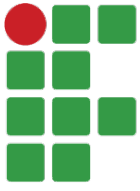
Programação Segura

Segurança Computacional

Ricardo de la Rocha Ladeira
{ricardo.ladeira@ifc.edu.br}



INSTITUTO FEDERAL
Catarinense
Campus Blumenau

Programação Segura

- **Programação segura** é o conjunto de procedimentos recomendados para programar de forma segura.
- São regras simples, mas as vezes negligenciadas, tais como utilizar
  - softwares atualizados
  - padrões de projetos
  - bibliotecas e frameworks confiáveis
- Todos já foram amplamente testados e desenvolvidos (em tese) considerando boas práticas.

Programação Segura

- Documentar corretamente.

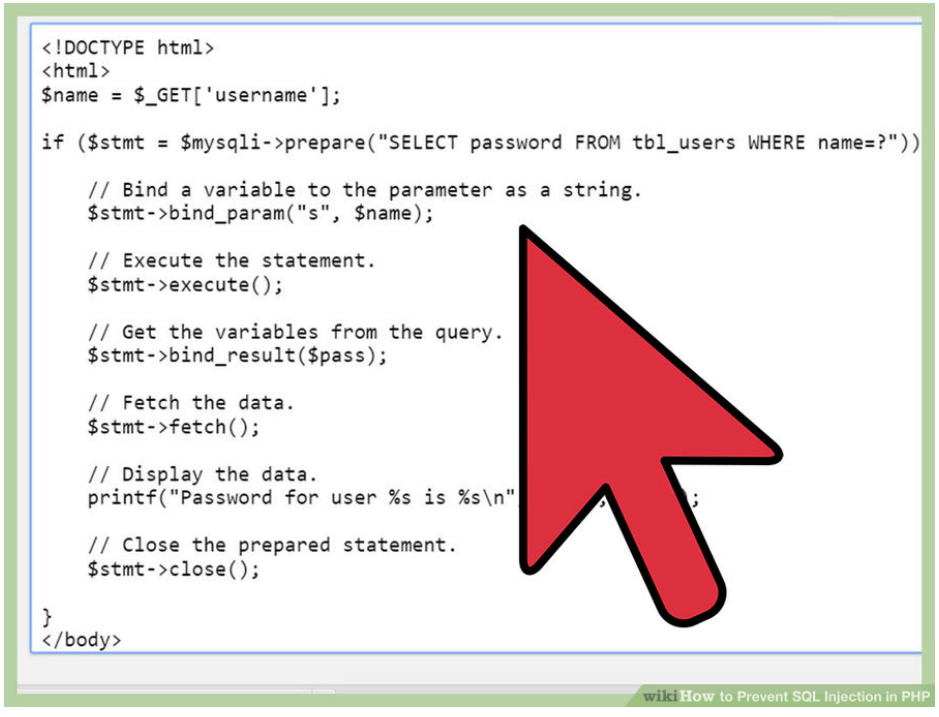
- Não omita informações.
- Não burle informações.
- Explique de forma que seja possível entender!

Programação Segura

- Realize tratamento em todo tipo de entrada de dados.
- Que tipo de ataque isso previne? SQL Injection, por exemplo, utilizando prepared statements.

Programação Segura

Figura: Prepared Statement.



Fonte: `http://www.wikihow.com/Prevent-SQL-Injection-in-PHP`

Programação Segura

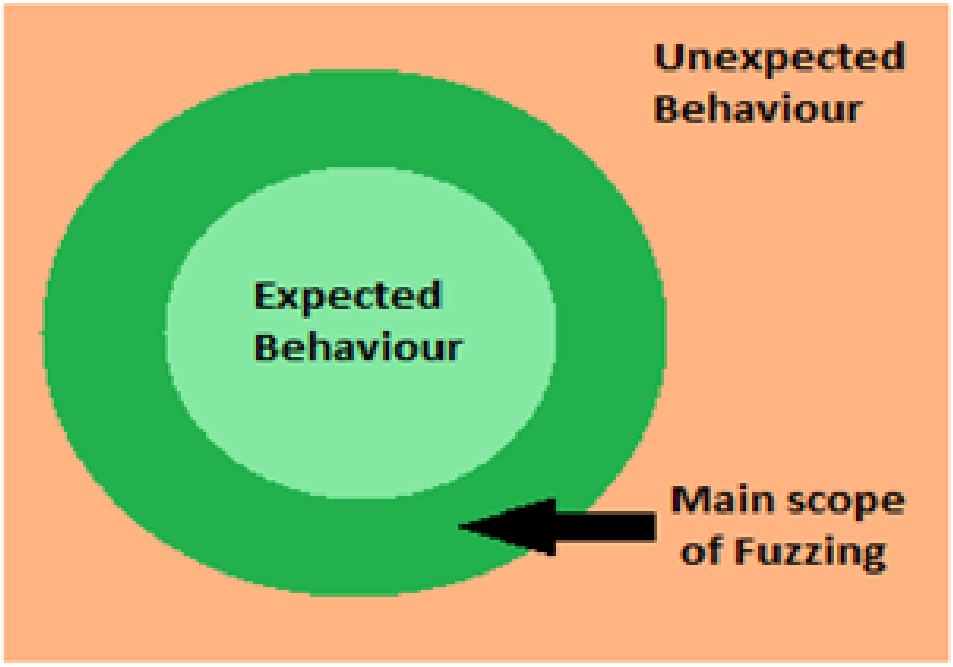
- Teste todo tipo de entrada de dados.
  - Insira uma palavra aonde se espera um número.
  - Insira um valor maior que o esperado, excedendo limites.
- Quais foram os resultados? Realize os tratamentos adequados nestes casos.

Programação Segura

- Utilize ferramentas de teste de segurança.
- Técnica fuzzing (confusão).
  - Ferramentas: Burp Suite, Peach Fuzzer, Radamsa...
- Verifique vulnerabilidades: `https://pentest-tools.com/home`

Programação Segura

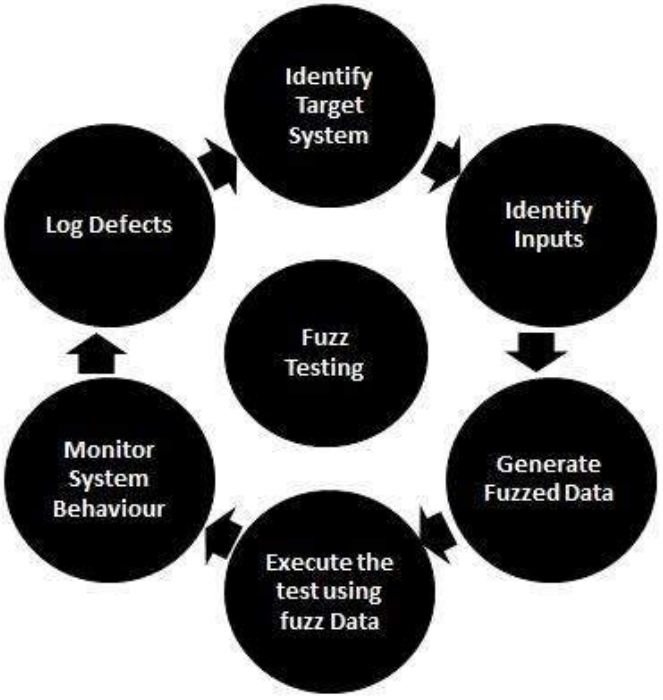
Figura: Fuzzing test.



Fonte: `http://resources.infosecinstitute.com/wp-content/uploads/010412_1727_FuzzingMuta2.png`

Programação Segura

Figura: Fuzzing test.



Fonte: Tutorials Point.

Programação Segura

- Realize testes de sessão para verificar se o usuário está autenticado.
- Implemente algum mecanismo de *timeout* por ociosidade sempre que for possível.
- Também quando possível, implemente as opções de autenticação de dois fatores (verificação em duas etapas) e via certificado digital.

Programação Segura

- Implemente um mecanismo que dificulte ataques de força bruta na autenticação.
  - Bloqueio do IP após x tentativas [em t unidades de tempo]
  - Bloqueio da conta
  - (Re)Captcha
- Quando não há regras assim, é fácil realizar um ataque.

Programação Segura

- Configure o arquivo `robots.txt` para páginas de acesso que não devam ser encontradas por motores de busca.
- Não insira meta-tags nestas páginas.
- Utilize diretórios e nomes de arquivos não óbvios.

Programação Segura

- Procure não permitir o uso de senhas fracas.
- Implemente uma forma de recuperação de senhas não trivial.
- Não salve dados sensíveis em texto claro no SGBD. Aprimore esta solução
  - salvando o *hash* do dado sensível;
  - salvando o *hash* do dado sensível concatenado com uma palavra fixa (*salt* fixo);
  - salvando o *hash* do dado sensível concatenado com uma palavra variável (*salt* variável);
  - salvando o *hash* do dado sensível concatenado com *salt* variável e *pepper*;
    - ***Pepper*** seria como um *salt* fixo, mas salvo externamente ao SGBD (código-fonte, variável de ambiente, arquivo de configuração...). Assim, o invasor precisaria ter acesso ao SGBD e ao outro recurso para quebrar o hash.
  - **utilizando `bcrypt`: algoritmo de hash gerado para senhas.**
- Não utilize nomes de colunas amigáveis.

Programação Segura

- **bcrypt**:
  - Algoritmo de *hash* criado especificamente para senhas/dados sensíveis, ou seja, não é um *hash* de propósito geral.
  - Baseado no Blowfish (blowfish-based crypt).
  - Possui como entradas a senha, o *salt* (variável) e o custo.
  - Considerado um algoritmo adaptativo, pode se tornar mais lento (custoso) para evitar ataques de força bruta.
  - `https://bcrypt-generator.com/`
  - `htpasswd -bnBC 10 '' Ricardo | cut -c2-`
    - Experimente aumentar gradativamente o custo e controlar o tempo, executando:

```
for i in $(seq 10 16); do time htpasswd -bnBC $i ''
    Ricardo | cut -c2-; done # ou...
time htpasswd -bnBC 11 '' Ricardo | cut -c2-
time htpasswd -bnBC 12 '' Ricardo | cut -c2-
time htpasswd -bnBC 13 '' Ricardo | cut -c2-
...
```

Programação Segura

- Utilize um sistema de versionamento consistente, mas não compartilhe com quem não deva ter acesso.
- Forneça permissões mínimas, somente a quem precisar. Opere somente com o privilégio necessário.
  - **Exemplo**: um usuário que acessa seu servidor web precisa saber qual é a versão dele?
- Remova imediatamente as permissões assim que for necessário.
- Se não for necessário utilizar o administrador, não utilize.

- Essas diretrizes costumam fazer parte dos **princípios de projeto**.

Programação Segura

- Exemplos de cláusulas para o servidor web Apache:

```
ServerTokens Prod
ServerSignature Off
```

- **ServerTokens**: vem com valor padrão `OS`, exibindo algo como `Server: Apache/2.2.2 (Ubuntu)`. O valor `Prod` exibirá somente `Server: Apache`.
- **ServerSignature**: acrescenta uma assinatura no rodapé da página quando houver documentos gerados pelo servidor, tais como páginas com mensagens de erro. É útil quando uma cadeia de proxies é acessada (que servidor retornou a mensagem?). Por padrão vem com o valor `Off`.

Programação Segura

- **Princípios de projeto**, segundo BISHOP (2020):
  - Princípio do **menor privilégio**: fornecer apenas o privilégio necessário.
  - Princípio de **padrões à prova de falhas**: o acesso ao recurso deve ser negado, exceto se expressamente autorizado.
  - Princípio de **economia do mecanismo**: os mecanismos de segurança devem ser simples.
  - Princípio da **mediação completa**: todo acesso ao recurso deve ser verificado para garantir que ele é permitido.
  - Princípio do **design aberto**: a segurança do recurso não deve depender do sigilo de seu projeto ou implementação.
  - Princípio da **separação de privilégios**: um sistema não deve conceder permissão com base em uma única condição.
  - Princípio do **mecanismo minimamente comum**: mecanismos de acesso a recursos não devem ser compartilhados.
  - Princípio da **aceitabilidade psicológica**: mecanismos de segurança não devem dificultar o acesso ao recurso em comparação ao acesso sem eles.

Programação Segura

- Teste o retorno de funções.
- Utilize funções intrinsecamente seguras.
  - Exemplo de função insegura: `strcpy`
- `strcpy_s` não é!
- Bom exemplo: `https://stackoverflow.com/questions/32136185/difference-between-strcpy-and-strcpy-s`

Programação Segura

- Evite o armazenamento (temporário ou permanente) de dados sensíveis em arquivos.
- Da mesma forma, evite guardar senhas, chaves ou outros dados sensíveis no código, pois é possível obtê-los por meio de engenharia reversa.

Programação Segura

- Assinar o código garante a sua integridade. Visual Studio possui esta opção. A assinatura do código permite identificar se o arquivo foi alterado.
- Permite também protegê-lo por senha.

Programação Segura

- Ofuscação de código o torna menos suscetível à Engenharia Reversa. Menos suscetível. Não é sinônimo de "torna impossível"!
- É a simples escrita do código (fonte ou binário) de um jeito que o torne difícil de ser entendido.

Programação Segura

- Ferramentas de ofuscação:
  - ConfuserEx
  - Crypto Obfuscator
  - Agile.NET
  - Babel Obfuscator
  - ...

Programação Segura

Figura: Humor sobre ofuscação de código.

Fonte:
`http://vidadeprogramador.com.br/2011/07/20/codigo-ofuscado/`

Programação Segura

Métricas

- Número de bugs arrumados
- Número de vezes que é citado no CERT.br
- Tempo entre a descoberta de bugs
- Tempo necessário para encontrar um bug
- Tempo médio para consertar um bug
- Porcentagem do orçamento da área de TI gasto com segurança da informação
- ...

Programação Segura

Leitura Complementar

- Exemplo de ciclo de vida de desenvolvimento seguro no livro **Segurança e Auditoria em Sistemas de Informação**, de Maurício Rocha Lyra.

Referências

- BISHOP, Matt. **Computer Security: Art and Science.** Addison-Wesley, 2002. Disponível em: `http://nob.cs.ucdavis.edu/book/book-aands/aands13.pdf`. Acesso em: 16 fev. 2020.
- LYRA, Maurício Rocha. **Segurança e Auditoria em Sistemas de Informação**. Rio de Janeiro: Editora Ciência Moderna. 2008.
- ZARPELÃO, Bruno Bogaz. Métricas de segurança. In: **Revista Infra Magazine 9.** [s. d.]. Disponível em: <`http://www.devmedia.com.br/metricas-de-seguranca-revista-infra-magazine-9/26786`>. Acesso em: 26 fev. 2016.

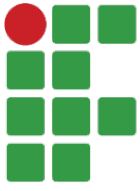
Exercício

1. Pesquise e explique alguma métrica de segurança da informação.
2. Crie um código simples em Java. Procure um descompilador Java e verifique se a ferramenta consegue obter o seu código-fonte a partir do arquivo `.class`.

Programação Segura

Segurança Computacional

Ricardo de la Rocha Ladeira
{ricardo.ladeira@ifc.edu.br}



INSTITUTO FEDERAL
Catarinense
Campus Blumenau