

Como evitaríamos los ataques a FarmSpace, mediante la desinfección de entradas.

Tres de los cinco ataques a aplicaciones más comunes tales como inyección SQL , Cross-site scripting (XSS) e inclusión remota de archivos (RFI). comparten algo en común que es la necesidad de su desinfección de entrada.

Los tres exploits representarían una amenaza para nuestra aplicación cuando los datos son enviados al servidor por el usuario final. Si el usuario final es un buen tipo, los datos que enviará al servidor son relevantes para su interacción con el sitio web. Pero cuando el usuario final es un pirata informático, puede aprovechar este mecanismo para enviar la entrada del servidor web, que se construye deliberadamente para escapar del contexto legítimo y podría ejecutar acciones no autorizadas.

La desinfección de entrada o más conocida como "*Sanitize input*" se utilizará para la limpieza y el restregado de la entrada del usuario para evitar que un bot o alguien que quiera darle mal uso salte la cerca y explote los agujeros de seguridad. Pero la desinfección completa de los insumos es difícil.

Posibles peligros de los datos entrantes en FarmSpace:

En nuestra aplicación hay tres caminos que pueden tomar los datos para llegar desde la aplicación de nuestro usuario al servidor:

OBTENCIÓN DE SOLICITUDES: Estos son parámetros incluidos en la URL, a menudo generados por la entrada del formulario en una página web. Los parámetros en una solicitud GET aparecen después del signo de interrogación en una URL.

<http://example.com/page?parameter=value&also=another>

Cualquiera puede manipular fácilmente los datos en una solicitud GET simplemente editando la URL.

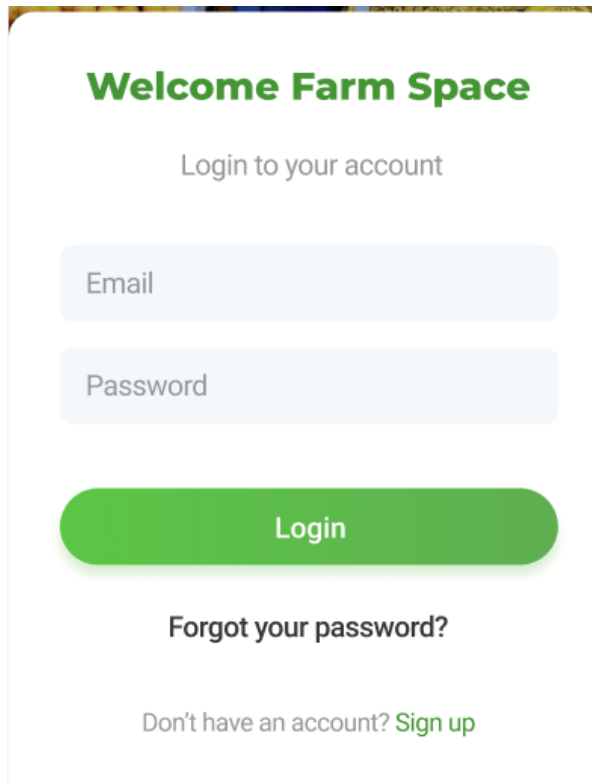
SOLICITUDES POST: Estos son parámetros incluidos en la información del encabezado enviada desde el navegador al servidor web. Los datos POST no aparecen en la URL, pero los piratas informáticos pueden manipular nuestra aplicación mediante complementos del navegador como Tamper Data simplemente con código personalizado utilizando una biblioteca como cURL.

COOKIES: Las cookies creadas por nuestra aplicación pueden ser un problema ya que pueden contener datos explotables. Las cookies se almacenan como archivos de texto sin formato en la máquina del usuario final en la que un pirata informático puede modificarlas fácilmente para manipular los datos de entrada enviados al servidor.

Posibles ataques y defensas:

La explotación de las debilidades de desinfección de entrada serán por PHP en nuestra aplicación.

ATAQUE: Formulario devuelto: En este ataque, el pirata informático explotaría el acceso a la aplicación, el cual devolvería un formulario incompleto haciendo eco de la propia entrada del usuario.



Welcome Farm Space

Login to your account

Email

Password

Login

Forgot your password?

Don't have an account? [Sign up](#)

```
<método de formulario = "GET" action =  
"submit.php">  
<input type = "text" name = "email"  
  value = "<? php echo $ _GET ['email'];?>" />  
<input type = "text" name = "password"  
  value = "<? php echo $ _GET ['password'];?>" />  
</form>
```

Imaginando un supuesto caso en el que el usuario envió el formulario anterior, pero se devolvió por un error de validación, tal vez no pudo completar un campo obligatorio. El formulario hace eco de la entrada del usuario para que los campos ya estén completados con su entrada anterior. Pero no hay desinfección de entrada: el código simplemente repite la entrada exacta del usuario. Ahora supongamos que el usuario ingresa estos datos en el campo "email":

```
"> <script> window.open (" http://hacksite.com ");  
</script>
```

Cuando se genera el formulario, este "email" hace algo furtivo: cierra la etiqueta INPUT e inserta una etiqueta Javascript que ejecuta código para abrir una ventana emergente con una URL a un sitio web controlado por el pirata informático que podría incluir spam o malware. Peor aún, puede que este comentario se guarde en una base de datos y otros visitantes del sitio web puedan verlo, entonces podría engañar a sus navegadores para que abran la ventana emergente del sitio del pirata informático.

DEFENSA: Cualquier ataque como este, donde la entrada del usuario se repite en la página web, *requiere* que los datos sean desinfectados antes de la salida. Debido a que nos estamos defendiendo contra la inyección de nuestra aplicación, debemos revisar y quitar los caracteres especiales HTML, incluidos los corchetes de etiquetas (<>) y la entidad comercial (&) para que el navegador no los represente.

Los desarrolladores de PHP pueden usar la función *filter_input* para hacer el trabajo pesado:

```
$ safe_data = filter_input (INPUT_GET, 'email',  
FILTER_SANITIZE_SPECIAL_CHARS);
```

Debido a que esta función solo funciona en un solo parámetro GET a la vez, es posible que necesitemos escribir una función para crear una nueva matriz (por ejemplo, *\$ SAFE_GET*) que itera a través de los parámetros GET y los desinfecta todos de una vez.

Alternativamente, estableceremos una directiva en el archivo *php.ini* de forma predeterminada para desinfectar toda la entrada para la seguridad de HTML:

```
filter.default="special_chars"
```

ATAQUE: Atributos sin comillas . La especificación oficial de HTML no requiere que cite atributos de etiqueta HTML. Eso significa que ambas sintaxis son igualmente legales:

Ingresa sus datos



Productor



Consumidor

```
<a href=details.php?id=<?php echo $userid;?>>
```

```
<a href="details.php?id=<?php echo $userid;?>">
```

En la primera sintaxis sin las comillas que encierran el atributo *href* , podemos ponerlo en un nuevo atributo que pasa directamente por el filtro de desinfección:

```
http://farmspace.com/detalles.php?id=x+onclick=alert\(/hacked/\)
```

Dada esta URL, la página con el código anterior produce HTML con un enlace ahora manipulado para activar el propio código Javascript del hacker cuando se hace clic en él.

DEFENSA: Incluiremos todos los atributos entre comillas. Aunque en términos sencillos, esta defensa habla de la necesidad de disciplina y coherencia en el desarrollo de la aplicación como su propia defensa contra las vulnerabilidades de saneamiento de entradas.

ATAQUE: inyección SQL: Los ataques que podrían explotar nuestra base de datos SQL subyacente en que pueden aprovechar la desinfección de entrada

defectuosa. Sin embargo, es importante recalcar que la desinfección de entrada no es 100% eficaz contra la inyección de SQL.

Una de las primeras pruebas que podría hacer un atacante en FarmSpace es probar si su código envía consultas SQL sin ningún tipo de desinfección de datos. Por ejemplo, el hacker visitará nuestro formulario de inicio de sesión e ingresará una sola cita en su dirección de correo electrónico, como esta:

`badguy@hacker.net'`

Si nuestro backend no desinfecta esta entrada, causará un error de sintaxis cuando la base de datos interprete el SQL. Lo más probable es que la salida de error se envíe a la página web y el pirata informático ahora sabrá que puede construir entradas más sofisticadas para agregar cláusulas a la consulta SQL que pueden volcar datos de su base de datos.

DEFENSA: Revisaremos todas las entradas del usuario antes de incluirlas en una consulta SQL. Lo haremos usando la función *mysql_real_escape_string()*. Cabe recalcar la gran importancia de utilizar esta función en cualquier consulta SQL que realizaremos.