



Ferramentas e técnicas de *Security*

João de Macedo
Nelson Gonçalves
João Aloísio



Introdução

Qualidade de *software*:

1. Número de defeitos encontrados após o lançamento.
2. Gravidade desses defeitos.
3. Esforço necessário para resolver esses mesmos defeitos.



Indicador de qualidade de software Segurança

- O que é?
- Para que serve?
 - Número de vulnerabilidades;
 - Tempo de resolução;
 - Número de incidente e a sua gravidade;



Ferramentas utilizadas

- Static Application Security Testing (SAST).
- Dynamic Analysis Security (DAST).
- Software Composition Analysis (SCA).



Exemplos de ferramentas

- Commercial Tools
 - Checkmarx SAST(CxSAST);
 - Veracode;
- Open Source or Free Tools
 - Flawfinder;
 - Brakeman;



Checkmarx SAST(CxSAST)

- Ferramenta SAST;
- Suporta cerca de 25 linguagens de programação;
- Scan gera relatórios de segurança/interface interativa;
- O código não necessita de compilar corretamente;



Checkmarx SAST(CxSAST)

Pontos únicos da ferramenta:

- Encontra as vulnerabilidades antecipadamente;
- Rápida correção;
- Facilidade de automação;



Veracode

- Veracode é uma ferramenta SAST mas também apresenta soluções para os outros tipos de ferramentas, DAST e SCA, e ainda teste manual de intrusão.
- Esta disponibiliza uma dashboard onde se pode observar o estado da aplicação
- Inclui uma API que pode ser adaptada para o software.
- Fornece também de dicas para solucionar as vulnerabilidades que encontra
- Suporta diversas linguagens como, Java, Python, C, entre outra



Veracode


Aspectos sobre a aplicação:

- Security Feedback While Coding:
 - Na escrita de código, o scan do IDE providência feed-back da segurança
 - Ajuda na rápida correção do código
 - Oferece exemplos de código, guias e links diretos para tutoriais da aplicação
- High Accuracy Without Tuning :
 - A aplicação tem uma taxa menor que 1,1% de falsos-positivos
- Focus On Fixing :
 - Segundo Veracode, para além de encontrar, apresenta soluções em que os utilizadores têm uma taxa cerca de 70% de solucionar os problemas.
 - É providenciado dicas, guias e tutoriais dos problemas encontrados.



Flawfinder

- Ferramenta open source;
- Para linguagem de programação C/C++;
- Produz uma lista com possíveis falhas de segurança;



```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    char *dummy = (char *) malloc (sizeof(char) * 10);
    char *readonly = (char *) malloc (sizeof(char) * 10);

    strcpy(readonly, "laranjas");
    strcpy(dummy, argv[1]);
    printf("%s\n", readonly);
}
```



```
flawfinder --html --context overflowHeap.1.c > results.html
```

Flawfinder Results

Here are the security scan results from [Flawfinder version 2.0.11](#), (C) 2001-2019 [David A. Wheeler](#). Number of rules (primarily dangerous function names) in C/C++ ruleset: 223

- overflowHeap.1.c:10: **[4]** (buffer) *strcpy*: Does not check for buffer overflows when copying to destination [MS-banned] ([CWE-120](#)). Consider using *snprintf*, *strcpy_s*, or *strncpy* (warning: *strncpy* easily misused).

```
strcpy(dummy, argv[1]);
```

- overflowHeap.1.c:9: **[2]** (buffer) *strcpy*: Does not check for buffer overflows when copying to destination [MS-banned] ([CWE-120](#)). Consider using *snprintf*, *strcpy_s*, or *strncpy* (warning: *strncpy* easily misused). Risk is low because the source is a constant string.

```
strcpy(readonly, "laranjas");
```

Analysis Summary

Hits = 2

Lines analyzed = 12 in approximately 0.01 seconds (1491 lines/second)

Physical Source Lines of Code (SLOC) = 10

Hits@level = [0] 1 [1] 0 [2] 1 [3] 0 [4] 1 [5] 0

Hits@level+ = [0+] 3 [1+] 2 [2+] 2 [3+] 1 [4+] 1 [5+] 0

Hits/KSLOC@level+ = [0+] 300 [1+] 200 [2+] 200 [3+] 100 [4+] 100 [5+] 0

Minimum risk level = 1

Not every hit is necessarily a security vulnerability.

There may be other security vulnerabilities; review your code!

See '[Secure Programming HOWTO](#)' (<https://dwheeler.com/secure-programs>) for more information.



Analysis Summary

Hits = 36

Lines analyzed = 117 in approximately 0.02 seconds (6559 lines/second)

Physical Source Lines of Code (SLOC) = 80

Hits@level = [0] 16 [1] 9 [2] 7 [3] 3 [4] 10 [5] 7

Hits@level+ = [0+] 52 [1+] 36 [2+] 27 [3+] 20 [4+] 17 [5+] 7

Hits/KSLOC@level+ = [0+] 650 [1+] 450 [2+] 337.5 [3+] 250 [4+] 212.5 [5+] 87.5

Suppressed hits = 2 (use --neverignore to show them)

Minimum risk level = 1

Not every hit is necessarily a security vulnerability.

There may be other security vulnerabilities; review your code!

See '[Secure Programming HOWTO](https://dwheeler.com/secure-programs)' (<https://dwheeler.com/secure-programs>) for more information.

- test.c:32: **[5]** (buffer) *gets*: Does not check for buffer overflows ([CWE-120](#), [CWE-20](#)). Use *fgets()* instead.

```
gets(f);
```

- test.c:56: **[5]** (buffer) *strncat*: Easily used incorrectly (e.g., incorrectly computing the correct maximum size to add) [MS-banned] ([CWE-120](#)). Consider *strcat_s*, *strlcat*, *snprintf*, or automatically resizing strings. Risk is high; the length parameter appears to be a constant, instead of computing the number of characters left.

```
strncat(d,s,sizeof(d)); /* Misuse - this should be flagged as riskier. */
```

- test.c:57: **[5]** (buffer) *_tcsncat*: Easily used incorrectly (e.g., incorrectly computing the correct maximum size to add) [MS-banned] ([CWE-120](#)). Consider *strcat_s*, *strlcat*, or automatically resizing strings. Risk is high; the length parameter appears to be a constant, instead of computing the number of characters left.

```
_tcsncat(d,s,sizeof(d)); /* Misuse - flag as riskier */
```

- test.c:60: **[5]** (buffer) *MultiByteToWideChar*: Requires maximum length in *CHARACTERS*, not bytes ([CWE-120](#)). Risk is high, it appears that the size is given as bytes, but the function requires size as characters.

```
MultiByteToWideChar(CP_ACP,0,szName,-1,wszUserName,sizeof(wszUserName));
```



Brakeman

- Brakeman é um scanner de segurança para aplicações Ruby on Rails que analisa o código fonte da aplicação.
- Depois verificar o código da aplicação, produz um relatório de todos os problemas de segurança encontrados



Brakeman

Vantagens:

- Não requer instalação ou configuração, basta correr
- Como os requisitos do Brakeman é o código-fonte, o Brakeman pode ser executado em qualquer fase do desenvolvimento
- O Brakeman foi desenvolvido especificamente para aplicações Ruby on Rails, pode verificar facilmente as definições de configuração para obter as melhores práticas



Brakeman

Limitações:

- Apenas quem desenvolve a aplicação pode entender se determinados valores/pontos são perigosos ou não. Por default, Brakeman é extremamente suspeito, isto pode levar a muitos “falsos positivos”
- Ao contrário de outros scanners de vulnerabilidade dinâmica que podem testar o servidor web e a base de dados o Brakeman não relata se um servidor web ou outro software tem problemas de segurança.

== Brakeman Report ==

Application Path: /Users/jcollin/work/brakeman/test/apps/rails5

Rails Version: 5.0.0

Brakeman Version: 4.3.1

Scan Date: 2018-09-24 12:13:04 -0700

Duration: 0.6386 seconds

Checks Run: BasicAuth, BasicAuthTimingAttack, ContentTag, CreateWith, CrossSiteScripting, DefaultRoutes, Deserializing, ForgerySetting, HeaderDoS, I18nXSS, JRubyXML, JSONEncoding, JSONParsing, LinkTo, LinkToHref, MailTo, MassAssignment, PermitAttributes, QuoteTableName, Redirect, RegexpDoS, Render, RenderDoS, RenderInline, ResponseSplitFile, SessionManipulation, SessionSettings, SimpleFormat, SingleQuotes, SkipBeforeFilter, SprocketsPathTraversal,

== Overview ==

Controllers: 5

Models: 3

Templates: 17

Errors: 0

Security Warnings: 27

== Warning Types ==

Cross-Site Scripting: 17

Dangerous Eval: 2

Dangerous Send: 1

Mass Assignment: 2

Path Traversal: 1

Redirect: 1

Remote Code Execution: 1

SQL Injection: 2

== Warnings ==

Confidence: High

Category: Cross-Site Scripting

Check: LinkToHref

Message: Unsafe parameter value in `link_to` href

Code: link_to("xss", url_for(params[:bad]))

File: app/views/users/show.html.erb

Line: 7

Confidence: High

Category: Cross-Site Scripting

Check: CrossSiteScripting

Message: Unescaped parameter value

Code: strip_tags(params[:x])

File: app/views/users/sanitizing.html.erb

Line: 3

Confidence: High

Category: Cross-Site Scripting

Check: CrossSiteScripting



Conclusão

- Atualmente a segurança de um produto software é essencial dado que o utilizador fornece informações confidenciais/sensíveis.
- Para uma empresa/produto quanto mais cedo uma falha de segurança for detectada, mais rapidamente esta irá ser corrigida tendo assim menos custos e um impacto mais reduzido
- Podemos recorrer ao uso de ferramentas existentes que previne essas falhas ou que as alerte, podendo até fornecer uma solução para as mesmas em diferentes etapas do desenvolvimento de software
- É importante aplicar boas práticas de desenvolvimento de software seguro tendo em conta que nenhuma ferramenta é infalível mas são mais uma ajuda para obter um código mais seguro.