Segurança Ivo Machado

PROTEGENDO SUAS WEB APPLICATIONS - 4 DICAS

De forma crescente, atacantes estão explorando falhas em aplicações web, expondo empresas a diversas ameaças, incluindo roubo de informações, manipulação de seus dados e upload de malwares em sites corporativos vulneráveis, para que usuários desavisados sejam infectados.

Um caso recente foi da falha no OsCommerce, que permitiu que mais de 400.000 sites de comércio online fossem infectados com exploits para diversas versões do browser Internet Explorer.

Neste artigo, procuro descrever algumas das melhores formas de proteger sua aplicação contra esses ataques e, de forma prática e possível, elevar a segurança de sua aplicação.

Infelizmente, alguns desenvolvedores ainda deixam a validação de dados de entrada, assim como filtro de códigos de escape, fora de seu To-Do list; ou pior, muitas vezes esses filtros são criados de forma tão pobre que não conseguem impedir que ataques com potencial mais elevado tenham êxito.

Com esse overview na mente, vamos olhar com cuidado os métodos que todo desenvolvedor deve empregar para prevenir input-validation attacks e implementar de forma mais eficiente a filtragem de caracteres de escape. Vamos utilizar, então, essas dicas em forma de um Checklist

1 - ESPECIFIQUE SEMPRE OS TIPOS DE VARIÁVEIS

Limite o tipo de dados que podem ser introduzidos em certos campos. Por exemplo, se você apenas precisa aceitar um valor, Integer em um determinado campo, assegure-se que seu código rejeitará qualquer outro tipo de caractere que possa ser utilizado.

Uma forma de garantir isso é usando a classe validadora do ESAPI. Essa classe tem um método isValidInteger que pode ser utilizado da seguinte forma em um código Java:

```
Validator validator = ESAPI.validator();
boolean isValidInteger =
  validator.isValidInteger("Numero _ teste", "42",
0, 999, false);
```

2 - NÃO UTILIZE UMA BLACK-LIST DE CARACTERES;

É muito mais válido utilizar uma White-List aceitando apenas os que são bons.

Tentar criar uma lista compreensiva de todos os "bad characters" para todos os diferentes tipos de ataques é praticamente impossível. Contudo, ao criar um código de validação, defina quais caracteres devem ser aceitos utilizando uma White-List, como, por exemplo, A-Z, a-z, e 0-9, e rejeite todos os outros

"INFELIZMENTE, ALGUNS DESENVOLVEDORES AINDA DEIXAM A VALIDAÇÃO DE DADOS DE ENTRADA, ASSIM COMO FILTRO DE CÓDIGOS DE ESCAPE, FORA DE SEU TO-DO LIST; OU PIOR, MUITAS VEZES ESSES FILTROS SÃO CRIADOS DE FORMA TÃO POBRE QUE NÃO CONSEGUEM IMPEDIR QUE ATAQUES COM POTENCIAL MAIS ELEVADO TENHAM ÊXITO"

restantes. Esse tipo de abordagem tem um resultado muito melhor do que tentar sempre abastecer uma Black-List que sempre será usada de forma reativa.

3 - FAÇA LIMITAÇÃO DO TAMANHO DOS INPUTS

Se você pergunta a idade de alguém, manter um campo com três dígitos cobrirá todos possíveis casos, inclusive se a pessoa tiver mais de cem anos. Se você pergunta o nome de alguém, cem caracteres ou mais serão mais do que necessários. Dessa forma, mesmo que você já esteja filtrando os caracteres, isso ajudará a limitar também o estado real de cada campo, fazendo com que fique muito difícil para um atacante injetar caracteres utilizados em um possível ataque.

Para efetuar ambas as validações, tanto a dos caracteres em nossa White-List como o valor limite nos campos, podemos utilizar mais uma vez a classe Validator no Java. Dessa vez, entretanto, utilizando o método is Validlnput:

```
Validator validator = ESAPI.validator();
boolean isValidFirstName =
   validator.isValidInput("Test _ AccountName",
"Checkingl",
"AccountName", 100, false);
```

A expressão regular definida para AccountName no arquivo de configuração do ESAPI é la-zA-ZO-9]*$. Essa expressão regular essencialmente diz que o Input está permitido apenas para caracteres que contêm letras maiúsculas e minúsculas e números de zero a nove. Adicionalmente, na chamada do isValidlnput, especificamos que o maxLenght para o input não pode exceder cem caracteres. Tais regras de validação do Input bloquearão diversos tipos de ataques.

4 - UTILIZE CODIFICAÇÃO EM OUTPUTS

O Output Encoding é utilizado quando dados são enviados de volta para o usuário ou utilizados por outro componente na aplicação. Fazendo uma filtragem correta e depois aplicando a codificação, muitos tipos de ataques podem ser prevenidos.

Por exemplo, certos caracteres, como os símbolos de maior ou menor (<, >), são comumente utilizados em ataques de Cross-Site Scrtpting (XSS). Convertendo esses caracteres para sua entidade HTML equivalente (exemplo, "<" se torna "&lt" e ">" se toma "&gt"), muitos strings de ataque XSS podem ser prevenidos. A classe Encoder contém numerosos métodos que podem ser utilizados para codificar dados para diferentes situações.

Especificamente, o encodeForHTML method pode ser empregado para prevenir ataques HTML-Based XSS, como o exemplo a seguir

```
Encoder encoder = ESAPI.encoder();
String encodedString = encoder.
encodeForHTML("<script>alert
   ('xss')</script>");
```

Quando esse código é executado, a variável encodeString irá conter o seguinte valor, que é mais difícil de ser executado em um Browser como tentativa de XSS attack:

```
&lt;scriptSgt;alert&#40;S#39;xss&#39;S#41;Slt;S#47;s
cript&gt;
```

O método encodeForHTML atualmente utiliza o conceito de White-List para fazer sua codificação. Ela trata um limitado set de caracteres (letras, números, vírgula, ponto, banas, underscores e espaços) como seguro e o HTML codifica todo o resto. **x**

Ivo Machado
Diretor de operações da In2Sec - Intelligence to Security, formado em Análise de Sistemas com MBA em Gestão de Riscos, tendo já atuado em diversas empresas nacionais e multinacionais, no Brasil e no exterior. Especialista em Gestão de Riscos e Segurança da Informação, é responsável pelas operações das Américas da Certificadora TrustSign e das consultorias de segurança e-Horus e A.
E-mail: **ivo.machado@in2sec.com**

Fonte: Wide, São Paulo, ano 8, n. 87, p. 84-85, nov./ dez. 2011.