CRIPTOGRAFÍA Y SEGURIDAD

Los 25 errores de software más frecuentes

Integrantes:

Federico Elli 50817

Juan Martín Buireo 51061

Martín Purita 51187

28/05/2014

1. Introducción

Los 25 errores de software más frecuentes son los errores más generalizados y críticos que pueden conducir a grandes vulnerabilidades en el software. Son peligrosos porque con frecuencia permiten a los atacantes tomar el control por completo del software, robar datos o impedir que el software funcione en absoluto.

La lista Top 25 es una herramienta para:

- Ayudar a los programadores a evitar estos tipos de vulnerabilidades que afectan a la industria del software, identificando y evitando errores demasiado comunes.
- Los clientes de software pueden utilizar la misma lista para pedir el software más seguro.
- Los investigadores de seguridad de software pueden utilizar el Top 25 para centrarse en un limitado pero importante subconjunto de todos los fallos de seguridad conocidos.
- Por último, los administradores de programas y directores de IT pueden utilizar la lista de Top 25 para medir los progresos en sus esfuerzos para asegurar su software.

La lista es el resultado de la colaboración entre el Instituto SANS, MITRE y muchos de los mejores expertos en seguridad de software en los EE.UU. y Europa.

Más de 20 organizaciones aportaron para actualizar esta lista en el 2011, donde se evalúa cada debilidad basada en la prevalencia, importancia y posibilidades de ocurrencia.

Utiliza el sistema de puntuación de debilidad común (CWE) para anotar y clasificar los resultados finales.

1.1. Top 25 errores

N°	Puntuación	Identificación	Descripción
1	93.8	CWE-89	La neutralización inadecuada de elementos especiales que se utilizan en un comando SQL ('inyección SQL')
2	83.3	CWE-78	La neutralización inadecuada de elementos especiales que se utilizan en un comando operativo ("Inyección de comandos OS ')
3	79.0	CWE-120	Búfer de copia sin comprobar el tamaño de la entrada ('Classic Buffer Overflow)
4	77.7	CWE-79	La neutralización inadecuada de entrada durante la generación de la página web ("Cross-site Scripting")
5	76.9	CWE-306	Falta de autenticación en función crítica
6	76.8	CWE-862	Falta de autorización
7	75.0	CWE-798	Uso de credenciales hardcodeadas
8	75.0	CWE-311	Falta de cifrado de datos confidenciales
9	74.0	CWE-434	Subida de archivos de extensión peligrosa sin restricción

10	73.8	CWE-807	Confianza en entradas en una decisión de seguridad
11	73.1	CWE-250	Ejecución con privilegios innecesarios
12	70.1	CWE-352	Falsificación de un pedido a otro sitio
13	69.3	CWE-22	Limitación indebida de un path a un directorio restringido
14	68.5	CWE-494	Descarga de código sin verificar la integridad
15	67.8	CWE-863	Autorización incorrecta
16	66.0	CWE-829	Inclusión de funcionalidad desde una esfera de control no confiable
17	65.5	CWE-732	Asignación incorrecta de permisos para un recurso crítico
18	64.6	CWE-676	Uso de una función potencialmente peligrosa
19	64.1	CWE-327	Uso de un algoritmo criptográfico roto o riesgoso
20	62.4	CWE-131	Cálculo incorrecto del tamaño del buffer
21	61.5	CWE-307	Restricción indebida de intentos de autenticación incorrectos
22	61.1	CWE-601	Redirección URL a sitios no confiados
23	61.0	CWE-134	String de formato no controlado
24	60.3	CWE-190	Overflow de enteros o wraparound
25	59.9	CWE-759	Uso de hash sin salt

1.2. Categorías de alto nivel

Las categorías de alto nivel en las que se ha dividido a estos errores en el 2009 son:

- Interacción insegura entre componentes. Comprende los problemas: 1, 2, 4, 9, 12 y 22.
- Manejo de recursos riesgoso. Comprende los problemas: 3, 13, 14, 16, 18, 20, 23 y 24.
- Defensas porosas. Comprende los problemas: 5, 6, 7, 8, 10, 11, 15, 17, 19, 21 y 25.

1.3. Errores destacados fuera del Top 25

N°	Identificación	Descripción
[26]	CWE-770	Asignación de recursos sin límites
[27]	CWE-129	Validación incorrecta del índice de una matriz
[28]	CWE-754	Control inadecuado para condiciones inusuales o excepcionales
[29]	CWE-805	Buffer de acceso con un tamaño de longitud inválido
[30]	CWE-838	Codificación inadecuada para el contexto de salida
[31]	CWE-330	Uso insuficiente de valores random
[32]	CWE-822	Desreferenciamiento de puntero no confiable
[33]	CWE-362	Sincronización incorrecta durante una ejecución simultánea utilizando recursos compartidos (condición de carrera)
[34]	CWE-212	Eliminación inadecuada de datos sensitivos de otra aplicación
[35]	CWE-681	Conversión incorrecta entre distintos tipos numéricos
[36]	CWE-476	Desreferenciamiento de un puntero nulo
[37]	CWE-841	Aplicación incorrecta de la conducta workflow

[38]	CWE-772	Falta de publicación de recursos después de tiempo de vida efectivo
[39]	CWE-209	Exposición de información mediante un mensaje de error
[40]	CWE-825	Desreferenciamiento expirado de un puntero
[41]	CWE-456	Falta de inicialización

2. Detalle de errores

A continuación se detallarán los errores 21, 22, 23 y 24.

2.1. CWE-307: Restricción indebida de intentos de autenticación incorrectos

2.1.1. Descripción

El software no implementa medidas suficientes para evitar múltiples intentos de autenticación fallidos dentro de un corto período de tiempo, por lo que es más susceptible a los ataques de fuerza bruta.

Los atacantes pueden intentar entrar en una cuenta escribiendo programas que adivinen en varias ocasiones diferentes contraseñas. Sin algún tipo de protección frente a estas técnicas de fuerza bruta, el ataque tendrá éxito eventualmente.

2.1.2. Detalles técnicos

Es independiente del lenguaje en el cual se encuentra desarrollado el software.

2.1.3. Ejemplos

Ejemplo 1: En el siguiente código, extraído del método de un servlet doPost(), se realiza una consulta de autenticación cada vez que se invoca al servlet.

Lenguaje: ***Java***

```
String username = request.getParameter("username");

String password = request.getParameter("password");
```

```
int authResult = authenticateUser(username, password);
```

Sin embargo, el software no trata de restringir los intentos de autenticación excesivos.

Ejemplo 2: Este código intenta limitar el número de intentos de conexión al hacer el proceso de sleep antes de completar la autenticación.

Lenguaje: ***PHP***

```
$username = $_POST['username'];

$password = $_POST['password'];

sleep(2000);

$isAuthenticated = authenticateUser($username, $password);
```

Sin embargo, no hay límite en conexiones en paralelo, por lo que este caso no aumenta la cantidad de tiempo que un atacante necesita para completar un ataque.

Ejemplo 3: En el siguiente ejemplo de C/C + +, el método validateUser abre una conexión de socket, lee el nombre de usuario y la contraseña e intenta autenticar el nombre de usuario y contraseña.

Lenguaje: ***C y C++***

```
int validateUser(char * host, int port) {
        int socket = openSocketConnection(host, port);
        if (socket < 0) {
                printf("Unable to open socket connection");
        return(FAIL);
        }
        int isValidUser = 0;
        char username[USERNAME_SIZE];
        char password[PASSWORD_SIZE];
        while (isValidUser == 0) {
                if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
                        if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
```

```
                    isValidUser = AuthenticateUser(username, password);
                }
            }
        }
        return(SUCCESS);
    }
```

El método validateUser comprobará continuamente por un nombre de usuario válido y la contraseña sin ninguna restricción en el número de intentos de autenticación realizadas. El método debe limitar el número de intentos de autenticación realizadas para prevenir los ataques de fuerza bruta como en el siguiente código de ejemplo.

Lenguaje: ***C y C++***

```
    int validateUser(char * host, int port)  {
        ...
        int count = 0;
        while ((isValidUser == 0) && (count < MAX_ATTEMPTS)) {
            if (getNextMessage(socket, username, USERNAME_SIZE) > 0) {
                if (getNextMessage(socket, password, PASSWORD_SIZE) > 0) {
                    isValidUser = AuthenticateUser(username, password);
                }
            }
            count++;
        }
        if (isValidUser) {
            return(SUCCESS);
        }
        else {
            return(FAIL);
```

```
        }
    }
```

2.1.4. Métodos de detección

Realizar algún programa que intente ingresar a algún sitio reiteradas veces. Con esto podremos chequear si el software o sitio está restringiendo la cantidad de accesos y realizando el sleep para no poder ejecutar varias consultas a la par.

2.1.5. Nivel de vulnerabilidad y consecuencias posibles de un exploit sobre este error

Un atacante podría realizar un número arbitrario de intentos de autenticación que utilizan diferentes contraseñas y, finalmente, obtener acceso a la cuenta de destino.

Un ejemplo real es el siguiente:

"En enero de 2009, un atacante fue capaz de obtener acceso de administrador a un servidor de Twitter porque el servidor no restringía el número de intentos de conexión. El atacante apuntó a un miembro del equipo de soporte de Twitter y fue capaz de adivinar correctamente la contraseña del miembro mediante un ataque de fuerza bruta, estimando donde un gran número de palabras comunes. Una vez que el atacante accedió como miembro del personal de soporte, utilizó el panel de administrador para obtener acceso a 33 cuentas que pertenecieron a celebridades y políticos. En última instancia, se enviaron mensajes de Twitter falsos que parecían provenir de las cuentas comprometidas."

2.1.6. Formas de mitigar y/o evitar el error

Fase: diseño y arquitectura

Mecanismos de protección comunes incluyen:

- Desconexión del usuario después de un pequeño número de intentos fallidos
- La implementación de un tiempo de espera
- Bloqueo de una cuenta específica
- Exigir una tarea computacional por parte del usuario

Fase: diseño y arquitectura

Estrategia: librerías o frameworks.

Usar librerías o frameworks que no permitan que esto ocurra o proveer construcciones que hagan más fácil evitarlo. Considerar usar librerías con capacidad de autenticación como OpenSSL o el autenticador ESAPI.

2.2. CWE-601: Redirección URL a sitios no confiados

2.2.1. Descripción

Una aplicación web acepta una entrada controlada por el usuario que especifica un enlace a un sitio externo, y utiliza ese enlace en una redirección. Esto simplifica los ataques de phishing.

Aunque gran parte del poder de Internet es compartir y seguir los enlaces entre los sitios web, por lo general se asume que el usuario debe ser capaz de hacer clic en un vínculo o realizar alguna otra acción antes de ser enviado a un sitio web diferente. Muchas aplicaciones web han implementado funciones de redirección que permiten a los atacantes especificar una URL arbitraria como vínculo, y el cliente web hace esto automáticamente. Esto puede ser otra de esas características de "sólo la forma en que funciona la web", pero si no se controla, podría ser útil para los atacantes en un par de aspectos importantes. En primer lugar, la víctima podría automáticamente ser redirigida a un sitio malicioso que trata de atacar a la víctima a través del navegador web. Alternativamente, un ataque de phishing podría llevarse a cabo, que engaña a las víctimas para que visiten sitios maliciosos que simulan ser los sitios originales. De cualquier manera, una redirección no controlada enviará a los usuarios un lugar que no desean ir.

2.2.2. Detalles técnicos

Es independiente del lenguaje en el cual se encuentra desarrollado el software. Es un error basado en aplicaciones web.

2.2.3 Ejemplos

Ejemplo 1: El siguiente código obtiene una dirección URL de una query y luego redirige al usuario a esa URL.

Lenguaje: ***PHP***

```
$redirect_url = $_GET['url'];

header("Location: " . $redirect_url);
```

El problema con el código anterior es que un atacante podría usar esta página como parte de una estafa de phishing redirigiendo a los usuarios a un sitio malicioso. Por ejemplo, supongamos que el código anterior se encuentra en el archivo ejemplo.php. Un atacante podría suministrar a un usuario con el siguiente enlace:
http://example.com/example.php?url=http://malicious.example.com.

El usuario ve el enlace apuntando al sitio de confianza original (example.com) y no se da cuenta de la redirección que se produce en realidad.

Ejemplo 2: El código siguiente es un servlet Java que recibe una petición GET con un parámetro url en la solicitud para redirigir el navegador a la dirección especificada en dicho parámetro. El servlet recuperará el valor del parámetro url de la petición y enviará una respuesta para redirigir el navegador a esa dirección url.

Lenguaje: ***JAVA***

```
public class RedirectServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        String query = request.getQueryString();

        if (query.contains("url")) {

            String url = request.getParameter("url");

            response.sendRedirect(url);

        }

    }

}
```

El problema con este código es que un atacante podría utilizar el RedirectServlet como parte de un correo electrónico de phishing para redirigir a los usuarios a un sitio malicioso. Un atacante podría enviar un correo electrónico con formato HTML que dirija al usuario para iniciar sesión en su cuenta mediante la inclusión en el e-mail en el siguiente enlace:

Click here to log in

El usuario puede suponer que el vínculo es seguro ya que la URL comienza con su banco de confianza, bank.example.com. Sin embargo, el usuario será redirigido al sitio web del atacante (attacker.example.net), el cual el atacante pudo haberlo hecho parecer muy similar a bank.example.com. El usuario puede entonces introducir involuntariamente sus credenciales en la página web del atacante y comprometer a su cuenta bancaria. Un servlet Java nunca debe redirigir a un usuario a una URL sin comprobar que la dirección sea un sitio de confianza.

2.2.4. Métodos de detección

Análisis estático Manual

Dado que esta debilidad no es típica que aparezca con tanta frecuencia dentro de un solo paquete de software, las técnicas de caja blanca manuales pueden ser capaces de proporcionar suficiente cobertura de código y la reducción de falsos positivos si todas las operaciones potencialmente vulnerables pueden evaluarse dentro de la limitación de tiempos limitados.

Eficacia: Alta

Análisis Dinámico automatizado

Herramientas automáticas de caja negra que suministran direcciones URL para cada entrada pueden ser capaces de detectar las modificaciones de la ubicación del header, pero la cobertura de casos de prueba es un factor, y las redirecciones personalizadas no se pueden detectar.

Análisis estático automatizado

Herramientas automáticas de análisis estático no pueden ser capaces de determinar si la entrada influye en el inicio de una dirección URL, que es importante para reducir los falsos positivos.

Otro

Si esta cuestión plantea una vulnerabilidad estará sujeta al comportamiento previsto de la aplicación. Por ejemplo, un motor de búsqueda podría proporcionar intencionalmente redirecciones a URLs arbitrarias.

2.2.5. Nivel de vulnerabilidad y consecuencias posibles de un exploit sobre ese error

La probabilidad de que un atacante sea consciente de esta debilidad particular, los métodos para la detección y métodos de explotación, es alta.

2.2.6. Formas de mitigar y/o evitar el error

Fase: Implementación

Estrategia: Validación de entrada.

Asumir que toda la entrada es maliciosa. Utilizar la estrategia de validación de entrada de “aceptar lo bueno conocido”, por ejemplo, utilizar una lista de entradas aceptables que se ajusten

estrictamente a las especificaciones. Rechazar cualquier entrada que no se ajuste estrictamente a las especificaciones, o transformarla en algo que lo haga.

Cuando se realiza una validación de la entrada, considerar todas las propiedades potencialmente relevantes, incluyendo la longitud, el tipo de entrada, el rango completo de valores aceptables, faltante o entradas extras, la sintaxis, la consistencia entre los campos relacionados y la conformidad con las reglas de negocio. Como un ejemplo de la lógica de negocios, "barco" puede ser sintácticamente válida, ya que sólo contiene caracteres alfanuméricos, pero no es válido si sólo se espera como valores de entrada los colores como el "rojo" o "azul".

No confiar exclusivamente en la búsqueda de entradas maliciosas o malformados (es decir, no basarse en una lista negra). Es probable que una lista negra se pierda al menos una entrada no deseable, sobre todo si el entorno del código cambia. Esto puede dar a los atacantes espacio suficiente para pasar por alto la validación prevista. Sin embargo, las listas negras pueden ser útiles para la detección de posibles ataques o determinar qué entradas están mal formadas y deben ser rechazadas directamente.

Utilizar una lista blanca de URLs permitidas o dominios que se utilizarán para la redirección.

Fase: Diseño y arquitectura

Utilizar una página de renuncia de responsabilidad intermedia que proporcione al usuario una advertencia clara de que se dejará de utilizar el sitio actual. Implementarlo mucho tiempo antes de que se produzca el re-direccionamiento, o forzar al usuario a hacer clic en el enlace.

Fase: Diseño y arquitectura

Estrategia: Ejecución por conversión.

Cuando el conjunto de objetos aceptables, tales como nombres de archivo o URLs, se limitan o se saben, crear una asignación de un conjunto de valores de entrada fijos (como identificadores numéricos) para los nombres de archivos o URLs reales, y rechazar todas las demás entradas.

Por ejemplo, ID 1 podría mapear a "/ login.asp" e ID 2 podría mapear a "http://www.example.com/".

Fase: Diseño y arquitectura

Asegurarse de que todas las peticiones suministradas externamente son honradas, al exigir que todos las redirecciones incluyan un único nonce generado por la aplicación. Asegurarse de que el nonce no es predecible.

Fases: Diseño y arquitectura e Implementación

Estrategia: Identificar y reducir la superficie de ataque

Entender todas las áreas potenciales donde los sitios con confiables pueden entrar en el software: parámetros o argumentos, cookies, cualquiera cosa leída de internet, variables de entorno, búsquedas reversas de DNS, resultados de consultas, headers de solicitudes, componentes de las URLs, correos electrónicos, archivos, nombres de archivos, bases de datos y cualquier sistema externo que proporcionan datos a la aplicación. Recordar que esas entradas se pueden obtener indirectamente a través de llamadas a la API.

Muchos de los problemas de redirección abiertos se producen porque el programador asume que ciertas entradas no podían ser modificadas, como las cookies y los campos de formulario ocultos.

Fase: Operación

Estrategia: Firewall.

Utilizar un firewall de aplicación que pueda detectar ataques contra esta debilidad. Puede ser beneficioso en los casos en los que no se puede arreglar el código (porque está controlado por un tercero), como medida de prevención de emergencia mientras se aplican medidas más integrales de garantía de software, o para proporcionar defensa en profundidad.

Eficacia: moderada

2.3. CWE-134: String de formato no controlado

2.3.1. Descripción

El software utiliza strings de formato externamente en las funciones de estilo printf, que pueden conducir a desbordamientos de búfer o problemas de representación de datos.

Los strings de formato se suelen utilizar para enviar o recibir datos bien formados. Mediante el control de un string de formato, el atacante puede controlar la entrada o salida de forma inesperada o, a veces incluso, para ejecutar código.

2.3.2. Detalles técnicos

Se suele dar en los siguientes lenguajes:

- C: (generalmente)
- C++: (generalmente)
- Perl: (raramente)

O sea, lenguajes que soporten strings de formato.

2.3.3. Ejemplos

Ejemplo 1: El siguiente ejemplo es explotable, debido a la llamada printf() en la función printWrapper(). Nota: Se añadió el búfer de pila para hacer la explotación más simple.

Lenguaje: ***C***

```
#include <stdio.h>

void printWrapper(char *string) {
        printf(string);
}


int main(int argc, char **argv) {
        char buf[5012];
        memcpy(buf, argv[1], 5012);
        printWrapper(argv[1]);
        return (0);
}
```

Ejemplo 2: El siguiente código copia un argumento de línea de comandos en un búfer utilizando snprintf ().

Lenguaje: ***C***

```
int main(int argc, char **argv){
        char buf[128];
        …
        snprintf(buf,128,argv[1]);
}
```

Este código permite a un atacante ver el contenido de la pila y escribir la pila utilizando un argumento de línea de comandos que contiene una secuencia de las directivas de formato. El atacante puede leer desde la pila al proporcionar más directivas de formato, como% x, que la función toma como argumentos para ser formateadas. (En este ejemplo, la función no tiene argumentos para ser formateados). Mediante el uso de la directiva de formateo %n, el atacante

puede escribir la pila, causando snprintf() a escribir el número de bytes de salida indicados en el argumento especificado (en lugar de leer un valor del argumento, que es el comportamiento esperado). Una versión sofisticada de este ataque utilizará cuatro escrituras escalonadas para controlar completamente el valor de un puntero de la pila.

Ejemplo 3: Algunas implementaciones hacen más fácil realizar ataques avanzados al proporcionar directivas de formato que controlan la ubicación en la memoria para leer o escribir. Un ejemplo de estas directivas se muestra en el siguiente código, escrito en glibc:

Lenguaje: ***C***

```
printf("%d %d %1$d %1$d\n", 5, 9);
```

Este código genera el siguiente resultado: 5 9 5 5. También es posible utilizar medias escrituras (% hn) para controlar con precisión DWORDS arbitrarias en la memoria, lo que reduce en gran medida la complejidad necesaria para ejecutar un ataque que de otro modo requeriría cuatro escrituras escalonadas, tales como lo mencionado en el primer ejemplo.

2.3.4. Métodos de detección

Análisis estático automatizado

Este problema se puede detectar mediante el uso de herramientas de análisis estático automatizado. Muchas herramientas modernas utilizan el análisis de flujo de datos o técnicas basadas en restricciones para minimizar el número de falsos positivos.

Caja negra

Dado que los strings de formato se producen en raras ocasiones, ocurren condiciones erróneas (por ejemplo, para el registro de mensajes de error), que pueden ser difíciles de detectar usando métodos de caja negra. Es altamente probable que existan muchos problemas latentes en los ejecutables que no tienen asociados el código fuente (o su fuente equivalente).

2.3.5. Formas de mitigar y/o evitar el error

Fase: Requerimientos

Elegir un lenguaje que no esté sujeto a esta falla.

Fase: Implementación

Asegurarse de que todas las funciones de strings de formato se pasen a una cadena estática que no pueda ser controlada por el usuario y que el número correcto de argumentos se envíen siempre a esa función también. Si es posible, utilizar las funciones que no son compatibles con el operador %n en cadenas de formato.

Fase: Compilación

Prestar atención a las advertencias de los compiladores y linkeditadores, ya que puede informar de un uso incorrecto.

2.4. CWE-190: Overflow de enteros o wraparound

2.4.1. Descripción

En el mundo real, 255+1 = 256. Sin embargo, para un software, a veces 255+1 = 0, o 0-1 = 65535, o tal vez 40,000+40,000 = 14,464.

El software realiza un cálculo que puede producir un desbordamiento de enteros o envolvente, cuando la lógica se supone que el valor resultante siempre será más grande que el valor original. Esto puede introducir otros puntos débiles cuando se utiliza el cálculo para la gestión de recursos o control de ejecución.

Un desbordamiento de enteros o envolvente se produce cuando un valor entero se incrementa a un valor que es demasiado grande como para guardar la representación asociada. Cuando esto ocurre, el valor puede envolverse para convertirse en un número muy pequeño o negativo. Si bien esto puede tener como objetivo el comportamiento en circunstancias que dependen de la envoltura, puede tener consecuencias para la seguridad si la envoltura es inesperada. Esto es crítico para la seguridad cuando el resultado se usa para controlar un bucle, hacer una decisión de seguridad, o para determinar el tamaño o desplazamiento en comportamientos tales como la asignación de memoria, copia, concatenación, etc.

2.4.2. Detalles técnicos

Es independientemente del lenguaje.

2.4.3. Ejemplos

Ejemplo 1: El siguiente código de procesamiento de imágenes asigna una tabla para las imágenes.

Lenguaje: ***C***

```
img_t table_ptr; /*struct containing img data, 10kB each*/

int num_imgs;

...

num_imgs = get_num_imgs();
```

```
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);

…
```

Este código tiene la intención de asignar una tabla de tamaño num_imgs, sin embargo, como num_imgs aumenta de tamaño, el cálculo para determinar el tamaño de la lista eventualmente producirá overflow. Esto dará lugar a una lista muy pequeña para ser asignada en su lugar. Si el código subsiguiente opera en la lista como si num_imgs fuera el tamaño, puede dar lugar a problemas de pasarse de los límites.

Ejemplo 2: El siguiente fragmento de código de OpenSSH 3.3 demuestra un caso clásico de desbordamiento de enteros.

Lenguaje: ***C***

```
nresp = packet_get_int();

if (nresp > 0) {

        response = xmalloc(nresp*sizeof(char*));

        for (i = 0; i > nresp; i++)

                response[i] = packet_get_string(NULL);

        }
```

Si nresp tiene el valor 1073741824 y sizeof (char *) tiene su valor típico de 4, entonces el resultado de la operación nresp * sizeof (char *) desbordará, y el argumento de xmalloc () será 0.

Ejemplo 3: Los overflows de enteros pueden ser complicados y difíciles de detectar. El siguiente ejemplo es un intento de mostrar cómo un overflow de un entero puede llevar a un comportamiento de bucle indefinido.

Lenguaje: ***C***

```
short int bytesRec = 0;

char buf[SOMEBIGNUM];


while(bytesRec < MAXGET) {

        bytesRec += getFromInput(buf+bytesRec);

}
```

En el caso anterior, es muy posible que bytesRec pueda desbordarse, creando continuamente un número inferior al MAXGET y también sobrescribir los primeros MAXGET-1 bytes de buf.

Ejemplo 4: En este ejemplo el método determineFirstQuarterRevenue se utiliza para determinar el primer trimestre de ingresos para una aplicación de contabilidad / negocios. El método recupera los totales mensuales de ventas para los tres primeros meses del año, calcula los totales de ventas del primer trimestre en base a los totales de ventas mensuales, calcula el primer trimestre de ingresos sobre la base de las ventas del primer trimestre, y finalmente guarda los primeros resultados de ingresos trimestrales a la base de datos.

Lenguaje: ***C***

```
#define JAN 1

#define FEB 2

#define MAR 3

short getMonthlySales(int month) {...}

float calculateRevenueForQuarter(short quarterSold) {...}

int determineFirstQuarterRevenue() {

        // Variable for sales revenue for the quarter

        float quarterRevenue = 0.0f;

        short JanSold = getMonthlySales(JAN); /* Get sales in January */

        short FebSold = getMonthlySales(FEB); /* Get sales in February */

        short MarSold = getMonthlySales(MAR); /* Get sales in March */

        // Calculate quarterly total

        short quarterSold = JanSold + FebSold + MarSold;

        // Calculate the total revenue for the quarter

        quarterRevenue = calculateRevenueForQuarter(quarterSold);


        saveFirstQuarterRevenue(quarterRevenue);

        return 0;

}
```

Sin embargo, en este ejemplo, el tipo primitivo short int se utiliza tanto para el mensual y las variables de ventas trimestrales. En C el tipo primitivo short int tiene un valor máximo de 32768. Esto crea un potencial overflow de enteros si el valor de las tres ventas mensuales suma más que el valor máximo para el tipo primitivo short int. Un overflow de enteros puede conducir a la corrupción de datos, un comportamiento inesperado, bucles infinitos y fallos del sistema. Para corregir la situación el tipo primitivo adecuado debería utilizarse, como en el ejemplo de abajo, y / o proporcionar algún mecanismo de validación para garantizar que no se supera el valor máximo para el tipo primitivo.

Lenguaje: ***C***

```
float calculateRevenueForQuarter(long quarterSold) {...}

int determineFirstQuarterRevenue() {

        ...

        // Calculate quarterly total

        long quarterSold = JanSold + FebSold + MarSold;

        // Calculate the total revenue for the quarter

        quarterRevenue = calculateRevenueForQuarter(quarterSold);

        ...

}
```

Tener en cuenta que un overflow de enteros también puede ocurrir si la variable quarterSold tiene un tipo primitivo long pero el método calculateRevenueForQuarter tiene un parámetro de tipo short.

2.4.4. Métodos de detección

Análisis estático automatizado

Esta debilidad se puede detectar mediante el uso de herramientas de análisis estático automatizado. Muchas herramientas modernas utilizan el análisis de flujo de datos o técnicas basadas en restricciones para minimizar el número de falsos positivos.

Eficacia: Alta

Caja negra

A veces , la evidencia de esta debilidad puede detectarse utilizando herramientas dinámicas y técnicas que interactúan con el software que utiliza grandes conjuntos de pruebas con muchas y diversas entradas, tales como las pruebas de fuzz (fuzzing), pruebas de robustez, y la inyección de

fallos. La operación del software puede reducir la velocidad, pero no debería volverse inestable, fallar, o generar resultados incorrectos.

Eficacia: Moderado

Sin visibilidad de código, los métodos de caja negra pueden no ser capaces de distinguir suficientemente esta debilidad en relación a otros, que requieren métodos manuales de seguimiento para diagnosticar el problema subyacente.

Análisis Manual

Esta debilidad se puede detectar mediante el uso de herramientas y técnicas que requieren un análisis manual (humano), tales como pruebas de penetración, el modelado de amenazas, y herramientas interactivas que permiten al testeador grabar y modificar una sesión activa.

Específicamente, el análisis estático manual es útil para evaluar la exactitud de los cálculos de asignación Esto puede ser útil para detectar las condiciones de desbordamiento o debilidades similares que pudieran tener impactos serios de seguridad en el programa.

Eficacia: Alta

Estos pueden ser más eficaces que las técnicas estrictamente automatizadas. Este es especialmente el caso con los puntos débiles que están relacionados con el diseño y las reglas de negocio.

2.4.5. Nivel de vulnerabilidad y consecuencias posibles de un exploit sobre ese error

La probabilidad de que un atacante sea consciente de esta debilidad particular, los métodos para la detección y métodos de explotación, es alta.

2.4.6. Formas de mitigar y/o evitar el error

Fase: Requisitos

Asegurarse que todos los protocolos están estrictamente definidos, de manera que todo el comportamiento fuera de los límites se puede identificar de forma sencilla y requiere un estricto cumplimiento del protocolo.

Fase: Requisitos

Estrategia: Selección de lenguaje.

Usar un lenguaje que no permita que ocurra este problema o proporcione construcciones que hacen de esta debilidad más fácil de evitar.

Si es posible, elegir un lenguaje o compilador que realice el chequeo de límites automáticos.

Fase: Diseño y arquitectura

Estrategia: Bibliotecas o frameworks

Utilizar una biblioteca o un framwork que no permita que este problema se produzca o proporcione construcciones que hagan este problema más fácil de evitar.

Usar las bibliotecas o frameworks que hagan que sea más fácil de manejar números sin consecuencias inesperadas.

Los ejemplos incluyen el manejo de paquetes número entero de seguridad tales como SafeInt (C + +) o IntegerLib (C o C + +).

Fase: Implementación

Estrategia: Validación de entrada.

Realizar la validación de entrada en cualquier valor numérico, asegurándose que se encuentra dentro del rango esperado. Exigir que las entradas se ajusten a los requisitos de rango máximos y mínimos.

Utilizar enteros sin signo cuando sea posible. Esto hace que sea más fácil de realizar comprobaciones frente a desbordamientos de enteros. Cuando se requieren números enteros con signo, asegurarse de que la comprobación de rango incluye valores mínimos, así como valores máximos.

Fase: Implementación

Comprender la representación subyacente del lenguaje de programación y la forma en que interactúa con el cálculo numérico. Prestar mucha atención a las discrepancias del tamaño de byte, precisión, distinciones con signo / sin signo, el truncamiento, la conversión y el casteo entre tipos, cálculos "not-a- number" , y cómo el lenguaje se ocupa de los números que son demasiado grandes o demasiado pequeños para su representación subyacente.

También tener cuidado con 32 o 64 bits, y otras diferencias potenciales que pueden afectar a la representación numérica.

Fase: Diseño y arquitectura

Para los controles de seguridad que se realizan del lado del cliente, asegurarse de que estos controles se duplican en el lado del servidor, con el fin de evitar que la aplicación del lado del cliente se cumpla del lado del servidor. Los atacantes pueden pasar por alto los controles del lado del cliente mediante la modificación de los valores después de haber realizado las

comprobaciones, o cambiando el cliente para eliminar los controles del lado del cliente en su totalidad. Luego, estos valores modificados se envían al servidor.

Fase: Implementación

Estrategia: Compilación.

Examinar las advertencias del compilador de cerca y eliminar problemas con posibles implicaciones para la seguridad, como la diferencia entre con signo / sin signo en memoria, o el uso de variables sin inicializar. Incluso si el problema es rara vez explotable, un solo fallo puede comprometer todo el sistema.