formas de detección

Se deben validar los permisos de los distintos directorios que componen la aplicación, el sistema de archivos del directorio y verificar que no existen permisos otorgados que no sean necesarios.

Mitigaciones

Otorgar los permisos necesarios. En particular las aplicaciones GeneXus necesitan autorización para escribir y leer en las carpetas configuradas en el client.cfg o web.config:

- *PrivateTempStorage*
- *PublicTempStorage*

Configurar correctamente usuario de base de datos

Usuarios de datos con permisos excesivos de ejecución (como crear, modificar o eliminar tablas) pueden ser utilizados para causar daños en caso del compromiso de un servidor.

Formas de detección

Se deben validar que los permisos del usuario de base de datos configurado en GeneXus sean los mínimos necesarios y no incluyan operaciones de manipulación de esquemas.

Mitigaciones

Crear un usuario de base de datos para cada web application que forme parte de la aplicación (web app accesible desde internet, web app interna, etc.) dándole a cada usuario los permisos de creación, modificación, eliminación y visualización de tuplas en las tablas específicas que utiliza.

Realizar hardening de los servidores y del framework utilizado

Se denomina hardening al proceso por el cual utilizando un procedimiento escrito, se configura un software para mitigar vulnerabilidades conocidas y configuraciones potencialmente inseguras.

Cambio de claves de cifrado por defecto

Para el cifrado de los parámetros de la URL y contraseñas en la configuración, GeneXus utiliza una clave de cifrado simétrica. De no especificarse lo contrario, se utiliza un valor por defecto. Este valor por defecto es conocido dado que se encuentra en las rutinas generales del generador a utilizar (Java, .Net, Ruby, etc.).

Se deben especificar un par de claves (de sitio y de sesión) con valores generados de forma aleatoria.

Formas de detección

Se debe constatar la presencia del archivo Application.key en la aplicación web

Mitigaciones

Generar Claves de cifrado aleatorias para cada ambiente

Crear un archivo llamado Application.key (en el directorio virtual en el caso de .Net y en el directorio WEB-INF en el caso de Java) el que contenga dos líneas de 32 valores sexagesimales generados de forma aleatoria.

Propiedades del despliegue de GeneXus

Se deben configurar los directorios temporales de manejo de blobs para asegurar el principio de mínimo privilegio.

Formas de detección

Se debe inspeccionar en el archivo client.cfg (Java) o web.config (.Net) el valor de las propiedades TMPMEDIA_DIR y CS_BLOB_PATH. En caso de indicarse el valor "PrivateTempStorage" para la propiedad TMPMEDIA_DIR; debe ser modificada.

Mitigaciones

Especificar los siguientes valores para las propiedades

- TMPMEDIA_DIR: Directorio temporal en el cual la aplicación debe poder leer y escribir. Es de uso interno y no es necesario que sea accedido vía HTTP. A modo de ejemplo se puede utilizar el valor absoluto "/tmp/webapp".
- CS_BLOB_PATH: Directorio temporal en el cual la aplicación debe poder leer y escribir. Se utiliza para servir archivos temporales al cliente web. Es accedido vía HTTP.
- Para el resto del sitio web, no es necesario indicar permiso de escritura.

No poseer usuarios de prueba en producción

Es posible que usuarios y configuraciones utilizadas en ambiente de prueba, puedan encontrarse presentes en producción.

Formas de detección

Se deben probar las credenciales genéricas utilizadas en los ambientes de prueba en el ambiente de producción y validar que no sea posible un inicio de sesión exitoso.

Mitigaciones

Eliminar las credenciales de ambientes como Test, Capacitación, Desarrollo, etc.

A6-Sensitive Data Exposure

Referencia

https://www.owasp.org/index.php/Top_10_2013-Sensitive_Data_Exposure

Descripción

Dependiendo del sistema y de las regulaciones de las distintas industrias y gobiernos, cierta información (personal, financiera, de la organización) puede ser considerada sensible, siendo necesario resguardarla, generalmente utilizando mecanismos criptográficos. Cuando se falla en mantener su reserva, se está frente a una situación de exposición de datos sensibles.

Generalmente existen tres ubicaciones en donde se puede exponer esta información: en el almacenamiento, en su transmisión o al desplegarla.

Modelado de Amenazas

En caso de que un atacante obtenga información sensible, se puede ver afectada la imagen de la organización, aplicar sanciones económicas o de otra índole así como la pérdida de confianza en el sistema y en la organización.

Almacenamiento (contraseñas, números de tarjetas de crédito, logs, etc.)

Acciones realizadas por GeneXus

GeneXus no realiza acciones de cifrado en el almacenamiento de datos sensibles.

Acciones a realizar por los desarrolladores

Almacenamiento de Secretos compartidos

En el caso de la necesidad de un secreto compartido, se debe almacenar un hash del valor deseado y no el valor en sí.

Formas de detección

Esta falla de diseño se puede detectar de forma manual revisando los requerimientos.

Mitigaciones

Almacenar un hash previamente modificado (salting) del secreto compartido

Existen funciones criptográficas con ese cometido como por ejemplo SHA2 (256 o 512). A su vez, es recomendable alterar por medio de una función el valor a aplicarle la función de hash. Esta técnica se denomina *Salting*.

Para validar el secreto compartido, se debe calcular la misma función de alteración y función de hash para compararlo con el resultado almacenado.

Almacenamiento de Información sensible

En el caso de ser necesario almacenar información sensible para luego ser utilizada (contraseñas para el uso de un servicio, tarjetas de crédito u otros datos sensibles), debe ser almacenada tomando en cuenta ciertas medidas de seguridad.

Formas de detección

Esta falla de diseño se puede detectar de forma manual revisando los requerimientos.

Mitigaciones

Almacenar la información sensible cifrada

La información sensible debe de ser cifrada en capa de aplicación, antes de ser persistida. El uso de criptografía por medio de la persistencia (base de datos) no es suficiente, ya que de existir una inyección de SQL, la información sería clara para la capa de aplicación.

Cifrar la clave de cifrado

La clave de cifrado no debe de persistirse en la base de datos junto con el dato sensible. Una opción es almacenar la clave de cifrado en otro medio como puede ser un archivo de texto.

A su vez, debe cifrarse la clave de cifrado utilizada para la información sensible. Para cifrar esta clave, se puede utilizar el Site Key de GeneXus.

Sobre el algoritmo de cifrado, se recomienda utilizar el algoritmo *AES* con claves de 128 a 256 bits (http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf).

Datos sensibles en Logs

Con el fin de monitorizar, de depurar errores o de dejar registro, es común enviar información a los distintos logs de los servidores involucrados. Se debe evitar registrar en el log información sensible o secretos compartidos en texto claro.

Formas de detección

Esta falla de diseño se puede detectar de forma manual revisando los requerimientos.

Mitigaciones

Evitar el registro de datos sensibles en Logs.

De ser posible, se debe de evitar completamente el uso de los datos catalogados como sensibles como parte de la salida de los logs.

Enmascarar valores de datos sensibles almacenar en Logs.

En caso de ser completamente necesario registrar datos sensibles en el log, se debe registrar información derivada de ésta, de forma que permita identificarla entre un conjunto de valores posibles pero que no la represente unívocamente sin tener en cuenta el contexto. A modo de ejemplo, es aceptable registrar los 4 últimos dígitos de una cuenta o una tarjeta o una función de hash de un password.

Datos sensibles en archivos intermedios

Puede ser necesario generar un archivo intermedio (PDF, Excel) para enviar una respuesta a un pedido de un cliente. Si el archivo no es eliminado luego de caducar el pedido, información sensible puede ser comprometida.

Formas de detección

Esta falla de diseño se puede detectar de forma manual revisando los requerimientos.

Mitigaciones

Evitar el uso de archivos intermedios.

De ser posible, se debe de evitar completamente el uso de archivos intermedios. Es recomendable escribir la respuesta directamente a la salida (tipo de datos *HTTPResponse*).

Eliminar los archivos intermedios luego de ser utilizados.

Luego de ser enviados en una respuesta, se debe verificar que los archivos intermedios generados son eliminados del servidor.

Acciones a realizar en el despliegue de la aplicación

Con el fin de mitigar filtrado de datos sensibles, se recomienda asignar los mínimos permisos necesarios a cada proceso que forma parte del sistema. A su vez, se recomienda que el directorio *PrivateTempStoragePath*, así como el directorio temporal que se pueda utilizar para generar archivos temporales, no sean servidos a clientes ni vía HTTP ni mediante ningún otro protocolo.

Transmisión

La comunicación entre distintos componentes de software en texto claro puede ser interceptadas y leídas por personas no autorizadas.

Acciones realizadas por GeneXus

Todos los objetos generados por GeneXus que pueden ser llamados desde la web (protocolo HTTP o SOAP) pueden ser ejecutados sobre HTTPS sin ser necesario ningún cambio en su generación.

En cuanto a comunicaciones de otra índole, se puede utilizar el protocolo específico sobre un canal TCP en una conexión protegida por TLS.

Acciones a realizar por los desarrolladores

Utilizar Canales cifrados en la comunicación con clientes y otros servicios

Se recomienda fuertemente no utilizar HTTP para la comunicación entre clientes y servidor. Cualquier dato transmitido sobre HTTP puede ser visto sin esfuerzo alguno por cualquier nodo intermedio participante en la cadena de ruteo IP. De forma análoga, utilizar LDAPS en vez de LDAP. Para aquellos protocolos que no soporten una versión cifrada del mismo, se sugiere utilizar TLS (cifrado a nivel de TCP).

Es importante notar que la aplicación completa debe utilizar HTTPS, incluyendo el contenido estático. Esto es debido a que en todo pedido puede incluirse información sensible (como

cookies). La tecnología actual hace despreciable la diferencia de performance entre el uso de HTTP y HTTPS.

Formas de detección

Esta vulnerabilidad puede ser detectada en el análisis de riesgo de la arquitectura, utilizando el GeneXus Security Scanner y de forma manual comprobando que existen pedidos al servidor que se realizan por medio de HTTP.

Mitigaciones

Utilizar HTTPS

Se debe indicar en la base de conocimiento que no se especifique protocolo (ni HTTP ni HTTPS en la generación). Las invocaciones a la aplicación deben hacerse de forma exclusiva por HTTPS.

Utilizar LDAPS

De forma análoga a HTTP y HTTPS, se sugiere utilizar LADPS en vez de LDAP. Esto podría evitarse si el servidor LDAP se encuentra conectado por una red segura físicamente (ambos en el mismo datacenter y en ubicaciones conocidas). La pérdida de performance es despreciable.

Comunicación con la BD sobre TLS en caso de ser necesario

En el caso no confiar en la seguridad física de las conexiones de red entre la base de datos y los servidores de aplicaciones (por el uso de infraestructura compartida por ejemplo), se recomienda establecer conexiones sobre TLS. En el caso que el driver de conexión de base de datos y el motor soporten cifrado; este último mecanismo es preferible.

Comunicación cifrada con otros servidores

En el caso de consumir o publicar información por medio de servicios a otros sistemas, se recomienda establecer canales mutuamente autenticados (de ser posible). A su vez, en el caso de desearse controlar la autorización de acceso de forma granular, se sugiere utilizar las tecnologías de seguridad de web services (WS-Security).

Acciones a realizar en el despliegue de la aplicación

En el despliegue, se deben utilizar certificados válidos (cadena de validación aceptada por los clientes) y se deben deshabilitar los algoritmos de cifrado débiles.

Despliegue (navegador)

Todo lo que enviado al navegador (browser), aunque se envíe como oculto, puede ser visualizado por el cliente. Los datos sensibles solo deben ser enviados en los casos estrictamente necesarios, evitando su transmisión a personas no autorizadas.

Acciones realizadas por GeneXus

GeneXus no realiza acciones para evitar el envío de información sensible al navegador. De ser utilizada de forma explícita o por medio de fórmulas, es enviada.

Acciones a realizar por los desarrolladores

No agregar datos sensibles ocultos

En el caso de utilizar un panel por dos perfiles distintos, los datos sensibles que no deben ser visualizados por todas las personas que acceden a un panel no deben de protegerse ocultando la información.

Formas de detección

Esta falla de desarrollo se puede detectar de forma manual revisando el código.

Mitigaciones

Quitar la información del panel

En caso de poseer datos sensibles que no son necesarios en un panel o transacción, se recomienda eliminarlos de la misma. Si se tratase de una transacción que define estructura, se puede crear una transacción paralela que lo contenga.

Cargar la información de forma selectiva

En caso que el dato sensible se utilice en ciertos casos únicamente, se debe cargar en una variable únicamente si es necesario.

En caso de situaciones complejas, se recomienda separar el objeto en tantos como perfiles distintos lo utilicen.

Caches intermedios y del browser

En el camino de la información del servidor al usuario que la consulta, posiblemente existan caches de contenido, proxies y/u otros appliances. A su vez, los navegadores poseen caches propios en los cuales almacenan información. De no especificarse los metadatos adecuados, la información podría ser almacenada por estos componentes y ser consultada posteriormente sin los correspondientes controles de autenticación y autorización.

Acciones realizadas por GeneXus

En todas las páginas generadas por GeneXus, así como los contenidos estáticos generados automáticamente, se agregan los cabezales HTTP correspondientes para indicar qué contenido puede ser cacheado (scripts, imágenes) y qué contenido no (paneles con datos por ejemplo).

Acciones a realizar por los desarrolladores

En el caso de generar páginas o contenido de forma personalizada, utilizando el tipo de dato *HTTPResponse*, se debe indicar si dicha respuesta es pública, se puede mantener en cache y por cuanto tiempo.

Formas de detección

Esta falla de desarrollo se puede detectar de forma manual revisando el código o utilizando GeneXus Security Scanner (tipo de dato *HTTPResponse*).

Mitigaciones

Incluir Cabezales HTTP de Cache

En caso de utilizar respuestas personalizadas por medio del tipo de dato *HTTPResponse*, determinar e incluir valores para los siguientes metadatos:

- Cache-Control
- Pragma
- Expires
- Last-Modified

A7- Missing Function Level Access Control

Referencia

https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control

Descripción

Cualquiera en nuestra red puede enviar pedidos a nuestra aplicación. Decimos que tenemos un problema de falta de funciones de control de acceso, de cuando un usuario anónimo puede tener acceso a funciones privadas o un usuario sin privilegios tiene acceso a funciones privilegiadas.

Este problema se puede dar si aplicación no protege adecuadamente el acceso a las diferentes funcionalidades. Si la aplicación tiene estos controles a nivel de configuración, la misma puede estar mal configurada. Si la misma tiene estos controles a nivel de código, algún desarrollador puede haberse olvidado de incluir dicho código.

Otra situación común en el que vemos presente este tipo de problemas, es cuando hacemos correctamente el control de acceso previamente a hacer visible la interfaz de usuario, pero olvidamos re ejecutarlo cuando ejecutamos en el servidor cada funcionalidad.

El enfoque utilizado para el análisis de los problemas de falta de funciones de control de acceso, es similar al utilizado para A4, eso se debe a que tenemos los mismos puntos de entradas.

Modelado de Amenazas

Todas las funcionalidades que no tienen un correcto control de acceso pueden estar comprometidas, el atacante puede acceder directamente a las mismas. Esto puede ocasionar un alto impacto tanto en el sistema (debido a las funcionalidades comprometidas), como al negocio por la imagen pública de la empresa.

Acciones realizadas por GeneXus

- GeneXus ejecuta todas las validaciones en el servidor, cuando una validación es ejecutada en el cliente, la misma se vuelve a hacer en el servidor.
- GAM (GeneXus Access Manager) verifica en forma automática la autorización al acceder a los diferentes tipos de objetos con interfaz web.

Acciones a realizar por los desarrolladores

Los objetos con interfaz web generados por GeneXus son:

- Web Panels (se consideran los Prompts un caso particular de los Web Panels)
- Transacciones
- Procedimiento Main con protocolo de llamado HTTP
- Web componente con URL Access en true
- Reportes

Si se utiliza GAM:

- Se debe verificar que todos los objetos verifiquen autorización, esto se debe hacer a nivel de acceso al objeto, y también se debe hacer a nivel de ejecución de cada evento.
- Se debe implementar la seguridad a nivel de datos en la carga de los mismos en la interfaz gráfica.

Si no se utiliza GAM, se verificar el control de acceso a cada funcionalidad. En el evento GeneXus en cuestión, se debe verificar la autorización del usuario.

En los casos que se deba verificar el acceso a nivel de dato, debe ser verificado a nivel de evento de igual forma.

Formas de detección

- Ejecución del GeneXus Security Scanner para verificar:
  - Ejecución el procedimiento que verifica la autorización (102) en web panels y transacciones.
  - Detección de web components con URL Access = true (107).
- Ejecución de la herramienta de testing automatizado URLChecker (la misma es capaz de ejecutar un pedido valido, con la configuración de un determinado usuario, esto nos sirve para detectar problemas en el mecanismo de autenticación y autorización).

A8-Cross-Site Request Forgery (CSRF)

Referencia

https://www.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_%28CSRF%29

Descripción

CSRF es cuando un sitio malicioso fuerza pedidos válidos a un sitio vulnerable. Esto se puede ver como si alguien carga contenido en nuestro navegador y genera el pedido válido. El pedido es válido si el usuario está autenticado en la aplicación víctima. Usualmente la generación del pedido se hace utilizando XSS.

Acciones realizadas por GeneXus

POST

En cada respuesta HTTP, GeneXus incluye automáticamente un token (generado por sesión en el servidor web). Este token es incluido en cada pedido HTTP realizado al servidor. De no existir el token correcto, el pedido de descarta.

A su vez, en caso de cifrarse los parámetros de la URL, se obtiene un enfoque de seguridad en profundidad, ya que el atacante debe poder cifrar los parámetros correctamente para la llamada.

GET

Dada la forma de trabajo de GeneXus al responder pedidos web, las llamadas con GET no necesitan protección contra CSRF.

En caso de cifrarse los parámetros de la URL, se obtiene un enfoque de seguridad en profundidad, ya que el atacante debe poder cifrar los parámetros correctamente para la llamada.

Acciones a realizar por los desarrolladores

POST

El desarrollado no debe realizar ninguna acción adicional. Por medio del Security Scanner, se deben identificar y estudiar los casos de utilización del tipo de dato HTTPRequest.

GET

En el caso de ejecutar lógica de negocio al invocar con el método GET, ésta puede ser llamada utilizando CSRF.

Formas de detección

Esta falla de desarrollo se puede detectar de forma manual revisando el código.

Mitigaciones

No almacenar cambios en la base de datos en el evento Start (cambios en los datos del negocio)

De tratarse de un web panel, evaluar si es posible ejecutar la lógica de negocio en un evento o únicamente al utilizar el método POST.

En caso de ser necesaria en el evento Start, evaluar si es necesario que realice modificaciones en los datos de la aplicación. Es importante notar que es válido y necesario almacenar pistas de auditoría incluso en estos casos.

Utilizar parámetro de frescura NOnce (Number Once) para indicar que es un GET solicitado

En el caso que se debe ejecutar lógica en el evento Start que cambia los datos del sistema o cuando se trata de un Procedure Main HTTP, se debe utilizar un parámetro NOnce (Number Once) que indique que el GET es válido ya que se accedió a otra página de la aplicación.

El mecanismo es el siguiente:

1. En el Panel que se menciona al Procedure Main HTTP, se genera un número aleatorio.
2. Se incluye en Link al Procedure Main HTTP y se almacena en la web session.
3. En el Procedure Main HTTP, se obtiene el valor almacenado en la web session y se compara contra el parámetro.
4. Si coinciden, se ejecuta la lógica y se elimina de la sesión el valor utilizado; sino se ignora el pedido.

Acciones a realizar en el despliegue de la aplicación

Verificar que el servidor no responda en los cabezales HTTP con metadatos de política de contenido cruzados ("Access-Control-Allow-Origin: *" por ejemplo).

A9-Using Components with Known Vulnerabilities

Referencia

https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities

Descripción

Generalmente los distintos componentes como librerías, frameworks y módulos de software ejecutan con todos los privilegios. Si los mismos poseen vulnerabilidades que se pueden explotar, podría ocurrir pérdida de información o incluso se podría tomar el servidor.

Para las aplicaciones GeneXus se deben tener consideraciones en 3 niveles distintos:

- Componentes que usa GeneXus
- Extensibilidad GeneXus
- Software base

Modelado de Amenazas

El uso de componentes con vulnerabilidades conocidas en cualquier capa del stack tecnológico puede permitir a un atacante utilizar a discreción los servidores comprometidos, falsificando su identidad, modificando datos y generar situaciones de repudio en las cuales se pueden invalidar transacciones o modificar balances, permitir la divulgación de información privilegiada, destruir información o utilizar la infraestructura como plataforma de un ataque.

Componentes que usa GeneXus

Las aplicaciones generadas con GeneXus utilizan un conjunto de clases estándar que se encuentran públicas para su consulta:

http://wiki.gxtechnical.com/commwiki/servlet/hwiki?GeneXus+Standard+Classes,

Asimismo se utilizan distintos componentes de terceras partas en las aplicaciones generadas:

http://wiki.gxtechnical.com/commwiki/servlet/hwiki?External+utilities+used+by+GeneXus+applications,

Los componentes externos generalmente no son accesibles directamente por los usuarios finales, por lo que explotar alguna vulnerabilidad de los mismos puede ser más complicado.

Acciones realizadas por GeneXus

- GeneXus se encarga de utilizar las últimas versiones de los distintos componentes que utiliza.
- GeneXus brinda nuevas versiones con vulnerabilidades corregidas en las distintas liberaciones que realiza.

Acciones a realizar por los desarrolladores

- Verificar que los componentes de terceros utilizados estén actualizados.
- Utilizar la última versión de GeneXus disponible.

Acciones a realizar en el despliegue de la aplicación

- Se sugiere cambiar el driver que se utiliza para la conexión con la base de datos por la versión más actual del mismo. La versión del driver que provee GeneXus solamente se debería utilizar para los prototipos.

Formas de detección

Se debe investigar si las versiones de los componentes utilizados poseen vulnerabilidades conocidas.

Extensibilidad GeneXus

GeneXus brinda distintos mecanismos de extensibilidad como User Controls, Extensions, Patterns y External Objects.

Acciones realizadas por GeneXus

- GeneXus Security Scanner controla el uso de User Controls en las aplicaciones, “User Control usage #121“.

Acciones a realizar por los desarrolladores

- Controlar los componentes externos que se utilizan y mantenerlos actualizados.

Acciones a realizar en el despliegue de la aplicación

- Verificar que se utiliza la última versión disponible del componente.

Formas de detección

Esta vulnerabilidad puede ser detectada tanto observando el código cómo utilizando el GeneXus Security Scanner.

Software base

Las aplicaciones generadas con GeneXus corren sobre software base estándar (servidores de aplicaciones, bases de datos, entre otros), por lo que se debe controlar que los mismos estén actualizados y con los parches de seguridad correspondientes.

Acciones realizadas por GeneXus

- GeneXus no tiene el control sobre este punto.

Acciones a realizar por los desarrolladores

- Ninguna

Acciones a realizar en el despliegue de la aplicación

- Se debe verificar el software base sobre el cuál se despliega la aplicación. Se recomienda buscar en las bases de datos de los distintos proveedores de software de manera de mantener los mismos actualizados.
- Se sugiere la implementación de políticas como por ejemplo la utilización de software que tenga soporte.

Formas de detección

Se puede utilizar software de terceros para la detección de vulnerabilidades y de versiones con vulnerabilidades conocidas en el software base.

A10-Unvalidated Redirects and Forwards

Referencia

https://www.owasp.org/index.php/Top_10_2013-A10-Unvalidated_Redirects_and_Forwards

Descripción

Las redirecciones y reenvíos sin validar son posibles cuando una aplicación web acepta entradas sin validar que son utilizadas como destino de una redirección.

Modificando la entrada insegura para que apunte a un sitio malicioso, un atacante puede generar un ataque de phishing y obtener las credenciales de un usuario.

Modelado de Amenazas

Al utilizarse una URL de un sitio de confianza, los ataques de phishing tienden a poseer mayor efectividad. Esto es debido a que los usuarios validan la URL como perteneciente a un sitio confiable.

A su vez, este tipo de ataque también puede ser utilizado para generar una URL que podría evitar el control de acceso de la aplicación y brindarle al atacante acceso a funciones privilegiadas del sistema, que de otra manera, no le serían accesibles

Acciones realizadas por GeneXus

GeneXus genera todas las llamadas internas de forma relativa, tanto si se utilizan los métodos Call y Link como las funciones Call y Link con objetos GeneXus.

Acciones a realizar por los desarrolladores

Funciones Link y Call como Variable con primer parámetro

En el caso de utilizar las funciones Link o Call como primer parámetro una variable, se debe verificar que dicho parámetro no es ingresado por un usuario.

Formas de detección

Esta vulnerabilidad puede ser detectada utilizando el GeneXus Security Scanner.

Mitigaciones

Comprobar el valor de la variable contra una lista blanca

En los casos que se utiliza una variable como primer valor de la funciones Link o Call, se debe verificar contra una lista blanca que el valor de la variable se encuentra dentro de los posibles.

Referencias

GX Community Wiki

http://wiki.gxtechnical.com/commwiki/servlet/hwiki?OWASP+2010+Top+10+Security+Risks+in+GeneXus+Applications,

http://wiki.gxtechnical.com/commwiki/servlet/hwiki?Going+into+production%3A+checklist+for+Applications+using+GAM,

GX Wiki

http://iwiki.genexus.com/hwiki.aspx?Seguridad+en+Aplicaciones+WEB+con+GX+segun+OWASP,

http://iwiki.genexus.com/hwiki.aspx?Seguridad+WEB+Tilo

Manual Security Scanner

http://wiki.gxtechnical.com/commwiki/servlet/hwiki?Security+Scanner+extension+user+manual,

Encuentro GX22

Seguridad, atributo crítico de un sistema

http://www.genexus.com/encuentro2012/xxii-encuentro-genexus--materiales?es,0,,2805

¿Qué hace hoy GeneXus por la seguridad de nuestras Aplicaciones?

http://www.genexus.com/encuentro2012/xxii-encuentro-genexus--materiales?es,0,,2864

Cómo hacer aplicaciones seguras en GeneXus Evolution 2

http://www.genexus.com/encuentro2012/xxii-encuentro-genexus--materiales?es,0,,2865

Encuentro GX21

Desarrollando aplicaciones seguras con GeneXus

http://www.genexus.com/encuentro2011/conferencia-materiales?es,0,,2382

¿Qué hace y cómo se utiliza la Seguridad Integrada a GeneXus?

http://www.genexus.com/encuentro2011/conferencia-materiales?es,0,,2444

Cómo incorporar seguridad a mis aplicaciones GeneXus

http://www.genexus.com/encuentro2011/conferencia-materiales?es,0,,2420

¿Qué resuelve GX Evolution 2 para la seguridad de sus aplicaciones Web y SD?

http://www.genexus.com/encuentro2011/conferencia-materiales?es,0,,2419

Encuentro GX20

Comprendiendo seguridad

http://www.events.genexus.com/portal/hgxpp001.aspx?16,73,1248,O,S,0,,2010

Seguridad en las aplicaciones. ¿Qué hace GeneXus por nosotros?
http://www.events.genexus.com/portal/hgxpp001.aspx?16,73,1248,O,S,0,,2009