**Guía Mejoras TOL**

**Sql Injection - Cross Site Scripting**

**Noviembre 2021**

buhos.cl

El siguiente documento presenta una serie de algoritmos y códigos que previenen Cross Site Scripting y Sql Injection.

1.- Cross Site Scripting

En archivos php que manejan contenido PHP y html es posible incorporar el siguiente código:

|  |
| --- |
| fix\_crs();   /\*\*  \* Fix Cross Site Scripting  \*/  function fix\_crs(){  if(!empty($\_POST)){  $\_POST = clean\_crs($\_POST);  }    if(!empty($\_GET)){  $\_GET = clean\_crs($\_GET);  }    if(!empty($\_REQUEST)){  $\_REQUEST = clean\_crs($\_REQUEST);  }    }    function clean\_crs($arr){  if(!empty($arr)){  foreach($arr as $kk => $vv){  $value = $vv;   $value= htmlspecialchars($value);  $value = htmlentities($value);  $value = strip\_tags($value);  $arr[$kk] = $value;  }   }  return $arr;  } |

Estas funciones sanitizan y previenen la incorporación de Cross-Site Scripting para contenido con origen POST, GET, REQUEST

2.- Sql Injection PHP Híbrido

En archivos php que manejan contenido PHP y html es posible incorporar el siguiente código:

|  |
| --- |
| $variable = fix\_sql\_injection($variable);  /\*\*  \* Escape de datos para eviar sql injection  \*/  function fix\_sql\_injection($data) {    if ( is\_numeric($data) ) { return $data; }    $non\_displayables = array(  '/%0[0-8bcef]/', // url encoded 00-08, 11, 12, 14, 15  '/%1[0-9a-f]/', // url encoded 16-31  '/[\x00-\x08]/', // 00-08  '/\x0b/', // 11  '/\x0c/', // 12  '/[\x0e-\x1f]/' // 14-31  );  foreach ( $non\_displayables as $regex ) {  $data = preg\_replace( $regex, '', $data );  }    $data = str\_replace("'", "''", $data );  return $data;  } |

En este escenario es factible sanitizar los parámetros de entrada proveniente de POST GET REQUEST, SESSION invocando a la función fix\_sql\_injection la cual devuelve el contenido sanitizado para prevenir Sql Injection.

3.- Sql Injection PHP Classes

|  |
| --- |
| class Tol {    public function doSomething($parametro){  $parametro = $this->fix\_sql\_injection($parametro);   //doSometing   }   /\*\*  \* Escape de datos para eviar sql injection  \*/  private function fix\_sql\_injection($data) {    if ( is\_numeric($data) ) { return $data; }    $non\_displayables = array(  '/%0[0-8bcef]/', // url encoded 00-08, 11, 12, 14, 15  '/%1[0-9a-f]/', // url encoded 16-31  '/[\x00-\x08]/', // 00-08  '/\x0b/', // 11  '/\x0c/', // 12  '/[\x0e-\x1f]/' // 14-31  );  foreach ( $non\_displayables as $regex ) {  $data = preg\_replace( $regex, '', $data );  }    $data = str\_replace("'", "''", $data );  return $data;  }   } |

En el escenario de una clase con acceso a base de datos se puede sanitizar los datos y prevenir Sql Injection utilizando la función fix\_sql\_injection para cada parámetro de entrada.

Conclusiones

Es factible manejar y prevenir la incorporación de código malicioso mediante técnicas de sanitización de parámetros evitando Sql Injection, Cross Site Scripting en otros tipo de ataque que afecten a las entradas y salidas de datos del sistema.