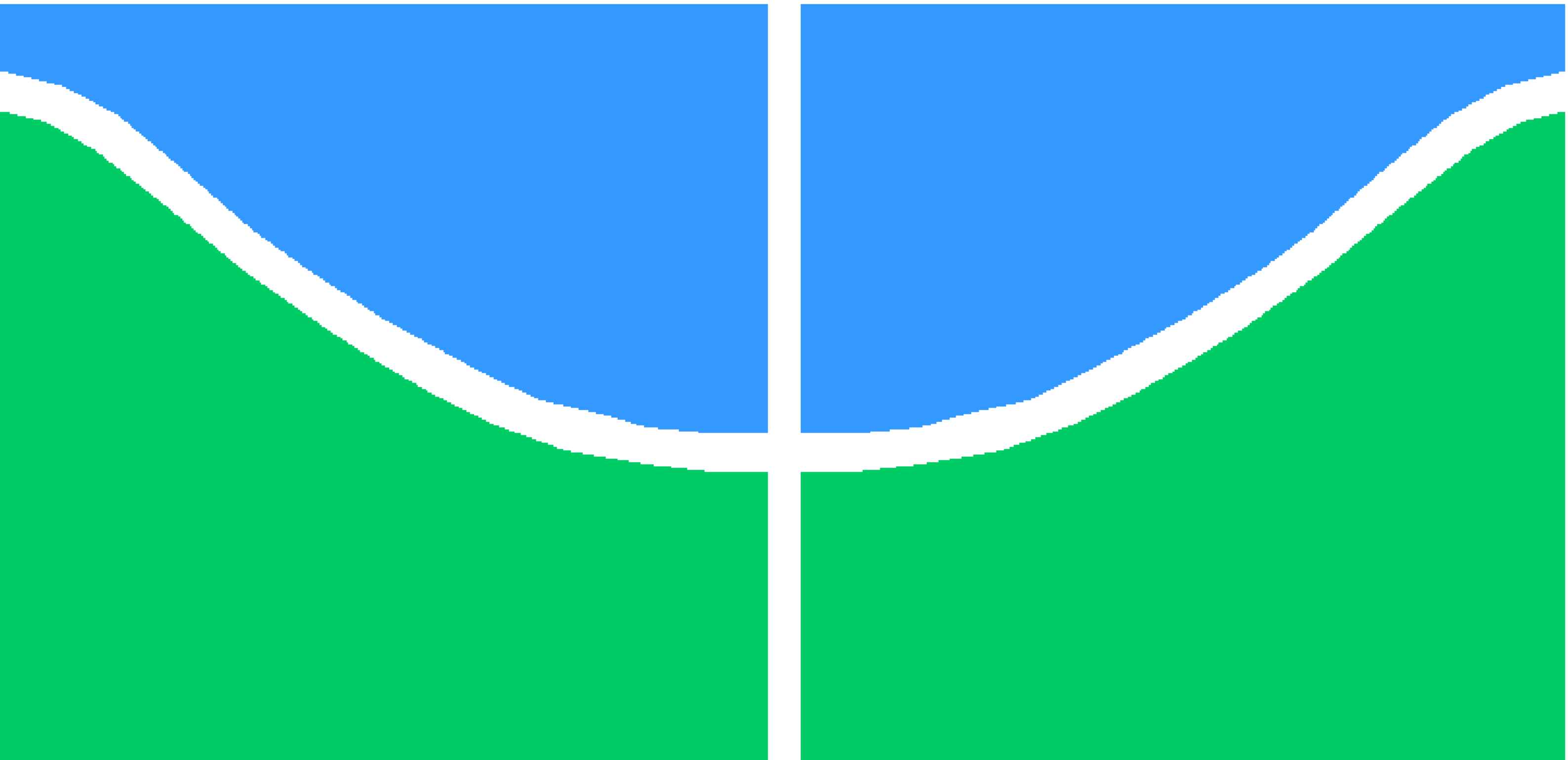

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Criação de um Guia de Boas Práticas para Desenvolvimento Seguro

Autor: Euler Tiago Rodrigues de Carvalho

Orientador: Prof. Dr. Tiago Alves da Fonseca

Brasília, DF

2017

Euler Tiago Rodrigues de Carvalho

Criação de um Guia de Boas Práticas para Desenvolvimento Seguro

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Tiago Alves da Fonseca

Brasília, DF

2017

Euler Tiago Rodrigues de Carvalho

Criação de um Guia de Boas Práticas para Desenvolvimento Seguro/ Euler Tiago Rodrigues de Carvalho. – Brasília, DF, 2017-

103 pp. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Tiago Alves da Fonseca

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2017.

1. Segurança da Informacão. 2. Engenharia de Software. I. Prof. Dr. Tiago Alves da Fonseca. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Criação de um Guia de Boas Práticas para Desenvolvimento Seguro

CDU 02:141:005.6

Euler Tiago Rodrigues de Carvalho

Criação de um Guia de Boas Práticas para Desenvolvimento Seguro

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 07 de Julho de 2017:

Prof. Dr. Tiago Alves da Fonseca
Orientador

Prof. Dr. Fernando William Cruz
Convidado 1

Prof. MSc. Renato Coral Sampaio
Convidado 2

Brasília, DF
2017

Dedico este trabalho aos meus pais,
Amélia Gonçalves da Silva e José Rodrigues de Carvalho,
e a minha irmã Eulália Priscila Gonçalves de Carvalho.

Agradecimentos

Agradeço a Deus, primeiramente, que me deu forças e sabedoria para alcançar meus objetivos, a minha família, pois me deu apoio nas horas boas e ruins sempre compreendendo meu esforço e não me deixando abater. A minha namorada que me incentivou e entendeu que eu precisava dedicar certos momentos ao meu curso. Ao meu orientador Prof. Dr. Tiago Fonseca pelos textos, orientação, disponibilidade e auxílio sempre que necessário. A Universidade de Brasília - Faculdade Gama e aos professores que durante esses longos anos me proporcionaram experiências incríveis e inesquecíveis contribuindo bastante para a minha formação pessoal e profissional.

"They did not know it was impossible,
so they did it!
(Mark Twain)

Resumo

O desenvolvimento de software tem crescido exponencialmente. Porém como é uma área muito nova e está em constante evolução, novos tópicos surgem a cada dia. Um dos aspectos mais críticos no que diz respeito ao desenvolvimento de software é a preocupação com a segurança da informação, principalmente com o advento da Internet, porque uma vez conectada à rede, não é possível delimitar a fronteira do sistema computacional inviabilizando qualquer estratégia tradicional de segurança sistêmica.

Se o programador tiver em mente a preocupação com a disponibilização de contramedidas de segurança desde a concepção do sistema computacional, é possível mitigar alguns problemas clássicos de segurança de maneira eficiente. Medidas preventivas, em geral, tendem a ser mais baratas do que medidas corretivas.

Este trabalho apresenta um breve estudo sobre as principais plataformas da computação, sejam tradicionais ou móveis. Abordando conceitos de Sistemas Operacionais, Linguagens de Programação e problemas de segurança clássicos, bem como as medidas necessárias para resolvê-los. Lançando mão de um *Quizz*, espera-se contribuir com a formação de programadores mais conscientes das responsabilidades afetas à segurança da informação.

Palavras-chaves: Desenvolvimento Seguro de Software. Segurança da Informação. Sistemas Operacionais.

Abstract

Software development has grown exponentially. However it is a very new area and is constantly evolving, new topics come up every day. One of the most critical aspects with regard to software development is the concern with information security, Especially with the advent of the Internet, because it is connected to the network, it is not possible to delimit the frontier of the computational system, rendering any traditional systemic security strategy impossible.

If the programmer has in mind the concern about security countermeasures from the computer system, it's possible to mitigate some classic security problems in an efficient way. Preventive measures, in general, tend to be cheaper than corrective measures.

This paper presents a brief study on the main computing platforms, whether traditional or mobile. Addressing concepts of Operating Systems, Programming Languages and classic security problems, as well as the necessary measures to solve them. We also developed a Quizz to help programmers be aware of information security issues.

Keywords: Secure Development. Information Security. Operational Systems. Quizz.

Lista de ilustrações

Figura 1 – Linguagens de Programação que são tendências no Github. 34
Figura 2 – Compilar Código Fonte em C. 36
Figura 3 – Compilar Código Fonte em Java. 36
Figura 4 – Adaptação do MVC utilizada para a aplicação. 56
Figura 5 – Resultados obtidos após uma rodada de perguntas. 59
Figura 6 – Gráfico com os dados coletados. 60
Figura 7 – Camadas do iOS. 96
Figura 8 – Representação do padrão de projeto MVC. 100
Figura 9 – Pseudo código para implementação de uma *UITableView*. 100
Figura 10 – Eventos na *Main Thread*. 101

Lista de tabelas

Tabela 1 – Características de acordo com cada geração de rede móvel. 40
Tabela 2 – Formas inseguras de envio e recebimento dos dados entre componentes, módulos, programas, processos, etc. 51
Tabela 3 – O Software não gerencia adequadamente a criação, uso, transferência ou destruição de recursos do sistema. 51
Tabela 4 – Técnicas defensivas mal configuradas que podem ser burladas ou ignoradas. 51
Tabela 5 – Resultados obtidos na metodologia completa 60
Tabela 6 – Dispositivos e versões de software. 95
Tabela 7 – Características entre as versões do iOS. 99

Lista de abreviaturas e siglas

AMPS	*Advanced Mobile Phone System*
AP	*Access Point*
BAN	*Body Area Network*
BSD	*Berkley Software Distribution*
CDMA	*Code Division Multiple Access*
CWE	*Commom Weakness Enumeration*
FHSS	*Frequency-hopping Spread Spectrum*
FSF	*Free Software Foundation*
GSM	*Global System for Mobile Communications*
HCI	*Host Controller Interface*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
iOS	Sistema Operacional Móvel da Apple Inc.
ISM	*Industrial Sientific and Medical*
JVm	*Java Virtual Machine*
MAC	*Media Access Control Address*
MVC	*Model-View-Controller*
NIST	*National Institute of Standards and Technology*
NT	*New Tecnology*
LAN	*Local Area Network*
L2CAP	*Logical Link Control and Adaptation Protocol*
OWASP	*Open Web Application Security Project*
PDA	*Personal Digital Assistant*

RFCOMM *Radio Frequency Communication*

SDP *Service Discovery Protocol*

SIG *Bluetooth Special Interest Group*

SO Sistema Operacional

SSID *Service Set IDentifier*

TDMA *Time Division Multiple Access*

VWAN *Very Wide Area Network*

WAN *Wide Area Network*

WEP *Wired-Equivalent Privacy Protocol*

WPA *Wi-Fi Protected Access*

WWDC *Apple Worldwide Developers Conference*

WWW *World Wide Web*

Lista de símbolos

US$	Moeda corrente nos Estados Unidos da América (Dólar)

Sumário

1 INTRODUÇÃO . . . 27
1.1 Contextualização e Justificativa . . . 27
1.2 Objetivos . . . 28
1.2.1 Objetivo Geral . . . 28
1.2.2 Objetivos Específicos . . . 29
1.2.3 Estrutura do Trabalho . . . 29

2 PLATAFORMAS TRADICIONAIS . . . 31
2.1 Breve resumo . . . 31
2.2 Sistemas Operacionais . . . 31
2.2.1 Conceito . . . 31
2.2.2 Unix-Like . . . 31
2.2.3 MS Windows . . . 32
2.3 Linguagens de Programação . . . 33
2.3.1 C/C++ . . . 35
2.3.2 Java . . . 35
2.3.2.1 Garbage Collector . . . 37
2.3.3 Python . . . 38

3 PLATAFORMAS MÓVEIS . . . 39
3.1 Mobilidade e Recursos . . . 39
3.1.1 Aplicações Móveis . . . 39
3.1.2 Redes Móveis . . . 39
3.1.3 Redes Sem Fio – Wi-Fi . . . 40
3.1.4 Bluetooth . . . 40
3.1.4.1 Serviços Tradicionais de Segurança no Bluetooth . . . 41
3.1.5 Geolocalização . . . 41
3.1.5.1 Riscos relacionados à Geolocalização . . . 42
3.1.5.2 Boas Práticas relacionadas à Geolocalização . . . 42

4 DESENVOLVIMENTO SEGURO . . . 43
4.1 Princípios de Segurança . . . 43
4.2 Segurança da Informação e Problemas Clássicos . . . 44
4.2.1 Armazenamento de Dados . . . 45
4.2.2 Divulgação de Informações . . . 45
4.2.3 Fraude Eletrônica (*Phishing*) . . . 45

4.2.4 Ruptura de Isolamento de Aplicações . . . 46
4.2.5 Overflow . . . 46
4.2.5.1 *Buffer Overflow* . . . 47
4.2.5.2 Integer Overflow . . . 48
4.2.6 Ataques a *Parsers* - Zip Bomb . . . 48
4.2.7 Vulnerabilização de Privacidade por Serviços de Localização . . . 49
4.2.8 Vulnerabilização de Segurança Física . . . 49
4.2.9 Sistemas Operacionais Vulneráveis . . . 50
4.2.10 *Malwares: Vírus, Worms, Trojans e Spyware* . . . 50
4.3 Erros Comuns no Desenvolvimento de Software . . . 50
4.4 Boas Práticas no Desenvolvimento Seguro . . . 52
4.4.1 Princípios da Arquitetura Segura . . . 52
4.4.2 Seguir Práticas de Programação Segura . . . 52
4.4.3 Validar as Entradas de Informações . . . 53
4.4.4 Utilize os Privilégios Mínimos Necessários . . . 53
4.4.5 Armazene Informações Corretamente . . . 53
4.4.6 Evite Ameaças Comuns . . . 54
4.4.7 Separação de Responsabilidades . . . 54
4.4.8 Defesa em Profundidade . . . 54
4.4.9 Falha Segura . . . 54
4.4.10 Evitar Segurança por Obscuridade . . . 54
4.4.11 Isolamento de Entidades Não Confiáveis . . . 54

5 O QUIZZ . . . 55
5.1 Breve resumo . . . 55
5.2 Como Funciona . . . 55
5.3 Aspectos Técnicos . . . 55
5.3.1 Tecnologias . . . 55
5.3.2 Aplicativo iOS . . . 55
5.3.2.1 Arquitetura . . . 55
5.3.2.2 Banco de Dados . . . 56
5.3.3 API . . . 57
5.3.3.1 Breve Descrição . . . 57
5.3.3.2 Arquitetura . . . 57
5.3.4 Links Úteis . . . 57

6 ANÁLISE DOS RESULTADOS . . . 59
6.1 Breve Resumo . . . 59
6.2 Resultados Obtidos . . . 59
6.2.1 Participantes que concluíram toda a metodologia proposta . . . 60

6.3 Limitações . . . 60

7 CONCLUSÃO . . . 63
7.1 Considerações Finais . . . 63
7.2 Limitações do Trabalho . . . 64
7.3 Trabalhos Futuros . . . 64

REFERÊNCIAS . . . 65

APÊNDICES 71

APÊNDICE A – QUESTIONÁRIO . . . 73
A.1 Metodologia Utilizada . . . 73
A.2 Perguntas e Respostas . . . 73
A.2.1 Conceitos sobre conhecimentos gerais . . . 73
A.2.2 C, C++ . . . 80
A.2.3 Python . . . 86
A.2.4 Java . . . 91

APÊNDICE B – DISPOSITIVOS APPLE . . . 95
B.0.1 Visão Geral . . . 95
B.0.2 iOS: Sistema Operacional dos Dispositivos Móveis da Apple . . . 95
B.0.2.1 Arquitetura iOS . . . 96
B.0.3 Vulnerabilidades do Sistema iOS . . . 97
B.0.4 Backup de Dados . . . 98
B.0.5 Recuperação de Dados . . . 98
B.0.6 Características Relevantes das versões do iOS . . . 98
B.0.7 Aplicações . . . 99
B.0.7.1 Estrutura de uma Aplicação . . . 101
B.0.7.2 Como a aplicação se comporta com o SO . . . 101
B.0.7.3 Como a aplicação se comporta com outras aplicações . . . 102
B.0.8 Desenvolvimento para Dispositivos iOS . . . 102
B.0.9 Recursos e Limitações para Desenvolvedores iOS . . . 103

1 Introdução

1.1 Contextualização e Justificativa

A necessidade de ter um cuidado relativo à segurança é cada vez maior. Em (GOMES et al., 2014) o Serpro aponta que a maior parte das vulnerabilidades ocorre na camada da aplicação e pode comprometer até as permissões de acesso do sistema por parte dos usuários. Os Sistemas Operacionais, seja para plataformas tradicionais ou plataformas móveis, possuem alguns mecanismos de segurança capazes de prover garantias mínimas de consistência do sistema computacional. O mesmo acontece com as Linguagens de Programação, o problema é que esses mecanismos muitas vezes são deixados de lado, seja por tempo, custo ou até mesmo falta de habilidade do programador.

Em (ADMINISTRATION; SECURITY, 2009), a SANS diz que a quantidade de vulnerabilidades descobertas em aplicações é muito maior do que as vulnerabilidades encontradas nos Sistemas Operacionais, e segundo o (GOMES et al., 2014), isso se deve ao fato de que os sistemas não são construídos com base nas boas práticas de programação e que algumas organizações não seguem um processo de desenvolvimento seguro. Um ótimo exemplo foi o que aconteceu com a Microsoft: por prover o Sistemas Operacionais mais utilizados e, também, por tentar facilitar ao máximo a sua utilização por parte dos usuários, o Windows ao longo da sua história sofreu bastante com ataques cibernéticos. Howard em (HOWARD; LIPNER, 2006), diz que após pressões por parte dos usuários, principalmente do Governo Americano, a Microsoft propôs um Ciclo de Vida de Desenvolvimento de Software Seguro para mitigar intervenções adversas em seus produtos, otimizando, desta forma, tempo e dinheiro para garantir maior confiabilidade no sistema.

Apesar de tomar todas as possíveis medidas de segurança, não é possível afirmar que um ecossistema computacional está realmente seguro se a segurança física não for garantida. Um ótimo exemplo disso é um Sistema Operacional baseado em Debian chamado de "*Bash Bunny*". Segundo (GRIMES, 2017) é necessário apenas conectar um *pendrive* que contém esse Sistema Operacional e alguns outros *scripts* para contornar as proteções na camada da aplicação. O funcionamento é bem simples: ao conectar o *Bash Bunny* através de uma porta USB, o sistema operacional o reconhece como um dispositivo confiável e consegue, ainda, acesso *root*: torna-se possível executar os *scripts* e programas que desejar.

Um aspecto importante ao se falar de segurança é que a utilização de dispositivos móveis tem se tornado algo habitual. Então a preocupação que antes existia para os computadores tradicionais deve continuar nesta nova plataforma, que, por sua vez, impõe

um novo conjunto de desafios afetos às novas funcionalidades acompanhantes da mobilidade. A cada dia que passa, surgem aparelhos com novos recursos e melhorias, deixando não só os dispositivos decadentes como as técnicas de segurança muitas vezes obsoletos. Segundo (DINO, 2016), as pessoas utilizam mais o celular do que o computador pessoal, seja por motivos de trabalho ou entretenimento. Segundo a mesma pesquisa, em média, os donos de *smartphones* possuem 15 aplicativos instalados. Esse crescimento é uma ótima oportunidade para desenvolvedores entrarem em um mercado novo que tem bastante potencial, isso considerando que, estatisticamente, o mercado cresce bem mais do que o de computadores pessoais. Helton diz em sua pesquisa (GOMES, 2016), que os clientes de bancos estão utilizando cada vez mais os *smartphones* para realizar operações bancárias, e segundo essa pesquisa (ROHR, 2015), a maioria dos aplicativos bancários apresentam problemas de segurança.

Katie Young diz em sua pesquisa (YOUNG, 2015), que o tempo gasto *on-line* em um dispositivo móvel triplicou entre os anos de 2012 e 2015, os aplicativos presentes nesses dispositivos podem facilitar bastante a vida dos usuários, mas também podem representar riscos. Informações pessoais, dados de lugares visitados, telefones, mensagens, etc, o *smartphone* é uma fonte de informações prontas para serem roubadas. Fica claro, dessa forma, que a utilização de dispositivos móveis vem crescendo de forma impressionante e a preocupação com segurança da informação deve acompanhar essa evolução, e tanto desenvolvedores quanto usuários podem contribuir com isso. Conhecer todos os recursos, limitações e capacidades disponíveis é essencial para melhorar aspectos de segurança da informação.

Tendo em vista todos esses aspectos referentes à segurança, além de conhecer as falhas mais comuns presentes em aplicações e sistema operacionais, é necessário também conhecer a tecnologia e os aspectos de segurança afetos à plataforma utilizada, para que seja possível criar mecanismos garantindo a segurança das informações manipuladas pelo sistema. Como nem todos os programadores conseguem medir o quanto são preocupados com segurança, esse trabalho propõe uma discussão sobre esses temas e sugere uma estratégia para averiguar o nível de aderência de cada programador aos princípios de segurança. Um dos produtos dessa monografia foi a concepção de um *Quizz* que tenta avaliar o perfil do programador, de forma a indicar os pontos fortes e fracos e ao mesmo tempo, incentivar à utilização dessas boas práticas.

1.2 Objetivos

1.2.1 Objetivo Geral

Dentro deste contexto, o presente trabalho apresenta uma sugestão boas práticas de programação defensiva.

Para o cumprimento do objetivo geral, foram traçados alguns objetivos específicos.

1.2.2 Objetivos Específicos

- Levantamento sobre a história da computação e sua evolução buscando entender a dinâmica relacionada à segurança da informação.
- Estudo sobre alguns pontos específicos de sistemas operacionais mais utilizados, tais como, linguagens de programação e aspectos relativos à segurança.
- Levantamento sobre a história da computação móvel e sua evolução buscando entender a dinâmica relacionada à segurança da informação.
- Levantamento sobre as tecnologias e recursos existentes em dispositivos móveis a fim de encontrar quais são as limitações e vulnerabilidades de cada tecnologia.
- Levantamento sobre a história dos dispositivos Apple e como as aplicações se relacionam tanto com o sistema operacional quanto com outras aplicações.
- Estudo de problemas e vulnerabilidades conhecidas, buscando aprender com problemas antigos recorrentes e que representam riscos.
- Levantamento de boas práticas comumente utilizadas em desenvolvimento seguro, principalmente no que diz respeito à linguagens de programação.
- Buscar os insumos necessários para a criação de um aplicativo do tipo *Quizz*, que teste e ao mesmo tempo incentive a adoção de práticas de programação defensiva durante o desenvolvimento de aplicações.
- Apresentação das práticas de programação defensiva atraves de uma forma lúdica em um aplicativo do tipo "*Quizz*" com o objetivo de testar as habilidades dos desenvolvedores no que diz respeito à utilização de técnicas que visam a confiabilidade e integridade do sistema.

1.2.3 Estrutura do Trabalho

Este documento está dividido em 7 capítulos. No Capítulo 2 é apresentado um breve resumo sobre Plataformas Tradicionais mostrando algumas características dos Sistemas Operacionais mais utilizados juntamente com 3 linguagens de programação e alguns aspectos de segurança e convenção de linguagem. O Capítulo 3 apresenta os principais recursos das Plataformas Móveis, apontando o quão difícil é prover segurança nessas plataformas. No Capítulo 4 são apresentados os Princípios de Segurança, Erros comuns no Desenvolvimento de Software e algumas Boas Práticas de Programação. No Capítulo 5 são apresentadas as questões técnicas do *Quizz* que foi proposto tais como: Arquitetura,

Linguagem de Programação, Hospedagem e Banco de Dados. Por fim, são apresentadas as Considerações Finais bem como Conclusão e Trabalhos Futuros propostos para a continuidade deste estudo que está sempre em evolução.

2 Plataformas Tradicionais

2.1 Breve resumo

Este capítulo mostra os principais conceitos que envolvem as plataformas tradicionais, passando desde a sua história, abordando os principais sistemas operacionais e linguagens de programação utilizadas, até alguns problemas clássicos e contramedidas de segurança.

2.2 Sistemas Operacionais

2.2.1 Conceito

Em (TANENBAUM; WOODHULL, 2009), Tanembaum define a tarefa do Sistema Operacional como:

> Controlar todos os recursos do computador e fornecer uma base sobre a qual os programas aplicativos podem ser escritos.

O mesmo autor diz que, um computador sem o software é apenas um monte inútil de metal. Com software um computador pode armazenar, processar e recuperar informações, etc. A grosso modo, o software de computador pode ser dividido em dois tipos: programas de sistema e programas aplicativos.

- **Programas de Sistema:** Gerenciam a operação do computador em si;
- **Programas Aplicativos:** Realizam o trabalho real desejado pelo usuário.

2.2.2 Unix-Like

No final da década de 70, Kenneth Thompson, Dennis Ritchie e outros da AT&T Bell Labs começaram a desenvolver um pequeno sistema operacional em um PDP-7 pouco usado. Alguns anos depois, o sistema foi reescrito utilizando a Linguagem C e trazendo uma série de inovações. Foi então que a Berkley juntamente com a comunidade acadêmica criou o BSD e, a partir de então, surgiu uma competição entre ambos sistemas: conforme uma inovação era lançada em uma das plataformas, ela também era utilizada na outra. Essa situação resultou em várias versões do Unix, algumas delas proprietárias. Com isso, em 1984, Stallman criou uma iniciativa chamada *Free Software Foundation* (FSF) que era um projeto para desenvolver uma versão livre do Unix. Essa iniciativa possui alguns

princípios, por exemplo, o software é livre pra ser usado, lido, modificado e redistribuído. Porém estavam tendo dificuldade para criar um kernel foi então que em 1991, Linus Torvalds começou a desenvolver um kernel chamado “Linux”.

Algumas empresas passaram então a criar suas próprias distribuições do Linux. Algumas das mais famosas são RedHat, Solaris, Mandrake, Debian, etc. Existem diferenças entre as várias distribuições, mas todas elas possuem a mesma base: o kernel do Linux e as bibliotecas GNU glibc. Uma vez que ambos são cobertos por licenças de estilo “*copyleft*”[1], as alterações dessas empresas geralmente devem ser disponibilizadas a todos, uma força unificadora entre as distribuições Linux em sua fundação que não existe entre o BSD e os sistemas Unix derivados da AT&T.

Sendo assim, esses sistemas “Unix-Like” compartilham uma série de mecanismos, dentre os quais alguns de segurança. Em geral, todas essas distribuições utilizam conceitos de “uid” e “gid” para cada processo e um sistema de arquivos com permissões de leitura, escrita e execução, processos, conexão de internet e *sockets*, hierarquia de arquivos de sistema, POSIX, ACL’s, DLL, Sinais, etc.

2.2.3 MS Windows

Silberchatz em seu livro (SILBERSCHATZ; GALVIN; GAGNE, 2013), diz que no início da década de 80, a Microsoft e a IBM formaram uma parceria para desenvolver um sistema operacional chamado OS/2, que a princípio, foi escrito para atender apenas os sistemas Intel 80286 de processador único. Em 1988, a Microsoft decidiu encerrar o esforço em conjunto com a IBM para desenvolver seu próprio sistema operacional portátil chamado de NT para dar suporte tanto as interfaces de programação de aplicativos (APIs) OS/2 quanto “POSIX”.

O mesmo autor afirma que durante o desenvolvimento deste sistema operacional, a API OS/2 foi substituída pela API de 32 bits do Windows. As primeiras versões do NT eram 3.1 e 3.1 *Advanced Server*. Na versão 4.0, o Windows NT adotou a interface de usuário do Windows 95 e incorporou o servidor de Internet e o software de navegador da web. Além disso, rotinas de interface do usuário e todo o código de gráficos foram movidos para o *kernel* tentando melhorar o desempenho do sistema. O Windows 2000 incorporou alterações significativas, principalmente o *Active Directory* (um serviço de diretório baseado em X.500), forneceu suporte para redes e laptop, para dispositivos *plug-and-play*, mais processadores e memória.

Em outubro de 2001, o Windows XP foi lançado como uma atualização para o Windows 2000, sistema operacional de *desktop*, e uma substituição para Windows 95/98. Em 2002, a edição de servidor do Windows XP tornou-se disponível (chamado Windows

[1] Copyleft é um método geral para tornar um programa livre e exigir que todas as versões modificadas e extensões do programa também sejam livres. (FOUNDATION, 2017)

.Net Server). O Windows XP atualizou a interface gráfica do usuário (GUI) com um *design* visual que aproveitou avanços de hardware mais recentes e muitos novos recursos de facilidade de uso. Vários recursos foram adicionados para reparar automaticamente problemas tanto em aplicativos quanto no próprio sistema operacional. Como resultado dessas mudanças, o Windows XP forneceu melhor experiência em redes e dispositivos (incluindo configuração de rede sem fio, mensagens instantâneas, mídia de fluxo contínuo e fotografia/vídeo digital), melhorias significativas de desempenho para os multiprocessadores de *desktop* e melhor confiabilidade em relação as versões anteriores.

Após 5 anos, foi lançada uma nova versão chamada Windows Vista, que não foi bem recebida. Embora esta versão incluísse muitas melhorias que mais tarde apareceriam no Windows 7, tais melhorias foram ofuscadas pela lentidão e por problemas de compatibilidade percebidos pelos seus usuários. O Windows 7 foi lançado em outubro de 2009 e contemplava as edições *desktop* e servidor, corrigindo ainda, a maioria dos problemas reportados.

Em 2012, a Microsoft (MICROSOFT, 2017) lançou o Windows 8, agora compatível tanto com computadores tradicionais quanto com *tablets*. Trazendo melhorias significativas em relação a desempenho, reconhecimento de voz, suporte à USB 3.0, etc. Porém não obteve tanta aprovação pelos usuários, pois o famoso "Botão de Iniciar" não existia mais. Até que então em 2013, a Microsoft lançou o Windows 8.1, trazendo dentre outras coisas, o "Botão Iniciar" novamente.

Em 2014, a Microsoft lançou a versão mais recente, o Windows 10. Trazendo varias modificações, entre elas: múltiplos ambientes de trabalho, novo navegador (Microsoft Edge), aplicativos renovados (Foto, Vídeo, Música, Loja, Outlook, Office Mobile e até o *Prompt* de Comando).

2.3 Linguagens de Programação

É possível definir Linguagem de Programação como sendo um conjunto de regras sintáticas e semânticas padronizadas utilizadas para enviar instruções para o computador. Ou seja, para criar qualquer tipo de aplicação, é necessário utilizar uma linguagem de programação.

O importante é lembrar que os padrões e convenções devem ser seguidos à risca, independente de qualquer linguagem. Dessa forma, estar familiarizado com as nuances de cada linguagem irá ajudar a evitar alguns erros comuns cometidos pelos programadores.

A Figura 1 apresenta as linguagens de programação que são mais utilizadas no GitHub (GITHUB, 2017).

Foram escolhidas as linguagens C, Java e Python para serem discutidas neste

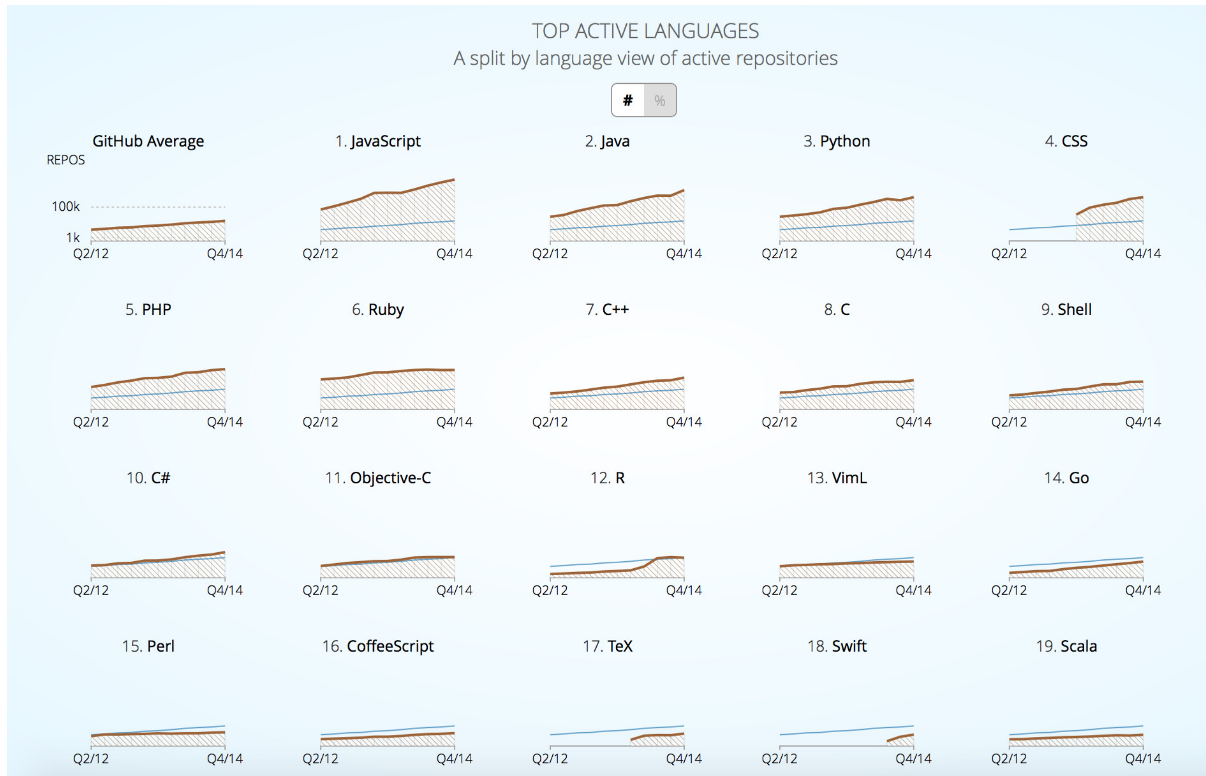

Figura 1 – Linguagens de Programação que são tendências no Github.

trabalho. A lista abaixo apresenta a justificativa desta escolha para cada linguagem.

1. **C** é a base utilizada nos Sistemas Operacionais baseados em Unix.
2. **Java** segunda linguagem mais utilizada pelos programadores que utilizam o GitHub.
3. **Python** terceira linguagem mais utilizadas pelos programadores que utilizam o GitHub.

Para a construção do protótipo do *Quizz*, foram utilizadas duas plataformas distintas, cada uma com sua linguagem de programação específica. Para a criação da aplicação móvel para a plataforma iOS, foi utilizado Swift. E para a criação da API REST foi utilizado o framework RubyOnRails. No contexto deste trabalho, não faz sentido explorar linguagens de programação que são evoluções e que já possuem mecanismos de programação defensiva por padrão. Por exemplo, a Apple cria uma série de condições que obrigam o desenvolvedor a seguir um padrão de programação e funcionalidades para que o aplicativo seja publicado na loja AppStore; e o RubyOnRails é um *framework* que para garantir que a aplicação funcione como esperado, é necessário seguir certos padrões. Maiores detalhes são apresentados na Seção 5. Em contrapartida, abordar linguagens que possuem problemas clássicos de programação defensiva pode gerar insumos mais consistentes e fáceis de entender, tais como C, Java e Python.

A seguir, são apresentadas algumas características dessas linguagens.

2.3.1 C/C++

C é uma das linguagens de programação mais populares e existem poucas arquiteturas que não dão suporte à essa linguagem. Criada em 1972, por Dennis Ritchie, ela foi utilizada para reescrever o Unix, que até então utilizava Assembly. Desde então, o C tem influenciado muitas outras linguagens de programação, mais notavelmente C++, que originalmente começou como uma extensão para C.

C é uma linguagem imperativa e procedural que deve fornecer acesso de baixo nível à memória e baixos requerimentos do hardware e também foi desenvolvido para ser uma linguagem de alto nível, para viabilizar reaproveitamento do código.

Ambas linguagens C e C++ incluem decisões de desenho fundamentais que tornam mais difícil escrever um programa seguro, pois permitem facilmente *Overflows*, forçando os programadores a criarem mecanismos para o gerenciamento de memória e não suportam exceções.

Para o gerenciamento de memória, o C dispõe de funções como: `malloc()`, `alloc()`, `free()`, `new()` e `delete()`, reconhecidas por não suportarem um mecanismo seguro para o gerenciamento de memória, o que pode significar um risco. O problema mais sério é que os programas podem erroneamente liberar memória que não deve ser liberada (por exemplo, porque já foi liberada). Isso pode resultar em uma falha imediata ou ser explorável, permitindo que um invasor faça com que algum código arbitrário seja executado. Ou ainda, se a memória utilizada não for liberada, o lixo de memória poderá ficar acumulado, causando lentidão e, muitas vezes, fazendo com que o programa pare de funcionar, mais uma vez, tornando um sistema computacional vulnerável a ataques.

Outro aspecto importante dessas linguagens, é que as variáveis são fortemente tipadas, ou seja, na declaração da variável é necessário deixar explícito qual será o tipo e se ela é *unsigned* ou *signed*.

Como em qualquer aplicação, a definição da Linguagem é uma decisão de projeto que deve ser bem planejada e baseada em uma séria de fatores, dentre os quais, a finalidade da aplicação. A Linguagem C é bastante recomendada para a escrita de kernel, módulos de kernel e outros artefatos de sistema básico.

2.3.2 Java

A Caelum (CAELUM, 2017), afirma que o Java é uma linguagem de programação interpretada, compliada e orientada a objetos. Um dos principais aspectos do Java é a utilização do conceito de Máquina Virtual, conhecido com JVM. Em linguagens de programação como o C, por exemplo, o código fonte é compilado para código de máquina

específico de uma plataforma e, naturalmente, dependendo de chamadas de sistema providas por um sistema operacional hospedeiro. Com isso, é necessário compilar uma vez para cada sistema operacional diferente, pois cada um fornece uma interface de interação diferente. Algumas vezes é necessário ainda reescrever partes do código para que a aplicação seja multiplataforma. A Figura 2 apresenta um esquema resumido de como funciona o processo de compilação de código fonte em C.

Figura 2 – Compilar Código Fonte em C.

Ao se desenvolver usando Java, o código fonte é compilado para *bytecode* e a JVM é responsável por fazer a comunicação entre a aplicação e o sistema operacional. Com isso, o Java consegue fornecer independência de plataforma e sistema operacional. A Figura 3 apresenta um esquema resumido de como funciona o processo de compilação de código fonte em Java. Fonte: (CAELUM, 2017).

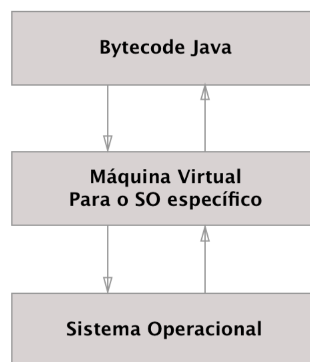

Figura 3 – Compilar Código Fonte em Java.

Esse funcionamento é interessante por que toda e qualquer instrução de código passa pela JVM, que por sua vez pode tirar métricas, decidir onde é melhor alocar a memória, identificar códigos maliciosos, entre outros. Uma aplicação em Java é totalmente isolada do sistema operacional pela JVM. Isso significa que se uma aplicação parar de funcionar, nem as outras aplicações e nem o sistema operacional é afetado.

Uma das características do Java é que o *bytecode* possui informações necessárias para ser transformada em Java novamente. Segundo (MIECZNIKOWSKI; HENDREN, 2002), o *bytecode* do Java é uma representação de programa baseada em pilha executada pela JVM. Todas as informações necessárias estão contidas no *bytecode*, que consiste em uma representação muito mais rica e de nível mais alto do que o código tradicional de baixo nível. Por exemplo, ele contém assinaturas de tipo completo para métodos e

chamadas de método. A natureza de alto nível do bytecode torna razoável esperar que ele possa ser descompilado de volta para Java. Isso pode ser bom ou ruim, dependendo do caso. Pois, caso o compilador seja conhecido, é possível através do *bytecode* ter acesso ao código, desta forma usuários maliciosos podem procurar por falhas no sistema e descobrir os segredos de negócio relacionados. Existem técnicas que são utilizadas para amenizar isso, por exemplo, *obfuscation*, que antes de criar o *bytecode*, altera os métodos e variáveis em nomes não significativos, tornando assim, mais difícil o trabalho de entender o código.

O Java também dá suporte a assinaturas digitais. Desta forma é possível identificar quem alterou o código pela última vez e quando essa modificação foi feita. Além disto, a Oracle (ORACLE, 2015), afirma que a capacidade de assinar e verificar arquivos é uma parte importante da arquitetura de segurança da plataforma Java. É possível configurar a diretiva para conceder privilégios de segurança aos *applets* e aos aplicativos. Por exemplo, você pode conceder permissão a um *applet* para executar operações normalmente proibidas, como gravar arquivos locais ou até mesmo executar programas localmente. Se você tiver baixado algum código que é assinado por uma entidade confiável, você pode usar esse fato como um critério para decidir quais permissões de segurança para atribuir ao código.

Segundo (DALE, 2009) uma característica importante do Java é que ele resolveu alguns problemas de alocação dinâmica de memória, pois, diferentemente de algumas outras linguagens, ele faz essa alocação de espaço na memória no momento de execução,em vez de ser no momento de compilação.

2.3.2.1 Garbage Collector

Em (ROMERO, 2017), o Tiago Romero afirma que problemas do *Garbage Collector* mau configurado podem parece simples em ambientes controlados de desenvolvimento, mas certamente farão a diferença em um ambiente de produção. Diferententemente das linguagens tradicionais como o C/C++, por exemplo, que alocam e desalocam memória explícitamente, o Java realiza este processo de forma automática, através do chamado *Garbage Collection*. Substituindo os ponteiros de memória por referências de objetos e deixando essa tarefa transparente ao programador, evitando assim problemas de vazamentos de memória e bugs de ponteiros.

De maneira geral, quando o programa está em execução, ele cria novos objetos, porém a partir de certo ponto, tais objetos não são mais necessários. A partir de então, esses objetos são chamados de “*garbage*”. Eventualmente, é possível que existam inúmeros objetos que não estão mais sendo usados, porém eles continuam consumindo memória. O que pode ocasionar em falta de memória disponível para continuar a correta execução do programa, tornando o sistema como um todo vulnerável.

A partir do momento em que um objeto não está mais sendo usado, o *Garbage*

Collector entra em ação desalocando esses objetos e liberando memória.

Levando em consideração que o programador não precisa se preocupar com o gerenciamento de memória, a produtividade pode aumentar, tornando-o livre para focar no desenvolvimento das funcionalidades da aplicação.

2.3.3 Python

Python é uma linguagem de alto nível e multi paradigma. Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e ainda exigir poucas linhas de código em implementações quando comparado ao mesmo programa implementado em outras linguagens. Python pode ser utilizado nos paradigmas orientado a objetos, imperativo, funcional e procedural.

Apesar dessa série de vantagens que o Python parece oferecer, ele também possui algumas características que podem representar riscos à segurança das aplicações desenvolvidas. Principalmente através de algumas funções *default*. Por exemplo, é possível permitir alguns dados sejam executados como partes de um programa. Isso inclui `exec()`, `eval()`, `execfile()` e `compile()`. E como em qualquer linguagem ou plataforma, a função `input()` também é surpreendentemente perigosa.

Os programas escritos em Python com privilégios que podem ser chamados por usuários que não possuem privilégios *root*(setuid/setgid); então, importar o módulo"*user*" pode ser um erro, pois este módulo faz com que o arquivo "pythonrc.py" seja lido e executado, o que viabiliza a execução de algum código arbitrário. Com isso, dar suporte à execução de um código não confiável em Python pode ser uma tarefa bastante difícil. Por exemplo, o Python chama muitos métodos "ocultos" por padrão.

Fundamentalmente, o Python foi projetado para ser uma linguagem limpa e objetiva, o que é bom para uma linguagem de uso geral, mas submete a aplicação a situações adversas que podem comprometer a segurança da aplicação ou de todo o sistema computacional.

O Python também suporta operações chamadas "*pickle*" e "*unpickling*" para convenientemente armazenar e recuperar conjuntos de objetos representados de forma contínua (diferentemente de simples referências à posições de memória). Essas funções são utilizadas para serialização e deserialização de objetos. Operações como essas são reconhecidas por representarem riscos, pois é possível injetar um código malicioso e enviar para outras pessoas. O próprio Python reconheceu essa falha e removeu essas funções isso da versão 2.3, e ainda, explicitamente deixou claro que não são operações seguras.

3 Plataformas Móveis

3.1 Mobilidade e Recursos

Este capítulo descreve como a computação móvel funciona, apresenta uma referência básica e uma visão geral de como o uso dos *smartphones* cresceu de forma impressionante nos últimos anos, como algumas das tecnologias disponíveis funcionam e quais são os recursos providos por elas. No Apêndice B encontra-se um estudo mais profundo da plataforma iOS, apresentando conceitos do sistema operacional utilizado, vulnerabilidades do sistema, recursos de segurança, aplicações, entre outros. É apresentado também um breve resumo sobre IoT (*Internet of Things*) que é uma forte tendência e promete revolucionar a interação homem-computador conhecida nos dias atuais.

Sacol (SACCOL; REINHARD, 2007) em seu estudo afirma que por este assunto ser bem recente, alguns conceitos podem gerar certa confusão conceitual. Desta forma, conforme a bibliografia, a maioria deles são entendidos pela linguagem natural e os que necessitarem de maior explicação terão, eventualmente, seus conceitos definidos e padronizados.

3.1.1 Aplicações Móveis

Desde o advento da computação, a ideia de se poder ter um computador tão pequeno a ponto de se carregar para qualquer lugar parecia meio futurística, e era; porém, nos dias atuais, isso já se demonstrou possível. O primeiro dispositivo com essa característica era conhecido como PDA. Gerido por mini sistemas operacionais, permitia ao usuário acesso à Internet e ao e-mail. Com o passar do tempo, surgiram outros dispositivos móveis similares a ele, que, além de substituir algumas capacidades básicas de um computador, começam a dar suporte para aplicativos que aproveitavam a mobilidade provendo funcionalidades que até então não estavam disponíveis em computadores tradicionais. Um grande exemplo desses dispositivos é o celular criado pela Apple Inc, que, em 2007, promoveu uma quebra de paradigma com o iPhone ao proporcionar uma nova forma de interagir com esses dispositivos móveis(INC, 2007). Maiores detalhes podem ser encontrados no Capítulo B.0.7 do Apêndice B.

3.1.2 Redes Móveis

Alguns dispositivos móveis possuem acesso à Internet. Isto é possível graças à uma rede de telecomunicações.

Tabela 1 – Características de acordo com cada geração de rede móvel.

Geração	Características
1G	Transmissão de Dados Analógica (AMPS) Taxas de 9600bps
2G	Transmissão digital de dados (TDMA, CDMA e GSM) Taxas de 9600bps a 14400bps Surgimento de aplicações WAP.
2,x G	Disponibilização de aplicações pré 3G
3G	Evolução CDMA e GSM Taxas de até 2Mbps Surgimento de aplicações multimídia.
4G	Elevação das taxas de transmissão de dados Tecnologias e aplicações ainda em discussão.

Em seu estudo, (FIGUEIREDO; NAKAMURA, 2003) apresenta um comparativo sobre o histórico das tecnologias disponíveis de acordo com suas características, mostrando a tecnologia atrelada às gerações e sua evolução. Este estudo é apresentado na Tabela 1. O mesmo autor deixa algumas discussões sobre o tema. Um estudo mais aprofundado sobre esse assunto é deixado para outros trabalhos.

3.1.3 Redes Sem Fio – Wi-Fi

Segundo (SACCOL; REINHARD, 2007), são tecnologias que possibilitam que dispositivos conectem-se entre si, permitindo a transmissão de dados e informações sem a necessidade do uso de cabos. Além de acesso à rede Internet sem fios (Wi-Fi), o trabalho cita ainda algumas tecnologias, como por exemplo, Infra-Vermelho(IR), Bluetooth, Wi-Max, etc. Seu uso mais comum é em redes de computadores, fornecendo acesso à Internet. Os protocolos de rede sem fio (Wi-Fi) utilizam WEP e WPA para segurança dos dados e informações. Existem uma série de limitações, desafios e oportunidades disponíveis nesta área que vem evoluindo de forma impressionante. Assim como o tópico anterior, um estudo mais profundo sobre este tema será deixado para outros trabalhos.

3.1.4 Bluetooth

Os esquemas de comunicação sem fio utilizados na computação móvel é, em alguns casos, provido pelo padrão Bluetooth, que além de permitir conexão sem fio a dispositivos como de fones de ouvido, *mouses*, teclados, proporciona ainda esquemas mais sofisticados de comunicações, um exemplo é a tecnologia conhecida como AirDrop, que a Apple (APPLE, 2016b) apresenta. Rohan em seu estudo (ROHAN, 2016), afirma que a tecnologia do Bluetooth está crescendo e evoluindo cada vez mais, atendendo a variados tipos de aplicações, de transferência de dados até sincronização de periféricos ou relógios inteligentes com outros dispositivos.

As possibilidades que o Bluetooth traz tanto para os desenvolvedores quanto para os usuários são inúmeras, porém os problemas com a segurança dos dados, espionagem, perda de dados privados e controle de dispositivos não autorizados vêem juntos com as funcionalidades e conveniências. Para promover o desenvolvimento e a aceitação em relação ao uso dessa tecnologia, segundo a própria fundação (SEARCH, 2016), em 1998 foi formado o *Bluetooth Special Interest Group* (SIG) e, desde então, a tecnologia tem evoluído bastante e novos padrões tem surgido com contramedidas a fim de minimizar os riscos em relação a segurança.

3.1.4.1 Serviços Tradicionais de Segurança no Bluetooth

Os desenvolvedores estão habituados a ter acesso a certos serviços básicos de segurança tais como autenticação, autorização, integridade e confidencialidade. Analisando cada serviço desses, podemos ter uma ideia do nível de segurança de uma determinada tecnologia. Segundo (DWIVEDI CHRIS CLARK, 2010), as especificações do Bluetooth incluem três serviços básicos: Autenticação, Autorização e Confidencialidade.

- **Autenticação** É a capacidade, fornecida pelo Bluetooth, de identificar os dispositivos antes e durante a comunicação entre eles.

- **Autorização** É a capacidade de fornecer acesso a recursos selecionados baseado nas permissões fornecidas pelo Bluetooth.

- **Confidencialidade** É a capacidade de proteger as comunicações durante a comunicação através da rede fornecida pelo Bluetooth.

Como o Bluetooth fornece apenas serviços de segurança à nível de dispositivo, eles não podem ser utilizados para limitar o acesso a dados sensíveis, ou algo assim. Caso algum serviço específico seja necessário, o desenvolvedor pode utilizar outros mecanismos. Por exemplo, se for necessário desenvolver uma aplicação com autenticação simples de usuário, o desenvolvedor deve implementar o mecanismo que deverá conversar com o dispositivo conectado de tal forma que a autenticação seja possível. Analisar todos os aspectos e limitações do Bluetooth é importante do ponto de vista do desenvolvedor, como qualquer outro assunto, para que o desenvolvimento seja planejado da melhor forma possível, levando em consideração os aspectos de segurança, ainda mais utilizando de uma tecnologia sem fio.

3.1.5 Geolocalização

Tanto no mundo *mobile* quando na computação em geral, a quantidade de aplicativos que usam a localização do usuário tem crescido bastante, seja pra possibilitar

alguma funcionalidade ou para melhorar a experiência do usuário. Atualmente, os serviços de localização são utilizados para marcar o local em que fotos foram tiradas, saber quais amigos estão por perto, saber quais estabelecimentos comerciais estão próximos do usuário, avisar ao chegar próximo do trabalho ou de algum outro local, e até mesmo quando determinado ônibus vai chegar na parada. Apesar de todos esses novos recursos e conveniências, a utilização deste serviço traz consigo uma série riscos para os usuários.

3.1.5.1 Riscos relacionados à Geolocalização

O impacto que sua utilização pode causar ainda é imprevisível, pois mensurar desde a quantidade até quais dados estão sendo coletados é uma tarefa realmente difícil. Usuários finais podem ter os seus dados armazenados em servidores não confiáveis de terceiros, e caso não sejam anônimos, pode representar um risco no caso de algum vazamento dessas informações.

Himanshu em seu estudo (DWIVEDI CHRIS CLARK, 2010), diz afirma que os desenvolvedores são responsáveis pelos dados dos usuários, e é recomendado que esses dados não sejam armazenados por um longo tempo e sejam anônimos quando possível, pois caso contrário podem representar um problema.

3.1.5.2 Boas Práticas relacionadas à Geolocalização

O mesmo autor cita uma série de boas práticas que se adotadas, podem aumentar significativamente a confiança do usuário ao utilizar um aplicativo, e o desenvolvedor, por sua vez, minimiza custos e riscos desnecessários. Alguns deles são apresentados a seguir:

- Utilize a menor precisão necessária para a funcionalidade desejada.
- Descarte os dados coletados o mais rápido possível após o uso.
- Mantenha os dados de forma anônima.
- Mostre ao usuário quando e quais dados estão sendo coletados.
- Tenha uma política de privacidade bem definida e consistente.
- Não repasse os dados coletados para outros serviços ou pessoas.
- Conheça as leis locais que são relevantes para o serviço e, se possível, adeque a política de privacidade e utilização para cada local onde ele existe.

4 Desenvolvimento Seguro

Este capítulo apresenta uma visão inicial sobre algumas práticas comuns que podem melhorar a segurança das informações processadas em um sistema computacional. Segundo (GOMES et al., 2014), escrever um código seguro é um dos aspectos mais importantes do ciclo de desenvolvimento que leva em consideração aspectos de segurança do sistema. Qualquer código está sujeito à ataques, e o risco é ainda maior quando são escritos sem a implementação adequada dos controles de segurança.

Ademais, esse é um tema em que se observa um volume elevado de atividades: a cada dia surgem novos tipos de ataques, novas vulnerabilidades são descobertas e pesquisadores buscam técnicas aprimoradas para agressão e defesa desses sistemas.

Todos os aspectos de falhas comuns conhecidas, recursos e características intrínsecas de cada Sistema Operacional apresentados nos Capítulos anteriores devem ser tomados apenas como ponto de partida. Para aperfeiçoar ainda mais os estudos em programação segura, são apresentadas condições adversas a serem observadas pelos desenvolvedores e profissionais de tecnologia da informação, erros comuns e boas práticas para enrobustecer sistemas computacionais.

4.1 Princípios de Segurança

Existem vários princípios e práticas que são, por convenção, aceitas pela comunidade de software. Alguns autores elencam tais princípios usando os conceitos de Confidencialidade, Integridade, Disponibilidade, Autenticidade, Não Repúdio, Privacidade. Segundo (WHEELER, 2015), em geral, todos os autores e órgãos responsáveis costumam condensar a descrição dessas práticas em termos de três princípios globais:

- **Confidencialidade** Somente pessoas autorizadas devem ter acesso à informações privadas.
- **Integridade** Os dados só podem ser modificados ou apagados por partes autorizadas de formas autorizadas.
- **Disponibilidade** Os dados são acessíveis às pessoas autorizadas em tempo hábil.

De qualquer forma, o ideal é identificar quais são os objetivos de segurança que cada aplicação necessita, independentemente de como esses princípios são chamados ou agrupados. Em geral, eles são identificados através das ameaças iminentes da aplicação em foco. A intenção é atender cada objetivo definido como necessário.

Existe um binômio "Segurança - Usabilidade" que, em muitos casos, são conflitantes. Para que um sistema seja seguro, é necessário criar mecanismos de segurança que podem interferir na usabilidade do software. O ideal é ter um software seguro e fácil de utilizar. Por exemplo, como visto no Capítulo 2.2.3, o Windows é um sistema que foi projetado para ser de fácil utilização, e, com isso, expõe o usuário final a uma série de vulnerabilidades, tais como a execução automática de algum processo e a supressão de interação mandatória com o usuário em relação a execução de certas aplicações em segundo plano. O fato é que alguns princípios podem se opor, o que demanda o balanceamento das decisões para que os objetivos anteriormente definidos sejam atendidos pelo menos no percentual que a aplicação necessita.

A segurança no desenvolvimento de software deve levar em consideração diversos fatores e não é um problema de solução única. Sua abordagem envolve desde pessoas tecnicamente capacitadas na linguagem, um planejamento criterioso no levantamento de requisitos, equipamentos com poder computacional suficiente e equipe de monitoramento pró-ativa para criar e sustentar aplicações com certo nível de segurança.

Em (FAGET, 2017), Dominique apresenta um exemplo da abordagem de amplo espectro para a solução de problemas de segurança da informação pode ser derivado das lições aprendidas com o ataque cibernético ocorrido em 12 de Maio de 2017 e que atingiu pelo menos 74 países. A boa prática de manter o sistema operacional sempre atualizado seria mais do que suficiente para proteger uma infraestrutura computacional desse ataque. Assim que uma falha de segurança é encontrada, os fabricantes e mantenedores dos Sistemas Operacionais costumam lançar atualizações o mais rápido possível corrigindo as falhas identificadas.

Outra boa prática é, no que diz respeito à programação, tomar como base um método de desenvolvimento seguro aumentando assim, a confiabilidade do sistema; além disso, há alguns princípios de computação segura que devem ser também levados em consideração.

4.2 Segurança da Informação e Problemas Clássicos

HImanshu (DWIVEDI CHRIS CLARK, 2010) afirma que com o advento da computação e sua difusão, em meados dos anos 1980, questões de segurança eram dificilmente levadas em consideração. Porém a partir do momento em que os dispositivos com algum tipo de informação sensível passaram a ter acesso à Internet, a segurança dos dados se torna uma questão importante. E assumir que o dispositivo poderá ser acessado por pessoas não confiáveis é o passo inicial para se preocupar com esse tipo de questão.

A maior parte desse trabalho de provimento de mecanismos de proteção é feita pelos próprios fabricantes que disponibilizam o Sistema Operacional, mas o desenvolvedor

também deve estar atento a alguns aspectos. Pois, segundo (SILBERSCHATZ; GALVIN; GAGNE, 2013) qualquer sistema de proteção se torna ineficaz caso a autenticação do usuário seja comprometida ou se um programa for executado por um usuário não autorizado.

Segundo (GOMES et al., 2014), o desenvolvedor deve conhecer os 25 erros de codificação mais comuns que deixam a aplicação vulnerável e as 10 vulnerabilidades mais comuns, ambos podem ser encontrados em (MARTIN et al., 2011) e (PROJECT, 2011), respectivamente.

A seguir, serão apresentados alguns cenários que possuem riscos potenciais no que diz respeito à segurança da informação.

4.2.1 Armazenamento de Dados

Himanshu em (DWIVEDI CHRIS CLARK, 2010), afirma que a possibilidade de pessoas mal intencionadas terem acesso à algum dispositivo é real e deve ser levada em consideração. Seja algum conhecido ou até mesmo desconhecido, e basta que se tenha acesso por um pequeno momento apenas para que os dados contidos no dispositivo estejam em risco. Porém, isso deixa de ser um problema a partir do momento em que estes dados estão protegidos por algum mecanismo, seja controle de acesso por uma senha ou dados em persistência criptografados. Tratar essa possibilidade garantindo que apenas as aplicações e/ou donos da informação tenham acesso quando necessário é um ponto importante.

4.2.2 Divulgação de Informações

O mesmo autor defende a ideia de que a segurança da informação está diretamente relacionada com a proteção de dados sensíveis; dessa forma, evitar que os mesmos sejam divulgados é essencial.

Se, por qualquer motivo, alguma pessoa não confiável tiver acesso à informações, o risco das mesmas serem divulgadas é real. Esta é uma questão importante para tratar e mitigar.

4.2.3 Fraude Eletrônica (*Phishing*)

É um grande problema das aplicações Web e também em dispositivos móveis. Principalmente devido ao comportamento imponderado de alguns usuários que clicam em itens sem pensar ou sem saber o que estão fazendo. Nos dispositivos móveis, em especial, muitos navegadores de Internet colapsam a URL do site que está sendo acessado. Com isso, os usuários podem facilmente serem vítimas de fraudes. O *phishing* é uma ataque que funciona da seguinte forma: um site idêntico ao de um banco, por exemplo, é criado e, quando um usuário sob ataque tenta acessar o site original, uma versão replicante (e

falsa) aparece com o objetivo, por exemplo, de coletar os dados da conta e senha de acesso ao sistema eletrônico. Quando o usuário tenta acessar a conta, aparece uma mensagem de erro, mesmo que os dados estejam corretos. Neste ponto, as informações digitadas nos formulários do site replicante são enviadas pra algum servidor de terceiros e o site falso redireciona o usuário para o site original. Muitas pessoas só percebem que foram vítimas quando vão verificar a fatura do cartão de crédito ou saldo na conta, muito tempo depois de ter acontecido.

4.2.4 Ruptura de Isolamento de Aplicações

Existe uma infinidade de tipos de aplicações que podem ser instalados, e na maioria dos casos, eles diferem bastante de um dispositivo para o outro. Cada tipo de aplicativo, para funcionar conforme esperado, precisa de um nível de acesso diferente às informações e recursos do dispositivo. A capacidade de isolar essas aplicações e os dados de que cada uma necessita é um passo importante para garantir que um aplicativo não autorizado tenha acesso aos dados de outros.

4.2.5 Overflow

Segundo (SILBERSCHATZ; GALVIN; GAGNE, 2013) o *Overflow* é um dos jeitos mais comuns para um invasor ter acesso não autorizado a um sistema. Em termos gerais, o ataque busca explorar algum bug de um programa. Esse bug pode ser um caso simples de má programação, em que o programador negligenciou a verificação dos limites no caracteres em campos de entrada. Nesse caso, o invasor envia mais dados do que o programa esperava. Ao usar a tentativa e erro, ou examinando o código-fonte do programa atacado se estiver disponível, o invasor determina a vulnerabilidade e escreve um programa para invadir o sistema da seguinte maneira:

- Utilizando algum campo de entrada, por exemplo login, o invasor adiciona um trecho de código que será gravado na pilha de execução do programa;
- Substitui o endereço de retorno atual da pilha de execução com algum código adicionado, como no próximo passo, e
- Escreve um conjunto simples de código para o próximo espaço na pilha que inclui os comandos que o atacante deseja executar, por exemplo, criar um usuário com permissão de root para ter acesso total ao sistema.

De qualquer forma, esse trecho de código que será executado poderá conter qualquer tipo de instrução o que torna a aplicação totalmente vulnerável. Segundo (SILBERSCHATZ; GALVIN; GAGNE, 2013), evitar esse tipo de situação é necessário para garantir a segurança da aplicação.

Existem vários tipos de *Overflow* que devem ser considerados, entre os mais comuns temos: *Buffer Overflow, Integer Overflow e Stack Overflow*. Em termos gerais, o Overflow é o "transbordamento" de memória, ou seja, é passar "mais do que é suportado".

A seguir são apresentados *Buffer Overflow* e o *Integer Overflow*.

4.2.5.1 *Buffer Overflow*

Em seu estudo (ONE, 1996), Aleph diz que o *Buffer Overflow* é provocado sempre que se coloca mais dados em um *buffer* do que ele pode armazenar. Um exemplo código em C que gera essa situação e a sua explicação é apresentado a seguir:

```
void function(char *str) {
  char buffer[16];

  strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
```

A função `function(args)` copia uma string fornecida sem a verificação de limites utilizando `strcpy()` ao invés de `strncpy()`. Ao tentar executar esse trecho de código, o sistema operacional deverá retornar um erro de *Segmentation Violation*, pois `strcpy()` está copiando o conteúdo de `*str (larger_string[])` na variável `buffer[]` até que um caractere nulo seja encontrado na string. Porém `buffer[]` é muito menor do que `*str` pois tem 16 bytes de comprimento, e tentar encher com 256 bytes significa que todos os 240 bytes após buffer na pilha estão sendo sobrescritos.

A variável `larger_string[]` foi preenchida com o caractere "A", o seu valor convertido para base hexadecimal é igual à "0x41". Isso significa que o endereço de retorno agora é "0x41414141", ou seja, fora do espaço de endereço do processo. É por isso que quando a função retorna e tenta ler a próxima instrução a partir desse endereço, o compilador emite o erro de *Segmentation Violation*.

Em resumo, através do *Buffer Overflow* é possível controlar e manipular o fluxo de execução do programa.

4.2.5.2 Integer Overflow

Segundo (DIETZ et al., 2015), o *Integer Overflow* é um bug nas linguagens C e C++ difícil de detectar e que pode levar a erros fatais ou vulnerabilidades exploráveis nas aplicações.

Apesar de existirem ferramentas que detectam esses bugs, nem todos os *overflows* são bugs. Erros numéricos de inteiros em aplicativos de software podem ser caros e exploráveis. Esses erros incluem *overflows*, conversões de valor que perdem precisão e usos ilegais de operações. Todos esses tipos de erro são chamados de *Integer Overflows*.

Em (ARNOLD, 2000), Arnold apresenta um exemplo que demonstra a sua gravidade foi o erro que destruiu o voo 501 do foguete Ariane 5 em 1996. Nesse acidente, um erro durante o *casting* de um valor de um ponto flutuante para um inteiro de 16 bits ocasionou uma pane no sistema que levou à sua explosão.

Detectar um *Integer Overflow* é relativamente simples usando um compilador modificado para inserir verificações em tempo de execução. No entanto, em linguagens de baixo nível como C e C++ que manipulam bits e bytes, é interessante ter esse cuidado.

4.2.6 Ataques a *Parsers* - Zip Bomb

Como citado na Sessão 2.3.3, ações de *parsing* podem representar riscos à segurança da aplicação e/ou computador. Existem alguns problemas clássicos, um deles é conhecido como "Zip Bomb", que transforma um arquivo que inicialmente possui espaço em disco extremamente pequeno, e após o *parsing* pode consumir toda a memória do computador. Mais detalhes são apresentados a seguir.

Segundo Selvaraj, (SELVARAJ; ANAND, 2012), a ideia do *Zip Bomb* surgiu após os servidores de e-mail serem configurados para descompactar todos os arquivos anexados e verificar o seu conteúdo em busca de vírus. Desta forma, se os atacantes enviassem uma "bomba" em arquivo de texto, por exemplo, um programa que repete milhões de vezes a letra "z", tal arquivo se comprimido seria relativamente pequeno, mas sua descompactação especialmente por versões anteriores dos servidores de correio usaria uma quantidade elevada do poder de processamento, RAM e espaço de troca, o que poderia resultar em negação de serviço. Fazendo com que os servidores parassem e passassem a negar o serviço.

Desta forma os atacantes perceberam que além dos servidores de e-mail, seria possível atacar computadores comuns também, para deixá-los vulneráveis à outros tipos de ataque.

4.2.7 Vulnerabilização de Privacidade por Serviços de Localização

Como citado anteriormente, a maioria dos dispositivos móveis atuais possuem GPS integrado e, nas aplicações com acesso a Internet, é possível saber onde o usuário está através do IP ou GPS.

Grande parte desses usuários desejam privacidade, mas, sem perceber, compartilham sua localização com aplicativos de mapa ou redes sociais e desconsideram o cenário em que possam estar sendo observados.

Quase todas essas aplicações permitem que o usuário escolha com quem deseja compartilhar essa informação ou tratam esses dados de forma anônima, mas existe a possibilidade de pessoas ou aplicações não confiáveis terem intenções adversas e, na grande maioria dos casos, desfavoráveis ao usuário alvo da coleta não voluntária de informações. Há, naturalmente, o caso em que se observam políticas de uso onde abertamente declaram que irão usar esses dados, todavia o usuário, por omissão, não lê na íntegra o documento e fica exposto aos efeitos adversos e, muitas vezes, indesejados da política de uso do fornecedor.

Então para que a utilização desta tecnologia não represente um risco, é importante ter bastante cuidado.

4.2.8 Vulnerabilização de Segurança Física

Em (DWIVEDI CHRIS CLARK, 2010), Himanshu afirma que não é possível garantir a segurança de um ecossistema se a segurança física não for garantida. Existem inúmeras formas de ataques físicos, seja pela rede, e-mails, pen drives, sites maliciosos, etc. Uma das ferramentas utilizadas para “hackear” dispositivos é conhecida como “Bash Bunny”. Segundo (GRIMES, 2017), é assustador o que esse dispositivo pode fazer pois as possibilidades são infinitas. Além do mais, é absurdamente fácil atacar computadores com esse dispositivo. O “Bash Bunny” é um sistema projetado especificamente para executar *scripts* ao se conectar à um computador. Possui uma interface USB, que o deixa muito parecido à um pen drive, inclusive ele possui também essa funcionalidade. Ele pode ser usado tanto em computadores com Windows, MacOS, Linux, Unix ou Android. Para burlar os sistemas de segurança do dispositivo a que é conectado, ele pode falsificar a sua identidade, se passando por um mouse ou teclado, por exemplo. O que torna esse método extremamente perigoso é que alguns *scripts* foram projetados para serem executados mesmo com a tela bloqueada, bastando apenas que seja conectado ao computador.

Ademais, o termo “dispositivos móveis” remete à mobilidade que os mesmos trazem consigo, o que, por outro ponto de vista, pode significar um risco no que diz respeito à roubos ou perdas dos equipamentos. O que pode acontecer, no melhor caso, é a perda do valor pago no dispositivo em questão; porém, o pior cenário é o acesso a informações

sensíveis contidas no aparelho, que em muitos casos pode valer até mais do que o próprio dispositivo e as consequências disso são até mesmo difíceis de mensurar.

4.2.9 Sistemas Operacionais Vulneráveis

Proteger um Sistema Operacional é uma tarefa difícil mas necessária. Porém quanto mais seguro for, melhor será a experiência do usuário. Pois a segurança muitas vezes está relacionada com a perda de informações, inatividade do sistema, etc. E quanto mais simples for para o usuário resolver esse tipo de problema, melhor.

4.2.10 *Malwares: Vírus, Worms, Trojans e Spyware*

Em qualquer dispositivo que possua algum tipo de conexão externa, essas ameaças podem representar um risco à segurança dos dados contidos no mesmo. A capacidade de se adaptar às mudanças levando em consideração os anos de conhecimentos anteriores sobre o tema é crucial para a criação de aplicações e sistemas operacionais mais seguros e resistentes ao ataques desses artefatos comumente referidos como *malwares*.

4.3 Erros Comuns no Desenvolvimento de Software

O Mitre (MITRE, 2017) em parceria com a SANS (NETWORKING; SECURITY, 2017) criou o *CWE/SANS Top 25* (MARTIN et al., 2011) que descreve os 25 erros de software mais perigosos. Este documento contribui para uma linguagem comum que descreve as vulnerabilidades da programação.

Segundo (GOMES et al., 2014), os erros descritos são considerados mais perigosos pelos seguintes motivos:

- Ocorrem com frequência;
- São fáceis de encontrar;
- São fáceis de ser explorados;
- Com frequência permitem que um atacante tome o controle total do software, roube informações ou faça com que o software pare de funcionar.

Segundo (GOMES et al., 2014), o *CWE/SANS Top 25* pode ser utilizado como forma de treinamento e conscientização dos programadores com ou sem experiência em segurança para que eles evitem os erros mais comuns de implementação de software.

O próprio documento divide estes 25 erros mais comuns em 3 (três) categorias gerais apresentadas a seguir.

Tabela 2 – Formas inseguras de envio e recebimento dos dados entre componentes, módulos, programas, processos, etc.

Interação Insegura Entre Componentes	
Posição	**Nome**
1	Validação imprópria de elementos usados em comandos SQL.
2	Validação imprópria de elementos usados em comandos do SO.
4	Validação imprópria de entrada de dados durante a geração da página Web.
9	Falta de restrições no upload de arquivos.
12	*Cross-Site Request Forgery (CSRF).*
22	Redirecionamento para URLs não confiáveis.

Tabela 3 – O Software não gerencia adequadamente a criação, uso, transferência ou destruição de recursos do sistema.

Risco no Gerenciamento de Recursos	
Posição	Nome
3	Cópia do buffer sem verificar o tamanho da entrada (“BufferOverflow”)
13	Limitação imprópria de um nome de caminho para um diretório restrito
14	Download do código sem verificação de integridade
16	Inclusão da funcionalidade da esfera de controle não confiável
18	Uso da função potencialmente perigosa
20	Cálculo incorreto do tamanho do buffer
23	Formato de String não controlado
24	Integer Overflow ou Wraparound

Tabela 4 – Técnicas defensivas mal configuradas que podem ser burladas ou ignoradas.

Configurações Inseguras	
Posição	**Nome**
5	Falta de autenticação em funções críticas.
6	Falta de autorização.
7	Informações credenciais explícitas no código.
8	Falta de criptografia de dados sensíveis.
10	Confiança em entradas inseguras na tomada de decisão.
11	Execução com privilégios desnecessários.
15	Autorização incorreta.
17	Atribuição incorreta de permissões para recursos críticas.
19	Uso de um algoritmo fraco de criptografia.
21	Restrição imprópria de tentativas excessivas de autenticação.
25	Uso de função de hash fracas.

- Interação insegura entre componentes (Tabela 2).
- Gerenciamento inadequado de recursos (Tabela 3).
- Configurações inseguras (Tabela 4).

Maiores detalhes podem ser encontrados no site oficial do CWE/SANS. (MARTIN et al., 2011)

4.4 Boas Práticas no Desenvolvimento Seguro

Vários conceitos, ferramentas, listas e orientações que são apresentadas neste trabalho estão disponíveis em uma iniciativa aberta que é focada em aspectos de segurança das aplicações Web conhecida como OWASP (OWASP, 2017). Segundo (GOMES et al., 2014), esta associação congrega especialistas de empresas privadas, órgãos governamentais, instituições acadêmicas e voluntários de todo o mundo. Trabalha principalmente na criação de artigos, metodologias, documentação, ferramentas e tecnologias para aprimorar a segurança das aplicações.

A seguir serão listadas algumas boas práticas que, se seguidas, ajudarão e muito nesse propósito.

4.4.1 Princípios da Arquitetura Segura

Segundo (GOMES et al., 2014), durante a elaboração da arquitetura do software, uma das metas é balancear os objetivos como o custo, a performance, a manutenabilidade e a segurança. É provável que um sistema com foco apenas em segurança apresente baixo desempenho e vice-versa. A seguir são apresentadas algumas orientações que, segundo autor, devem ser seguidas: Privilégios Mínimos, Separação de Responsabilidades, Defesa em Profundidade, Falha Segura, Evitar Segurança por Obscuridade, Isolamento de Entidades Não Confiáveis, Seguir Práticas de Programação Segura e Validar Entradas.

Essa lista é bem extensa, então esses princípios foram agrupados em 3 objetivos gerais:

- Minimizar o número dos alvos de alta consequência;
- Não expor componentes vulneráveis; e
- Negar ataques que venham a comprometer o sistema.

Algumas dessas principais práticas são apresentadas a seguir:

4.4.2 Seguir Práticas de Programação Segura

Estudar a linguagem na qual o aplicativo será desenvolvido é importante. Saber quais são as nuances e práticas de segurança que a linguagem oferece também. Para isso, tempo e experiência na linguagem contam bastante. Apesar de que, em muitas ocasiões, as

aplicações devem ser entregues em pouco tempo, não se pode ignorar testes e verificações de segurança, pois um produto potencialmente inseguro pode significar um problema no futuro.

A maior parte das linguagens possuem documentação extensa e guias para uma programação segura. O aproveitamento dessas informações é importante para fazer um código o mais seguro possível. E ainda evitar que erros comuns sejam praticados.

4.4.3 Validar as Entradas de Informações

A validação das entradas de informações vem desde o advento da programação Web nos anos 2000. E, desde então, é uma recomendação padrão da maioria das linguagens.

Nas plataformas tradicionais, o maior risco está em aplicações Web, pois, historicamente possuem muitos campos a serem preenchidos de forma livre, diferente dos dispositivos móveis, em que as entradas de dados quase não variam limitando-se a esquemas como *picker, switch, table*, ou seja, seletores com entradas pré-definidas.

De qualquer forma, a importância de se validar as entradas não pode ser subestimada.

4.4.4 Utilize os Privilégios Mínimos Necessários

As entidades devem receber os privilégios mínimos requeridos e durante o menor tempo necessário.

Permissões de acesso a recursos do dispositivo, tais como GPS, Internet, Contatos, Fotos, etc., devem ser os mínimos possíveis. Não faz sentido uma aplicação cujo objetivo seja apenas publicar fotos em alguma rede social solicitar permissão de acesso aos contatos ou mensagens do dispositivo. Este modelo de menor privilégio possível envolve apenas pedir o que é estritamente necessário para o funcionamento da aplicação.

Sua adoção garante que a aplicação não afete os outros programas instalados no dispositivo e que os recursos funcionem da forma como deveriam. Reduzindo a oportunidade de acesso não autorizado às informações sigilosas da aplicação.

4.4.5 Armazene Informações Corretamente

É recomendado evitar o armazenamento de informações confidenciais, como nomes de usuário ou senhas, em textos em claro (não protegidos por cifração) no dispositivo ou em algum local de fácil acesso. Utilize os recursos de criptografia e banco de dados provenientes da plataforma que permitem que os aplicativos armazenem tais informações com segurança e sem a necessidade de utilizar softwares de terceiros.

4.4.6 Evite Ameaças Comuns

Embora ameaças para aplicações sejam reais, é importante saber distinguir quais tipos de ameaças realmente devem ser levadas em consideração para o contexto da aplicação. Qualquer livro de segurança apresenta extensas listas de ameaças; é necessário entender como cada uma delas funciona e quais delas podem representar um risco. Esse processo é comumente conhecido como "Modelo de ameaça para a aplicação". Recomenda-se que a pesquisa de modelamento não deva ser exaustiva ou super completa, basta servir como um guia para os desenvolvedores entenderem como tratar cada tipo de ameaça.

4.4.7 Separação de Responsabilidades

Este princípio defende que devem ser atendidas múltiplas condições para completar uma determinada tarefa sensível ou acessar uma informação sigilosa.

4.4.8 Defesa em Profundidade

Consiste em uma aplicação com múltiplas camadas de proteção e cada camada deve fornecer mecanismos de segurança, caso a segurança da camada anterior seja transposta. Mesmo usando-se dessa estratégia, é importante lembrar de que a composabilidade nem sempre é viável em esquemas de segurança da informação.

4.4.9 Falha Segura

Caso ocorra uma falha no sistema ele deve retornar a um estado onde a segurança não tenha sido comprometida, por exemplo, onde os privilégios de acesso e escrita são revogados para todos os usuários.

4.4.10 Evitar Segurança por Obscuridade

A segurança não deve ser obtida pelo desconhecimento de como o sistema funciona, mas sim por um controle de segurança bem implementado.

4.4.11 Isolamento de Entidades Não Confiáveis

Isolar entidades e processos não confiáveis para que o comportamento deles não afete o funcionamento esperado do sistema ou consiga acesso a áreas restritas. Pode ser obtido através de *sandboxes*, máquinas virtuais, configurações de controle e acesso e módulos específicos, etc.

5 O Quizz

5.1 Breve resumo

O *Quizz* busca analisar o perfil de cada usuário indicando pontos fortes e fracos e incentivando a utilização de boas práticas de programação no que diz respeito à segurança da informação.

5.2 Como Funciona

O *Quizz* busca apresentar as principais falhas de segurança e os erros mais comuns de programação segura.

Para a correta utilização, é necessário aplicar um questionário pra traçar o perfil do programador. Em seguida, é necessário estudar aspectos de segurança, principalmente nos pontos fracos indicados pela primeira rodada de perguntas. É importante ressaltar que o usuário não fica restrito apenas a essa parte dos pontos fracos, pois ele tem acesso à todo o conteúdo levantado neste trabalho. Feito isso, é aplicada outra rodada de perguntas para avaliar o quão efetivo foi o estudo e, no final, apresentar um panorama geral de evolução. O usuário tem total liberdade e consegue saber o resultado após cada pergunta, bem como ao final de cada etapa.

5.3 Aspectos Técnicos

5.3.1 Tecnologias

Para a criação deste *Quizz*, foi criado um aplicativo para a plataforma iOS, utilizando a linguagem de programação Swift (SWIFT, 2016), e uma API para que esse aplicativo demande os dados e esteja sempre com questões atualizadas. Essa API foi desenvolvida utilizando o *framework* Ruby On Rails. Maiores detalhes podem ser encontrados no site oficial do RubyOnRails, (RUBYONRAILS, 2017).

5.3.2 Aplicativo iOS

5.3.2.1 Arquitetura

Como foi apresentado na Seção B.0.7.1, os aplicativos desenvolvidos para a plataforma iOS seguem o padrão MVC (*Model-View-Controller*). Para o desenvolvimento desta aplicação, foi utilizado uma arquitetura adaptada do MVC, uma vez que foi adicio-

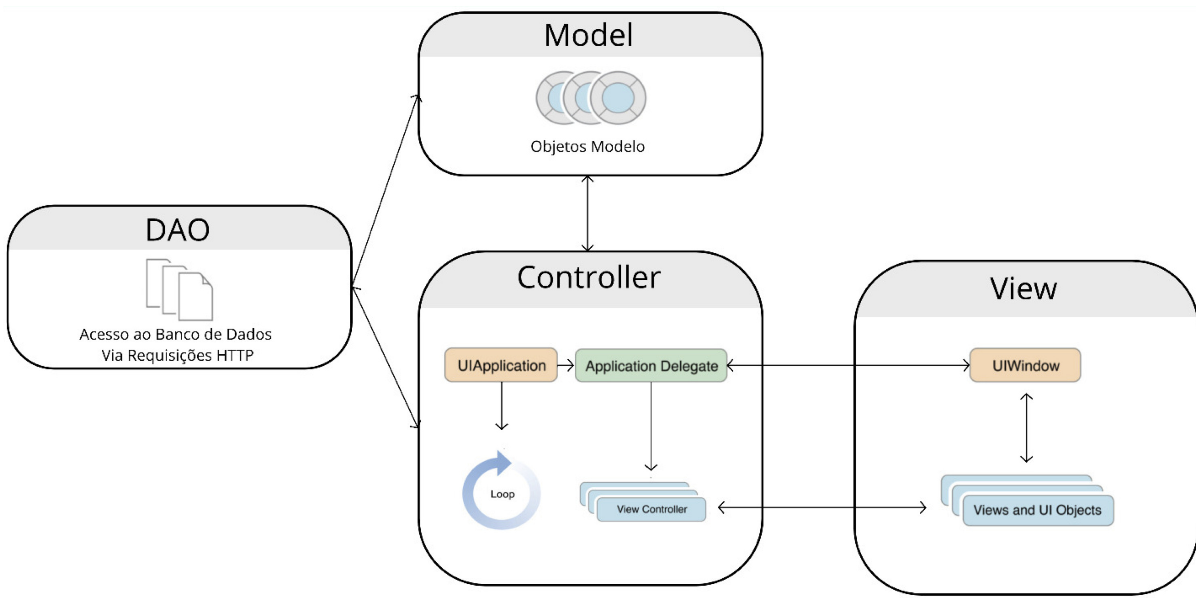

Figura 4 – Adaptação do MVC utilizada para a aplicação.

nada uma nova camada chamada de DAO (*Data Access Object*), que é responsável por se comunicar com o servidor de dados e enviar os dados para a camada da *Controller*. Então, em vez de utilizar as classes da Camada *Model* para acessar o banco de dados através das requisições HTTP na API, foi criado uma camada específica para este fim. Um esquema simplificado desta estrutura é apresentado na Figura 4.

Para leitores que estejam interessados em conher mais a fundo como funciona a arquitetura iOS e conhecer aspectos mais técnicos, é recomendada a leitura do Apêndice B, apresenta uma série de detalhes.

5.3.2.2 Banco de Dados

Segundo (APPLE, 2017), o CoreData é um *framework* que é usado para gerenciar os objetos da camada de modelo da aplicação. Esse *framework* fornece soluções automatizadas e generalizadas para tarefas comuns associadas ao ciclo de vida do objeto, incluindo a persistência.

Para esta aplicação, o CoreData foi utilizado, para o armazenamento local de informações do jogador. A API se concentra em armazenar apenas os dados referentes à evolução de cada jogador e um identificador para montar a tabela de "Melhores Jogadores" através do tratamento destes dados.

5.3.3 API

5.3.3.1 Breve Descrição

A API foi hospedada pelo Heroku (HEROKU, 2017), que é um servidor que oferece um plano gratuito e atende perfeitamente às demandas desta aplicação, porém este plano gratuito dá suporte apenas ao Banco de Dados Postgres, maiores detalhes podem ser encontrados em (POSTGRESQL, 2017). Essa limitação não foi um problema pois o Ruby on Rails também trabalha com esse banco de dados.

5.3.3.2 Arquitetura

Para a construção desta API, foi utilizado o padrão arquitetural do próprio Ruby on Rails, MVC, porém como é apenas uma aplicação API, não foi criada a camada View, que é responsável por apresentar os dados na Web. Esses dados são consumidos apenas pela aplicação desenvolvida.

5.3.4 Links Úteis

Tanto o código da aplicação quanto o da API estão hospedados no site Gitlab, para permitir a evolução do sistema em trabalhos futuros e ainda para dar oportunidade de outros programadores contribuírem com essa solução. É possível ter acesso a esses códigos através dos links: Aplicativo iOS e API em Ruby on Rails. A aplicação publicada na loja AppStore encontra-se disponível no link: *Quizz* AppStore.

6 Análise dos Resultados

6.1 Breve Resumo

De acordo com o que foi proposto como objetivo deste trabalho, foram levantadas algumas questões que serviriam de insumo para o *Quizz*. Desta forma, a partir do momento que algumas pessoas começassem a utilizar o aplicativo, se o fizerem da forma proposta, os resultados obtidos serviriam como indicativos das principais falhas deles. Por exemplo, ao final de cada rodada de perguntas, o próprio aplicativo apresenta os resultados obtidos, indicando os pontos fortes e fracos do jogador. A Figura 5 ilustra como o resultado é apresentado ao final da rodada de perguntas.

6.2 Resultados Obtidos

Foram inseridas 3 sessões contendo ao total 10 questões, o que, em geral, levaria em torno de 3 minutos para serem respondidas. Pela metodologia utilizada para o cálculo do total de pontos obtidos, cada pergunta tinha uma valor ponderado, mas, por ora, todas as perguntas receberam peso 1. Dessa maneira, cada acerto é igual a soma do seu peso e

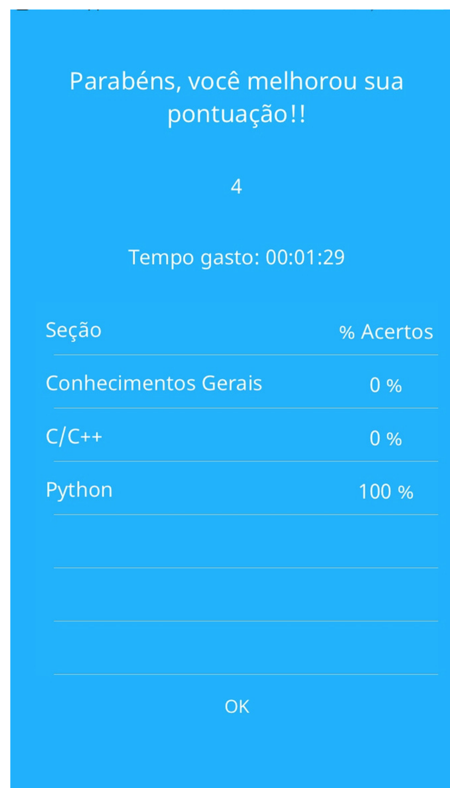

Figura 5 – Resultados obtidos após uma rodada de perguntas.

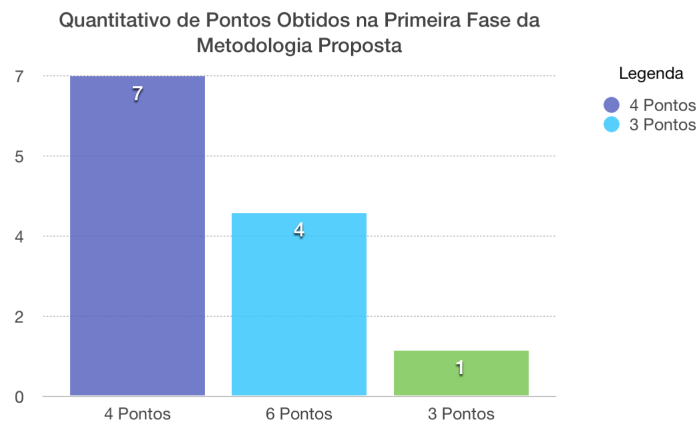

Figura 6 – Gráfico com os dados coletados.

Tabela 5 – Resultados obtidos na metodologia completa

	1ª Rodada	**2ª Rodada**
Jogador 1	4 Pontos	**8 Pontos**
Jogador 2	6 Pontos	**8 Pontos**

cada erro é igual a subtração do seu peso: uma pergunta errada anula uma certa. Ao final de cada sessão, o jogador tem seu percentual de acertos baseado neste quesito. Em média, o total possível a se obter seria 10 pontos, que representaria 100% de acerto em todas as questões. A Figura 6 apresenta os dados obtidos nesta primeira rodada de questões.

6.2.1 Participantes que concluíram toda a metodologia proposta

Durante a aplicação do *Quizz*, 2 jogadores se disponibilizaram para participar da metologia completa, ou seja, realizar uma primeira rodada de perguntas para traçar o perfil técnico, estudar o material proposto, de acordo com as indicações do protótipo (sessões que tiveram maior percentual de erros), e ainda, todo o conteúdo deste trabalho, e após isso, realizar outra rodada de perguntas.

O resultado é apresentado na Tabela 5. Desta forma, é possível verificar, ainda que em um espaço amostral reduzido, que o *Quizz* representa uma possibilidade de melhorar as capacidades técnicas dos jogadores.

6.3 Limitações

Devido ao custo do equipamento iOS e à dificuldade de encontrar jogadores disponíveis a participar de uma rodada de perguntas dentro do *Quizz*, não foi possível obter um espaço amostral com muitos jogadores. Houve 9 downloads direto da AppStore e alguns jogadores utilizaram celulares compartilhados, ou seja, utilizou-se o celular de outra

pessoa para responder o *Quizz*. Ao total, até o momento de apuração, 12 pessoas tinham respondido o *Quizz*.

A metodologia proposta era:

- realizar uma rodada de perguntas inicial para identificar as principais falhas e servir de parâmetro de evolução;
- estudar o texto de referência dentro do próprio aplicativo e
- realizar uma nova rodada de perguntas para ver o quanto o texto ajudou nos aspectos de segurança da informação.

Como os jogadores que participaram do experimento compareceram apenas à primeira rodada de perguntas, então, desta forma, ainda não foi possível avaliar o impacto do guia de boas práticas na evolução dos jogadores.

Como análise preliminar, foi possível mostrar que nenhum dos sujeitos submetidos ao *quizz* foi capaz de responder perfeitamente às respostas de todas as questões, sendo que a maioria dos usuários errou 3 questões na bateria inicial.

7 Conclusão

7.1 Considerações Finais

O fato de um programador conseguir construir um software em certa linguagem não garante a segurança dos dados que são manipulados. Alguns programadores simplesmente não conseguem medir o seu nível de adoção ou conhecimento das boas práticas de cada linguagem. O propósito deste trabalho, além de levantar alguns erros comuns que são cometidos por grande parte dos programadores, mesmo os experientes, foi criar uma ferramenta que conseguisse mensurar o nível de adesão dos princípios relacionados a segurança durante o desenvolvimento de um software.

Para realizar a medição da evolução do jogador, é necessário que ele realize o teste uma primeira vez antes de estudar o material disponibilizado; desta forma, espera-se ter um parâmetro inicial para futuras comparações. Após este passo, o próprio *Quizz* identifica em qual área que o jogador errou mais questões, onde supostamente, seria o principal ponto fraco. Após o estudo do material recomendado, é solicitado ao jogador que participe de uma nova rodada para avaliar a evolução a partir dos resultados obtidos anteriormente. É importante ressaltar que o jogador é livre para participar de quantas rodadas quiser, porém, se seguir o passo a passo indicado, a melhoria no seu desempenho deveria ser mais significativa.

Inicialmente o foco era apenas para desenvolvimendo *mobile*, porém durante as primeiras entregas deste trabalho, percebeu-se a necessidade de introduzir os conceitos de Sistemas Operacionais das plataformas tradicionais, pois os assuntos estão diretamente ligados e alguns dos principais problemas são compartilhados entre essas tecnologias. Desta forma, o escopo do trabalho aumentou e novos aspectos foram abordados.

Por fim, o desenvolvimento de software é uma área que está em constante evolução e estudos relacionados à linguagem e à plataforma alvo são essenciais e não devem ser negligenciados. Por exemplo, anualmente a Apple realiza um evento chamado WWDC onde são lançadas novas versões de dispositivos e linguagem. Maiores detalhes podem ser encontrados no site oficial do evento (APPLE, 2016a). Outras linguagens e plataformas também possuem uma periodicidade de lançamento, sem contar as atualizações críticas de segurança.

7.2 Limitações do Trabalho

Uma vez que as tecnologias e linguagens de programação estão em constante evolução, os materiais levantados para a criação deste trabalho muitas vezes estavam obsoletos, e algumas das afirmações que a bibliografia trazia já tinham sido corrigidos ou não estavam mais em uso. Então este é um trabalho que necessita de constante evolução e, de acordo com o *feedback* dos usuários, novos temas devem ser adicionados para que de fato este trabalho seja reconhecido como referência, pelo menos inicial, para os programadores que estão buscando aperfeiçoar suas habilidades no que diz respeito à programação segura.

Outra limitação deste trabalho é que o aplicativo *Quizz* foi criado apenas para usuários de dispositivos iOS. Uma solução para essa limitação é a criação de um módulo Web.

7.3 Trabalhos Futuros

Inicialmente, foi proposta a criação de um guia de boas práticas na programação voltada para a segurança das aplicações, acompanhada de um esquema para medir a absorção dos conteúdos tratados pelo guia através de um *Quizz*.

Desta forma, a seguir são propostos alguns trabalhos futuros, porém não exaustivos, que de poderiam agregar maior valor ao trabalho.

- Após reunir todo o material deste trabalho, é possível aplicar este teste em uma amostra de programadores e validar a sua eficácia, no que diz respeito a evolução do jogador;
- Adicionar e evoluir as perguntas contidas no *Quizz* e atualizar os temas;
- Aumentar a quantidade das linguagens de programação abordadas e adicionar questões referentes à cada uma com aspectos de segurança pode fazer com que este trabalho seja mais completo e atualizado, tendo em vista que as linguagens de programação também sofrem atualizações ao longo do tempo;
- Avaliar como os jogadores percebem e aplicam essas boas práticas na vida real, possivelmente através de um analisador de código voltado para este fim.

Referências

ABOUT iOS App Architecture. 2016. Disponível em: <https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html>. Acesso em: 10 nov. 2016. Citado 3 vezes nas páginas 96, 99 e 102.

ADMINISTRATION, N. S.; SECURITY. *SANS: Top Security Risks*. 2009. Disponível em: <https://www.cippguide.org/2011/05/31/sans-top-security-risks/>. Citado na página 27.

APPLE. *The Apple Worldwide Developers Conference*. 2016. Disponível em: <https://developer.apple.com/wwdc/about/>. Acesso em: 10 nov. 2016. Citado 2 vezes nas páginas 63 e 95.

APPLE. *Como usar o AirDrop com o iPhone, iPad ou iPod touch*. 2016. Disponível em: <https://support.apple.com/pt-br/HT204144>. Acesso em: 28 out. 2016. Citado 2 vezes nas páginas 40 e 95.

APPLE. *What Is Core Data?* 2017. Disponível em: <https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/index.html?utm_source=iosstash.io>. Acesso em: 25 mai. 2017. Citado na página 56.

APPLE Developer Program. 2016. Disponível em: <https://developer.apple.com/programs/enroll/>. Acesso em: 16 nov. 2016. Citado na página 103.

ARNOLD, D. N. *The Explosion of the Ariane 5*. 2000. Disponível em: <http://www-users.math.umn.edu/~arnold/disasters/ariane.html>. Acesso em: 25 abr. 2017. Citado na página 48.

BANKS, A.; EDGE, C. S. *Learning iOS Security*. [S.l.]: Packt Publishing, 2015. ISBN 1783551747, 9781783551743. Citado na página 97.

BLOQUEIO de Ativação do recurso Buscar iPhone. 2016. Disponível em: <https://support.apple.com/pt-br/HT201365>. Acesso em: 30 set. 2016. Citado na página 98.

CAELUM. *Apostila Java*. 2017. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java/#2-3-maquina-virtual>. Acesso em: 23 abr. 2017. Citado 2 vezes nas páginas 35 e 36.

CONTINUIDADE. 2016. Disponível em: <https://support.apple.com/pt-br/HT204681>. Acesso em: 10 nov. 2016. Citado na página 95.

DALE, N. *Programming and Problem Solving with Java*. Jones & Bartlett Learning, 2009. (Jones and Bartlett books in computer science). ISBN 9780763758271. Disponível em: <https://books.google.com.br/books?id=oq4gQJcZJXIC>. Citado na página 37.

DIETZ, W. et al. Understanding integer overflow in c/c++. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 25, n. 1, p. 2, 2015. Citado na página 48.

DINO. *Estatísticas do uso de celular no Brasil*. 2016. Disponível em: <http://exame.abril.com.br/negocios/dino/estatisticas-de-uso-de-celular-no-brasil-dino89091436131/>. Acesso em: 23 nov. 2016. Citado na página 28.

DOCUMENT Interaction. 2016. Disponível em: <https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/DocumentInteraction_TopicsForIOS/Introduction/Introduction.html#//apple_ref/doc/uid/TP40010403>. Acesso em: 16 nov. 2016. Citado na página 102.

DOCUMENT Interaction. 2016. Disponível em: <https://developer.apple.com/xcode/>. Acesso em: 16 nov. 2016. Citado na página 102.

DWIVEDI CHRIS CLARK, D. T. H. *Mobile Application Security*. 1a edição. ed. [S.l.]: McGraw-Hill Companies, 2010. ISBN: 978-0-07-163356-7, MHID: 0-07-163356-1. Citado 6 vezes nas páginas 41, 42, 44, 45, 49 e 95.

FAGET, D. *Onda de ciberataques atinge órgãos e empresas em ao menos 74 países*. 2017. Disponível em: <http://www1.folha.uol.com.br/mundo/2017/05/1883408-mega-ciberataque-derruba-sistemas-de-comunicacao-ao-redor-do-mundo.shtml>. Acesso em: 25 mai. 2017. Citado na página 44.

FIGUEIREDO, C. M.; NAKAMURA, E. Computação móvel: Novas oportunidades e novos desafios. *T&C Amazônia*, v. 1, n. 2, 2003. Citado na página 40.

FOUNDATION, F. S. *O Que é Copyleft*. 2017. Disponível em: <https://www.gnu.org/licenses/copyleft.pt-br.html>. Acesso em: 19 jun. 2017. Citado na página 32.

GITHUB. *GitHub is how people build software*. 2017. Disponível em: <https://github.com/about>. Acesso em: 15 abr. 2017. Citado na página 33.

GOMES, H. S. *Apps para smartphone se tornam canal nº 1 de bancos brasileiros*. 2016. Disponível em: <http://g1.globo.com/tecnologia/noticia/2016/03/apps-para-smartphone-se-tornam-canal-n-1-de-bancos-brasileiros.html>. Acesso em: 9 mai. 2017. Citado na página 28.

GOMES, L. R. et al. *Segurança no Desenvolvimento de Software*. [S.l.]: SERPRO - Serviço Federal de Processamento de Dados, 2014. Citado 5 vezes nas páginas 27, 43, 45, 50 e 52.

GRIMES, R. A. *Bash Bunny: Big hacks come in tiny packages*. 2017. Disponível em: <http://www.infoworld.com/article/3192084/security/bash-bunny-big-hacks-come-in-tiny-packages.html>. Acesso em: 9 mai. 2017. Citado 2 vezes nas páginas 27 e 49.

HEROKU. *Learn about building, deploying and managing your apps on Heroku*. 2017. Disponível em: <https://devcenter.heroku.com>. Acesso em: 25 mai. 2017. Citado na página 57.

HOWARD, M.; LIPNER, S. *The security development lifecycle: SDL, a process for developing demonstrably more secure software*. Microsoft Press, 2006. (Microsoft Press Series). Disponível em: <https://books.google.com.br/books?id=-kNGAQAAIAAJ>. Citado na página 27.

ICLOUD: Restaurar ou configurar seu dispositivo iOS do iCloud. 2016. Disponível em: <https://support.apple.com/kb/PH12521?locale=en_US&viewlocale=pt_BR>. Acesso em: 7 nov. 2016. Citado na página 98.

INC, A. *Apple Reinvents the Phone with iPhone*. 2007. Disponível em: <http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html>. Acesso em: 27 out. 2016. Citado na página 39.

INTER-APP Communication. 2016. Disponível em: <https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Inter-AppCommunication/Inter-AppCommunication.html>. Acesso em: 16 nov. 2016. Citado na página 102.

IONIC. 2016. Disponível em: <http://ionicframework.com>. Acesso em: 16 nov. 2016. Citado na página 103.

IOS Design Patterns. 2016. Disponível em: <https://www.raywenderlich.com/46988/ios-design-patterns>. Acesso em: 16 nov. 2016. Citado na página 101.

IOS Human Interface Guidelines. 2016. Disponível em: <https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/>. Acesso em: 16 nov. 2016. Citado na página 103.

IPHONE 7 - Visão Geral. 2016. Disponível em: <https://www.apple.com/br/iphone-7/ios/>. Acesso em: 10 nov. 2016. Citado na página 95.

MARTIN, B. et al. *2011 CWE/SANS Top 25 Most Dangerous Software Errors*. 2011. Disponível em: <http://cwe.mitre.org/top25/index.html>. Acesso em: 19 mai. 2017. Citado 3 vezes nas páginas 45, 50 e 52.

MICROSOFT. *Windows lifecycle fact sheet*. 2017. Disponível em: <https://support.microsoft.com/en-us/help/13853/windows-lifecycle-fact-sheet>. Acesso em: 23 abr. 2017. Citado na página 33.

MIECZNIKOWSKI, J.; HENDREN, L. Decompiling java bytecode: Problems, traps and pitfalls. In: ______. *Compiler Construction: 11th International Conference, CC 2002 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8–12, 2002 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. p. 111–127. ISBN 978-3-540-45937-8. Disponível em: <http://dx.doi.org/10.1007/3-540-45937-5_10>. Citado na página 36.

MITRE. *Corporate Overview*. 2017. Disponível em: <https://www.mitre.org/about/corporate-overview>. Acesso em: 19 mai. 2017. Citado na página 50.

NETWORKING, S. A.; SECURITY. *SANS About*. 2017. Disponível em: <https://www.sans.org/about/>. Acesso em: 19 mai. 2017. Citado na página 50.

ONE, A. *Smashing The Stack For Fun And Profit*. 1996. Disponível em: <http://phrack.org/issues/49/14.html>. Acesso em: 24 abr. 2017. Citado na página 47.

ORACLE. *Understanding Signing and Verification*. 2015. Disponível em: <https://docs.oracle.com/javase/tutorial/deployment/jar/intro.html>. Acesso em: 2 mai. 2017. Citado na página 37.

OWASP. *Welcome to OWASP*. 2017. Disponível em: <https://www.owasp.org/index.php/Main_Page>. Acesso em: 29 mai. 2017. Citado na página 52.

POSTGRESQL. *About Postgresql*. 2017. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 25 mai. 2017. Citado na página 57.

PROGRAM Membership Details. 2016. Disponível em: <https://developer.apple.com/programs/whats-included/>. Acesso em: 16 nov. 2016. Citado na página 103.

PROGRAMMING with Objective-C. 2016. Disponível em: <https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>. Acesso em: 10 nov. 2016. Citado 2 vezes nas páginas 96 e 102.

PROJECT, O. A. T. O. W. A. S. *OWASP Top 10 Application Security Risks - 2017*. 2011. Disponível em: <https://www.owasp.org/index.php/Top_10_2017-Top_10>. Acesso em: 19 jun. 2017. Citado na página 45.

ROCHA, A. M.; NETO, R. M. F. Introduç aoa arquitetura apple ios. 2014. Citado na página 96.

ROHAN. *Bluetooth Smart and Smart Ready Market worth 5.57 billion dollars by 2020*. 2016. Disponível em: <http://www.marketsandmarkets.com/PressReleases/bluetooth-smart-ready.asp>. Acesso em: 5 nov. 2016. Citado na página 40.

ROHR, A. *Apps de bancos brasileiros têm deficiências de segurança, diz pesquisa*. 2015. Disponível em: <http://g1.globo.com/tecnologia/blog/seguranca-digital/post/apps-de-bancos-brasileiros-tem-deficiencias-de-seguranca-diz-pesquisa.html>. Acesso em: 9 mai. 2017. Citado na página 28.

ROMERO, T. *Garbage Collection Entendendo e otimizando*. 2017. Disponível em: <http://www.devmedia.com.br/garbage-collection-entendendo-e-otimizando-parte-1/24082>. Citado na página 37.

RUBYONRAILS. *Imagine what you could build if you learned Ruby on Rails*. 2017. Disponível em: <http://rubyonrails.org>. Acesso em: 25 mai. 2017. Citado na página 55.

SACCOL, A. Z.; REINHARD, N. Tecnologias de informação móveis, sem fio e ubíquas: definições, estado-da-arte e oportunidades de pesquisa. *Revista de administração contemporânea*, SciELO Brasil, v. 11, n. 4, p. 175–198, 2007. Citado 2 vezes nas páginas 39 e 40.

SANTOS, M. M. Vulnerabilidade climática e consumo de energia elétrica em áreas urbanas. Citado na página 97.

SEARCH, A. *Bluetooth History*. 2016. Disponível em: <https://www.bluetooth.com/media/our-history>. Acesso em: 28 out. 2016. Citado na página 41.

SELVARAJ, C.; ANAND, S. A survey on security issues of reputation management systems for peer-to-peer networks. *Computer Science Review*, Elsevier, v. 6, n. 4, p. 145–160, 2012. Citado na página 48.

SILBERSCHATZ, A.; GALVIN, P.; GAGNE, G. *Operating System Concepts Essentials, 2nd Edition: Second Edition*. John Wiley & Sons, 2013. ISBN 9781118844007. Disponível em: <https://books.google.com.br/books?id=TIRbAgAAQBAJ>. Citado 3 vezes nas páginas 32, 45 e 46.

SWIFT. 2016. Disponível em: <https://swift.org>. Acesso em: 10 nov. 2016. Citado 3 vezes nas páginas 55, 95 e 102.

TANENBAUM, A. S.; WOODHULL, A. S. *Sistemas Operacionais: Projetjos e Implementação*. [S.l.]: Bookman Editora, 2009. Citado na página 31.

UNAUTHORIZED modification of iOS can cause security vulnerabilities, instability, shortened battery life, and other issues. 2016. Disponível em: <https://support.apple.com/en-us/HT201954>. Acesso em: 10 nov. 2016. Citado na página 95.

WATCH. 2016. Disponível em: <http://www.apple.com/br/watch/>. Acesso em: 10 nov. 2016. Citado na página 95.

WHEELER, D. A. *Secure Programming HOWTO*. [S.l.: s.n.], 2015. Citado na página 43.

YOUNG, K. *Fast-growth nations clock up the most hours for mobile web usage*. 2015. Disponível em: <http://www.globalwebindex.net/blog/fast-growth-nations-clock-up-the-most-hours-for-mobile-web-usage>. Acesso em: 23 nov. 2016. Citado na página 28.

Apêndices

APÊNDICE A – Questionário

A.1 Metodologia Utilizada

É importante ressaltar que para a criação das perguntas foram utilizados os conceitos apresentados neste documento e que forma retirados da bibliografia que possuem certo reconhecimento na área de programação defensiva.

Algumas perguntas são redundantes, ou seja, a mesma coisa é perguntada de forma diferente afim de evitar que o questionário seja aplicado sem necessariamente traçar o perfil do jogador.

O objetivo é realmente tentar perceber o quanto cada programador sabe de cada assunto. Desta forma, o questionário foi dividido em seções específicas. A primeira parte, A.2.1, busca analisar os conhecimentos de forma geral, sem um assunto delimitador, todo este documente serviu como referência. A segunda A.2.2, a terceira parte A.2.3, e a quarta A.2.4 parte apresentam, perguntas específicas sobre conceitos de programação defensiva relacionadas às linguagens de programação C/C++ e Python, respectivamente.

A.2 Perguntas e Respostas

A seguir são apresentadas as perguntas propostas que são utilizadas como insumo para a aplicação do questionário e são utilizadas no *Quizz*.

A.2.1 Conceitos sobre conhecimentos gerais

Para a criação das perguntas desta sessão, todo o texto foi considerado. Sem focar em um assunto específico, o objetivo destas perguntas é nivelar o conhecimento do usuário em relação à programação defensiva em geral.

O conceito de Desenvolvimento Seguro é tão grande e completo quanto o desenvolvedor desejar, dependendo da importância dada, e dos conceitos que são utilizados. Sabe-se que existem alguns princípios, que em geral, são aceitos pela comunidade, dos quais todo programador que tem essa preocupação de desenvolvimento seguro deve ter em mente durante o ciclo de vida do software.

1. Com isso em mente, marque a alternativa correspondente à alguns princípios relacionados à segurança.

   a Confidencialidade, Integridade e Disponibilidade

b Senhas, Antivírus e Criptografia

c Senhas, Firewall e Design Patterns

d Todas alternativas anteriores

Resposta: a.
Justificativa: Frequentemente objetivos de segurança são descritos em termos de 3 objetivos: ***Confidencialidade****: Somente pessoas autorizadas devem ter acesso à informações privadas;* ***Integridade****: As informaçõs só podem ser modificados ou apagados por pessoas e formas autorizadas ; e* ***Disponibilidade****: As informações devem estar acessíveis à partes autorizadas em tempo hábil.*

2. É correto afirmar que Desenvolvimento Seguro é um conceito que deve está presente durante todo o ciclo de vida de desenvolvimento do software?

   a Verdadeiro

   b Falso

   Resposta: Verdadeiro.
   Justificativa: O esforço de fazer um programa seguro desde o início é o que pode facilitar ou atrapalhar este objetivo. É recomendado que desde o início do desenvolvimento, o programado tenha essa preocupação em mente.

3. É correto afimar que para garantir que um software que está dentro de um ecosistema somente é seguro se todo o ecossistema também for?

   a Verdadeiro

   b Falso

Resposta: Verdadeiro.
Justificativa: A interface de um software seguro com outros que não são, pode comprometer as informações e ferir os princípios de segurança.

Após a explosão da Internet, é possível ver quase que diariamente falhas em softwares, crimes cibernéticos, roubo de informações sigilosas, etc, e tudo isso pode causar inúmeros prejuízos financeiros. E em geral, esse tipo de crime é possível, por erros comuns cometidos por programadores, erros esses que veem sendo repetidos durante décadas. Ou seja, a necessidade de se ter uma visão mais crítica sobre desenvolvimento seguro é latente. Até porque, nos dias atuais, software está presente em tudo, desde a geladeira até o computador de bordo do carro.

4. Apesar de a maioria dos programadores cometerem esses erros sem a intenção de deixar o software vulnerável, selecione a alternativa que melhor explica essa afirmação.

a A maioria das aulas de programação, ao tentar facilitar o aprendizado, não abordam conceitos de segurança, apesar disso existem livros e cursos voltados para a segurança das aplicações.

b As linguagens de programação realmente são inseguras então não há o que se fazer, uma vez que elas não se atualizam com a frequência ideal.

c É caro criar um software seguro, então os programadores não se preocupam com a segurança pois estão preocupados em desenvolver as funcionalidades.

d Não precisa se preocupar se houver problemas de segurança na aplicação.

Resposta: a.
Justificativa: Durante o aprendizado é mais conveniente abordar questões mais simples, e alguns programadores podem ter a sensação de estarem fazendo a coisa certa, sem sequer se preocupar com questões de segurança, para resolver esse problema, alguns autores que estão percebendo tendências, vem publicando livros que motivam os programadores à se preocuparem com essas questões.

Julgue com verdadeiro ou falso as afirmações abaixo.

5. As principais falhas encontradas em softwares, podem ser corrigidas apenas adicionando uma fase de "procura por falhas" no final do ciclo de vida de desenvolvimento do software.

   a Verdadeiro

   b Falso

   Resposta: Falso.
   Justificativa: Erros acontecem, os programdores são humanos, e quanto mais rápido forem descobertos, mais simples e barato será para corrigi-los.

6. É correto afirmar que algumas dessas vulnerabilidades encontradas são de softwares legados que eram usados apenas em ecossistemas internos, ou seja, não eram conectados à outros programas pela Internet. Dessa forma, durante o seu desenvolvimento, não era necessário ter o cuidado com segurança, e de uma hora pra outra passaram a transmitir e receber informações através da Internet.

   a Verdadeiro

   b Falso

   Resposta: Verdadeiro.
   Justificativa: Programas que até então eram feitos para serem executados apenas localmente, agora precisam forcener interfaces para troca de informações com outros software. Com isso, novas questões de segurança podem surgir.

7. Todas as possíveis falhas e vulnerabilidades já foram encontradas catalogadas e existem uma medida padrão que deve ser tomada caso alguma seja identificada. Essa medida padrão está disponível com exemplos, para as 20 linguagens de programação mais utilizadas.

   a Verdadeiro

   b Falso

   Resposta: Falso.
   Justificativa: O NIST que é o responsável por padronizar essas questões de tecnologia, começou um trabalho para padronizar e categorizar esses erros classicos, porém, novos erros e falhas podem surgir.

8. Tendo em vista todos os problemas que a Internet trouxe para a segurança da informação, é correto afirmar que o problema está na Internet, e não nos softwares desenvolvidos.

   a Verdadeiro

   b Falso

   Resposta: Falso.
   Justificativa: A Internet trouxe novas possibilidades para os software, e uma evolução para acompanhar é necessária. Porém o problema ainda continua nos softwares.

9. Se o programador seguir todas as convenções da linguagem utilizada, é correto afirmar que o software que ele desenvolveu estará totalmente livre de falhas e vulnerabilidades.

   a Verdadeiro

   b Falso

   Resposta: Falso.
   Justificativa: As convenções devem ser seguidas, porém somente isso não garante que um software seja seguro ou não.

10. Durante o desenvolvimento de certos tipos de programas, o desenvolvedor não precisa se preocupar com questões de segurança. Por exemplo: Jogos para *Smartphones*, Interface de arquivos na nuvem, Software para gerência de estoques de uma loja familiar, Servidores Locais, etc.

    a Verdadeiro

    b Falso

Resposta: Falso.
Justificativa: Independente do tipo de aplicação, se ela tratar dados sensíveis, questões de segurança devem ser discutidos. Claro que alguns software são bem mais críticos que outros, a depender da natureza do software. Porém todo software que é lançado no mercado significa uma nova possibilidade de ataque. Seja para burlar a licença, para software pagos ou até mesmo para roubar informações de softwares que tratam dados sensíveis.

11. Existem programas que testam o software e acabam com a possibilidade de existir qualquer tipo de falha no software.

    a Verdadeiro

    b Falso

    Resposta: Falso.
    Justificativa: Esse tipo de programa busca por falhas comuns, porém não significa que o software estará livre de falhas. Nenhum software é capaz de deixar outro software totalmente seguro.

12. Pessoas não confiáveis podem ter acesso de letura e escrita à algumas pastas públicas, é seguro criptografar a informação e salvar nessas pastas?

    a Verdadeiro

    b Falso

    Resposta: Falso.
    Justificativa: Apesar de estarem criptografas, as informações continuam lá o correto é salvar em locais apropriados para este fim.

13. Utilizar bibliotecas criadas por terceiros (Gems, CocoaPods, Library, ...) pode representar uma vulnerabilidade no software desenvolvido?

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Apesar de ser uma boa prática de programação, o reuso de software pode significar risco, então o correto a se fazer é verificar a fonte, e se possível verificar o código para garantir que ele não significa uma ameaça.

14. A arquitetura utilizada para construção do software pode trazer consigo algumas vulnerabilidades independente da linguagem de programação utilizada.

    a Verdadeiro

b Falso

Resposta: Verdadeiro.
Justificativa: O ideal é seguir a convenção da linguagem, ou ainda, imaginar estratégicamente uma arquitetura que aumente a segurança do software, principalmente em aplicações web.

15. A arquitetura do software deve variar conforme a finalidade da aplicação.

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Cada sistema é único devido à natureza do negócio que ele suporta, então para cada plataforma, tecnologia, ou linguagem existe uma arquitetura que atende questões de segurança, compatibildiade, extensibilidade, confiabilidade, manutenabilidade, etc.

16. Historicamente, um dos pontos mais críticos em relação à ataques está na entrada de dados (*inputs*).

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Entradas de dados não tratadas são consideradas pontos chave no que diz respeito à invasão de segurança.

17. *SQLInjection* não funciona com as melhorias que as linguagens vem trazendo ao longo dos anos.

    a Verdadeiro

    b Falso

    Resposta: Falso.
    Justificativa: Apesar de ser um ataque clássico, as entradas de dados não tratadas são consideradas pontos cruciais para exploração da segurança.

18. Historicamente, o Microsoft Windows é o Sistema Operacional mais vunerável.

    a Verdadeiro

    b Falso

Resposta: Falso.
Justificativa: O Microsoft Windows é o sistema que certa de 90% da população mundial utiliza, então para os hackers é mais conveniente criar vírus para esta plataforma, e ainda ele foi desenvolvido para ser amigável e prático para os usuários, por exemplo, processos que são automáticos sem a necessidade de intervenção do usuário, pode ser um ponto crítico.

19. Não existem vunerabilidades relacionadas às plataformas móveis(Android, iOS) e nem no que é chamado de IoT (*Internet of Things*).

   a Verdadeiro

   b Falso

   Resposta: Falso.
   Justificativa: Vunerabilidades ocorrem em qualquer plataforma, principalmente com o surgimento dos Smartphones, onde os usuários estão migrando do computador pessoal e fazendo tudo que precisam na palma da mão, direto do próprio dispositivo móvel.

20. Bluetooth, WiFi, Dados Móveis E-mail e Redes Locais representam riscos à segurança dos computadores que estão conectados.

   a Verdadeiro

   b Falso

   Resposta: Verdadeiro.
   Justificativa: Apesar de cada plataforma ter seu próprio mecanismo de segurança, interfaces de conexões sempre significam riscos, por isso é importante estar atento às questões de segurança.

21. Padrão UTF-8 tenta codificar os textos na menor quantidade de *bytes* possível. Isso pode significar um risco de segurança.

   a Verdadeiro

   b Falso

   Resposta: Verdadeiro.
   Justificativa: Os codificadores UTF-8 devem usar a codificação "mais curta possível", mas os descodificadores ingênuos podem aceitar codificações que são mais longas do que o necessário, isso significa que pode ser utilizado como input por pessoas não confiáveis.

22. A manipulação de arquivos comprimidos advindos de fontes não confiáveis pode representar um risco para todo o sistema.

a Verdadeiro

b Falso

Resposta: Verdadeiro.
Justificativa: Arquivos comprimidos, em geral, não costumam passam por verificação da mesma forma que um arquivo não comprimido. Dessa forma, se a fonte não for confiável, sua manipulação pode representar um risco. Por exemplo, existe uma espécie de ataque chamada "Zip Bomb" que é projetado para travar ou tornar inútil o todo o sistema, frequentemente utilizado para desativar o antivírus. Esse tipo de arquivo, em geral, possui um tamanho comprimido muito pequeno; porém, durante sua decodificação com auxílio de um descompressor não seguro, algumas instruções de recursividade podem tornar o arquivo muito grande e consome integralmente o espaço disponível em memória de persistência.

A.2.2 C, C++

Uma vez que as linguagens C/C++ apresentem características semelhantes, esta seção apresenta conceitos relacionados à ambas.

A Seção 2.3.1 serviu como principal fonte de informações para a criação das perguntas desta seção. Não obstante, conceitos que foram indiretamente citados neste documento também fazem parte da fonte de informações utilizada.

C, e C++ possem sérios problemas de segurança, porém é possível sim, escrever códigos seguros utilizando essa linguagem. Como essas linguagem exigem que os programadores construam mecanismos para gerenciamento de memória(utilizando malloc(), alloc(), free(), new e delete) *buffer overflows* são comuns. E o C tem uma fraqueza que não permite excessões, como outras linguagens muito utilizadas.

23. Liberar um espaço de memória que não deveria ser liberado pode significar uma falha grave. Selecione a opção que representa o potencial perigo se isso acontecer.

    a Como é apenas liberando memória, o programa fica mais rápido, e mais difícil para captar tentativas de invasão.

    b Isso permite que um invasor execute algum código arbitrário naquele espaço de memória.

    c Liberar espaço que não deveria ser liberado significa que pode faltar memória para o programa continuar sendo executado.

    d Faz com que o programa reaja de formas inesperadas.

Resposta: b.
Justificativa: O problema mais sério é que os programas podem erroneamente liberar memória que não deve ser liberada (por exemplo, porque já foi liberada). Isso pode resultar em uma falha imediata corrompendo outros espaços de memória já reservados ou os que ainda irão ser alocados.

24. Não liberar um espaço de memória que não está mais sendo utilizado, pode siginifcar um risco?

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Não liberar memória que não está sendo mais utilizada pode significar um risco para a aplicação, uma vez que haverá um acumulo desnecessário e se a memória não for suficiente, a aplicação pode parar de responder. Com isso, atacantes podem criar casos de negação para a aplicação, ainda que seja de baixo risco, é sim um risco.

25. Como o C/C++ são linguagens tipadas, a declaração de uma variável errada pode representar riscos?

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Principalmente tipos que permitem unsigned, tais como int, char, etc. O ideal mesmo é utilizar enum sempre que possível, e se for o caso, declare que a variável é unsigned.

26. É boa prática sempre ativar o maior número possível de avisos no compilador, tais como -*Wall, -Wpointer-arith -Wstrict-prototypes -O2, etc.*.

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Se houver uma possibilidade de falha, é sempre melhor descobrir enquanto o programa está em desenvolvimento do que quando é enviado para produção. Pode levar algum esforço para tornar os programas existentes em conformidade com todas essas verificações, mas essas verificações também podem ajudar a encontrar alguns problemas

Considere o trecho de código a seguir:

```
#include <string.h>
int main(){
   char * st;

   char *hw = "Oi\n";

   strcpy(st, "abc");
   printf("%s", *hw);
   return 0;
}
```

27. É correto afirmar que ele irá gerar um erro de “Segmentation fault/coredump error” no UNIX/Linux ou “general protection fault” no Windows, ou ainda algum outro erro?

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    *Justificativa: Sempre que um ponteiro for declarado, é necessário reservar um espaço de memória pra ele. Uma possível solução seria iniciar o ponteiro com “char * st[20];”, se já souber o tamanho necessário, ou utilizar a função malloc(), e depois utilizar o free() para liberar espaço não utilizado em memória.*

28. A função print irá escrever na tela a seguinte frase: “Oi”, pois está correta.

    Resposta: Falso.
    Justificativa: O “” pode servir tanto para diferenciar o ponteiro quanto para encontrar o endereço na memória, neste caso, não é necessário o seu uso. Pois ele resultaria no erro “Segmentation Fault” .*
    Uma possível solução para este código seria:

```
#include <string.h>
#include <stdlib.h>
int main(){
   char *st = malloc(20);

   strcpy(st, "abc");
   free(st);
```

```
    printf("%s", hw);
    return 0;
}
```

Durante a programação inúmeras variáveis são definidas, seja para controle, armazenamento de dados, troca de informações, etc. No caso da Linguagem C, existem vários tipos de variáveis que podem ser declaras, por exemplo, int, char, float, double, etc. No caso dos tipos que representam números, sejam inteiros ou decimais, existe ainda a necessidade de saber se ele pode assumir ou não valores negativos. Muitos programadores esquecem de definir quando uma variável é *unsigned*, pois por padrão ela é *signed*. O que pode ocasionar alguns erros e falhas comuns. Dada essa afirmação, julgue os itens abaixo.

29. Uma variável do tipo *Integer* basicamente é uma região de memória capaz de armazenar um valor inteiro com tamanho máximo de até 4 *bytes*. Considerando um computador com arquitetura 64 bites, é correto afirmar que se uma variável utiliza todos os 64 bits, em uma máquina que suporta apenas 32 bits o programa não responderá conforme esperado.

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Se o valor de uma variável ultrapassa a quantidade máxima de bites que o computador suporta, certamente o programa não irá responder da forma esperada. Em geral, o erro correspondente é o Segmentation Fault.

30. Uma variável do tipo integer, por *default* é *signed*, ou seja, pode assumir tanto valores positivos quanto negativos. É correto afirmar que uma variável *unsigned integer* ocupará metade do espaço de uma variável do tipo *integer*?

    a Verdadeiro

    b Falso

    Resposta: Falso.
    Justificativa: O tamanho máximo de uma variável integer varia entre -32,768 à 32,767 (2 bytes) ou -2,147,483,648 à 2,147,483,647 (4 bytes). Enquanto que o tamanho máximo de uma variável do tipo unsigned integer varia de 0 à 65,535 ou 0 à 4,294,967,295. Ou seja, o tamanho é sempre o mesmo, o que muda é que metade da parte positiva é dividida com a parte negativa. No fundo ambas tipagens podem assumir valores de 2 ou 4 bytes.

31. É correto afimar que alguns dos interpretadores da Linguagem C já estão sendo otimizados para evitar erros de *Overflow*?

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Mecanismos como ASLR (Address Space Layout Randomisation), DEP (Data Execution Prevention), etc. Estão sendo utilizados buscando evitar que esse tipo de erro aconteca. Mas é sempre bom estar atento à esses possíveis tipos de erro e tentar evitar que eles ocorram.

32. As pessoas utilizam a base decimal como padrão (0-9) já os computadores utilizam a base binária (0-1). Durante a conversão de base, o que diferencia o número 2 do -2 na base binária é o conceito de bit mais significativo(MSB).

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Considerando um computador de 4 bites, o número 2 é representado por 00010, enquanto que o número -2 é representado por 10010. Considerando que o bit mais à esquerda, é o bit de sinal.

33. *Integer Overflow* acontece quando uma operação aritmética resulta em um valor que não pode ser representado naquele tipo específico de variável.

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Cada tipo de variável tem um tamanho máximo, se o resultado de um cálculo aritmético estiver fora do intervalor aceito, por exemplo, se precisar de um tamanho maior do que o que a variável suporta, ou menor do que o tamanho mínimo.

34. Uma condição de *Overflow* sempre irá fazer o aplicativo "quebrar".

    a Verdadeiro

    b Falso

    Resposta: Falso.
    Justificativa: Uma condição de estouro dá resultados incorretos e, particularmente

se a possibilidade não foi antecipada, pode comprometer a confiabilidade de um programa e segurança. Sem necessariamente fazer com que o programa interrompa a execução. Em alguns processadores, sempre que acontece um overflow, ele retorna o valor inteiro máximo, sem que este seja de fato o resultado esperado.

35. Na linguagem C quando o programador é responsável pelo gerenciamento de memória, se um espaço de memória for liberado mais de uma vez, não há problema algum, inclusive, é indicado que existam redundâncias, no que diz respeito à liberação de memória que não está mais sendo usada.

    a Verdadeiro

    b Falso

    Resposta: Falso.
    Justificativa: Quando um espaço de memória é liberado mais de uma vez, o programa responde com um comportamento indefinido. Na prática, dupla liberaçao de um bloco de memória corromperá o estado do gerenciador de memória da aplicação, o que pode fazer com que outros blocos de memória sejam corrompidos ou falhar em alocações futuras.

Considere o trecho de código abaixo.

```
#include <math.h>
int main(){
   char *ptr;
   char *ptr1;
   ptr=malloc(5 * sizeof(*ptr));
   ptr1=ptr;
   free(ptr);
   free(ptr1);
}
```

Julgue os itens abaixo como verdadeiro ou falso.

36. Não existe a condição de *Double Free* uma vez que ponteiros diferentes chamam a função free().

    a Verdadeiro

    b Falso

Resposta: Falso.
Justificativa: Independente do nome dado aos ponteiros, ambos apontam para o mesmo espaço de memória.

37. Após ter liberado o ponteiro 'ptr' eu posso utilizar o ponteiro 'ptr1' normalmente.

    a Verdadeiro

    b Falso

    Resposta: Falso.
    Justificativa: Quando o 'prt' chama a função free(), não existe mais o espaço de memória previamente alocado, então para que seja possível utilizar o ponteiro 'ptr1' é necessário realocar a memória.

38. Ambos ponteiros 'prt' e 'ptr1' apontam para o mesmo espaço de memória.

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Quando o ponteiro 'ptr1' recebe o ponteiro 'ptr' ambos apontam para o mesmo espaço de memória.

39. É correto afirmar que como a maior parte das linguagens de programação, a Linguagem C fornece uma séria de mecanismos de segurança para os programadores, muitos deles automáticos, tais como tratamento de exceções, gerenciamento de memória automático, etc.

    a Verdadeiro

    b Falso

    Resposta: Falso.
    Justificativa: Uma das características da Linguagem C é que ela não dá suporte para o tratamento de exceções e os programadores devem criar seus próprios mecanismos para o gerenciamento de memória.

A.2.3 Python

A seção 2.3.3 serviu como base para a criação das perguntas, porém a maior parte das perguntas foram retiradas da blibliografia juntamente com a Internet e fóruns de desenvolvedores Python.

Python é uma linguagem de programação interpretada, orientada a objetos e de alto nível, e que exige poucas linhas de código se comparado ao mesmo programa em outras linguagens. O Python suporta módulos e pacotes, incentivando assim a modularidade do programa e a reutilização de código. Essas características tornam ele muito atraente e amplamente utilizada.

Tendo como referência a Linguagem Python bem como os critérios de boas práticas de programação, julgue os itens abaixo como verdadeiro ou falso.

40. Python é uma linguagem de programação "fortemente tipada", ou seja, na declaração das variáveis e métodos é necessário deixar explícito o tipo desejado.

    a Verdadeiro

    b Falso

    Resposta: Falso.
    Justificativa: Apesar de ser de tipagem dinâmica, o Python é de tipagem forte pois os valores e objetivos tem tipos bem definidos e não sofrem cast como C, ou outras linguagens.

41. O Python oferece uma ferramenta que pode aumentar a produtividade dos programadores: a metaprogramação.

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Uma das vantagens do Python é que ele oferece sim esse suporte. Metaprogramação é a criação de programas que escrevem ou manipulam outros programas (ou a si próprios) assim como seus dados, ou que fazem parte do trabalho em tempo de compilação. Em alguns casos, isso permite que os programadores sejam mais produtivos ao evitar que parte do código seja escrita manualmente.

42. No conceito de metaprogramação, Python pode ser considerada tanto metalinguagem quanto linguagem objeto.

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: A linguagem em que o metaprograma é escrito é chamada metalinguagem. A linguagem dos programas que são manipulados é chamada linguagem objeto.

A habilidade de uma linguagem de programação de ser sua própria metalinguagem é chamada reflexão. A reflexão facilita a metaprogramação, assim como ter uma linguagem de programação que é um tipo de dado de primeira classe de si mesma.

43. Selecione a opção que corresponde à finalidade da função eval() em Python.

   a É uma função que permite a execução de um código Python dentro dele mesmo.

   b É uma função matemática que verifica se o resultado é o esperado.

   c É uma alternativa à função *if .. else*.

   d Faz o *cast* de uma string em outro tipo de variável.

   Resposta: a.
   *Justificativa: eval() interpreta uma string como código. A razão pela qual tantas pessoas têm avisado sobre o uso deste é porque um usuário pode usar isso como uma opção para executar o código no computador. Se você tiver eval (input ()) e os importados, uma pessoa poderia digitar em input () os.system ('rm -R *') que apagaria todos os seus arquivos em seu diretório home. (Assumindo que você tem um sistema unix). Usar a função eval() sem consciência do que está fazendo pode representar um risco. Se você precisa converter strings para outros formatos, tente usar as coisas que fazem isso, como int()..*

Considere o trecho de código a seguir:

```
>>> def foo(bar=[]):
...     bar.append("xyz")
...     return bar
```

44. É correto afirmar que a chamada desse método sem a passagem de algum parâmetro irá retornar um array com o elemento "xyz" na posição 0?

   a Verdadeiro

   b Falso

   Resposta: Verdadeiro.
   Justificativa: Em Python é possível declarar funções com parâmetros opcionais. A chamada dessa função sem a passagem de qualquer parâmetro irá resultar em um array com um elemento na posição 0 cujo valor é igual à "xyz".

45. É correto afirmar que a chamada desse método mais de uma vez sempre irá resultar em um array com o elemento "xyz" na posição 0?

a Verdadeiro

b Falso

Resposta: Falso.
Justificativa: Um problema em Python faz com que o valor padrão para um argumento de função somente é avaliado uma vez, no momento em que a função é definida. Caso a função seja chamada mais de uma vez, cada vez que a função for chamada, o valor "xyz" será adicionado ao array de retorno. Um exemplo com a explicação do que acontece é apresentado a seguir:"

```
>>> foo()
["xyz"]
>>> foo()
["xyz", "xyz"]
>>> foo()
["xyz", "xyz", "xyz"]
```

46. Python utiliza delimitadores visuais de escopo de métodos que são obrigatórias.

a Verdadeiro

b Falso

Resposta: Verdadeiro.
Justificativa: Em Python, tanto espaços vazios quanto tabulações são utilizados como delimitador. Essa identação deve seguir obrigatoriamente um padrão. O que não acontece em outras linguagens como C ou Java, por exemplo.

47. Selecione a opção que contém somente palavras reservadas no Python:

a eval, and, as e func.

b try, raise, catch, else.

c class, except, as e pass.

d def, class, end e print.

Resposta: c.
Justificativa: func, catch, e and não são palavras reservadas dentro do Python.

Considere o trecho de código abaixo:

```
>>> odd = lambda x : bool(x % 2)
>>> numbers = [n for n in range(10)]
>>> for i in range(len(numbers)):
...     if odd(numbers[i]):
...         del numbers[i]
...
```

Julgue as questões a seguir.

48. O código acima possui um erro, então não será executado.

    Resposta: Verdadeiro
    Justificativa: Durante um loop em um array ou matriz, é um problema. Ao tentar executar esse trecho de código, ocorrerá um erro de IndexError: list index out of range

Considere o trecho de código abaixo:

```
>>> def create_multipliers():
...     return [lambda x : i * x for i in range(5)]
>>> for multiplier in create_multipliers():
...     print multiplier(2)
...
```

49. O resultado será:

    ```
    0
    2
    4
    6
    8
    ```

    a Verdadeiro

    b Falso

    Resposta: Falso.
    Justificativa: O resultado será:

    ```
    8
    8
    ```

```
8
8
8
```

Isso acontece devido ao comportamento de ligação tardia do Python, que diz que os valores das variáveis usadas em fechamentos são vistos no momento em que a função interna é chamada. Portanto, no código acima, sempre que alguma das funções retornadas é chamada, o valor de i é procurado no escopo no momento em que é chamado (e até então, o loop foi concluído, então i já foi atribuído o valor de 4).

A.2.4 Java

Considere o trecho de código abaixo:

```
void m(int n) {
   int k = 1, i = 2, j;

   if (i == n)
      k = j;
   else
      j = k;
}
```

50. É possível afirmar que há um erro de implementação?

    a Verdadeiro

    b Falso

    Resposta: Verdadeiro.
    Justificativa: Uma variável declarada na classe tem um valor inicial, porém uma variável local declarada dentro de um método não tem. É necessário iniciar a variável "j" com algum valor.

51. O Java tem um mecanismo que auxilia na serialização de objetos. É possível afirmar que serializar dados confidenciais em Java é seguro, pois será a JVM responsável pela manipulação dos dados?

    a Verdadeiro

b Falso

Resposta: Falso.
Justificativa: A serialização de dados está fora do controle da JVM. Se não houver nenhuma solução alternativa, os dados devem ser criptografados. Um invasor malicioso pode tentar imitar uma classe legítima pe usar essa técnica para instanciar o estado de um objeto.

52. O código irá compilar mesmo que não possua as“{ }” delimitando a função `if`.

   a Verdadeiro

   b Falso

   Resposta: Verdadeiro.
   Justificativa: Porém para facilitar o entendimento e a manutenabilidade do código a comunidade recomenda o seu uso.

53. É correto afirmar que não há riscos em utilizar métodos marcados como `@Deprecated`?

   a Verdadeiro

   b Falso

   Resposta: Falso.
   Justificativa: Métodos marcados com `@Deprecated` *apenas existem por questões de retrocompatibilidade, já existem novos métodos que levam ao mesmo resultado de forma mais elegante e devem ser usados.*

Considere o trecho de código a seguir:

```
StringBuilder query = new StringBuilder();
query.append(
   "select * from user u where u.name in (" + namesString + ")" );
try {
  Connection connection = getConnection();
  Statement statement = connection.createStatement();
  resultSet = statement.executeQuery(query.toString());
}
```

54. O código irá apresentar algum erro de compilação ou execução?

   Resposta: Falso.
   Justificativa: O código não apresenta erros de sintaxe ou algo do tipo.

55. É correto afirmar que da forma como o código foi estruturado, não há riscos para a segurança de dados manipulados pelo sistema?

    Resposta: Falso.
    Justificativa: Ao utilizar uma query criada desta maneira, torna possível a execução de códigos arbitrários através de SQLInjection. Uma possível solução para o código é apresentada abaixo:

```
StringBuilder query = new StringBuilder();
query.append( "select * from user u where u.name in (?)" );
try {
   Connection connection = getConnection();
   PreparedStatement statement =
connection.prepareCall(query.toString());
   statement.setString(1, namesString );
   resultSet = statement.execute();
}
```

Considere o trecho de código a seguir:

```
public void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
   String content = request.getParameter("some_param");
   //(..)
   response.getWriter().print(content);
}
```

56. Como o código acima não faz persitência em banco de dados, não existe problema utilizar a função `print`.

    Resposta: Falso.
    Justificativa: Apesar de não realizar ações relacionadas ao banco de dados, vulnerabilidades como essa são exploradas através do que é chamado de Cross Site Scripting (XSS).

57. É correto afirmar que o código acima representa um risco à segurança das informações dos usuários?

    Resposta: Verdadeiro.
    Justificativa: Através de Cross Site Scripting (XSS) é possível roubar informações de

vítimas sem quer elas percebam. Para evitar isso, é necessário remover o `print(content)` *e se for o caso, utilizar outro mecanismo para realizar a operação desejada.*

Considere o trecho de código a seguir:

```
public void setMyArray(String[] myArray) {
   this.myArray = myArray;
}
```

58. A execução deste código não acusará nenhum erro.

    Resposta: Verdadeiro.
    Justificativa: Porém não é uma boa prática alterar informações diretamente no Array. Isso vale para qualquer tipo de coleção em Java. Uma boa prática é fazer isso tanto no `set` *quanto no* `get`*. Uma possível solução é apresentada abaixo:*

```
public void setMyArray(String[] newMyArray) {
  if(newMyArray == null) {
    this.myArray = new String[0];
  } else {
    this.myArray = Arrays.copyOf(newMyArray, newMyArray.length);
  }
}
```

APÊNDICE B – Dispositivos Apple

B.0.1 Visão Geral

A Apple é uma empresa de tecnologia multinacional que produz hardware e software. Segundo (DWIVEDI CHRIS CLARK, 2010), talvez o iPhone seja o dispositivo com maior influência que surgiu no mercado nos últimos anos e alavancou a popularidade dos *smartphones* ao redor do mundo. Utilizando tecnologia de ponta, anualmente em seu evento de lançamento de produtos, a Apple apresenta uma nova versão e uma série de novidades para cada dispositivo. A Tabela 6 apresenta os dispositivos fabricados, qual o sistema operacional utilizado e a sua versão mais recente.

Segundo o próprio fabricante (IPHONE. . . , 2016), a maior vantagem destes dispositivos é que o hardware e o software foram criados juntos para que todos os recursos possam ser aproveitados ao máximo. Existe uma série de funcionalidades que possibilitam a comunicação entre todos os dispositivos Apple. Por exemplo, o AppleWatch conectado ao iPhone (WATCH, 2016), o AirDrop para compartilhamento de dados(APPLE, 2016b), o Continuidade (Handoff) para conectar iPhone, iPad, Mac, Watch e iPod (CONTINUIDADE, 2016), etc. Aliando todos esses recursos disponíveis com desenvolvedores espalhados pelo mundo, a Apple anualmente surpreende a todos no evento WWDC (APPLE, 2016a), onde os novos recursos são apresentados.

B.0.2 iOS: Sistema Operacional dos Dispositivos Móveis da Apple

Sempre buscando melhorar cada vez mais a experiência do usuário, anualmente é disponibilizada uma nova versão do iOS com significativas melhorias no que diz respeito à usabilidade, segurança, desempenho e interface. A Apple não permite que ele seja executado em hardware de terceiros, e nem que o usuário instale versões não autorizadas do iOS, o que é popularmente conhecido como *Jailbreaking* (UNAUTHORIZED. . . , 2016), garantindo assim, total controle tanto sobre o software quanto sobre o hardware.

Atualmente, desenvolver para esta plataforma requer conhecimentos em Swift (SWIFT, 2016), cuja versão mais atual é a 3.0 e com sua simplicaidade substitui a

Tabela 6 – Dispositivos e versões de software.

Dispositivo	Sistema Operacional	Última Versão
iPhone e iPad	iOS	10
Macbook	MacOS	10 (Sierra)
AppleWatch	WatchOS	3
AppleTV	TVOS	10

Figura 7 – Camadas do iOS.

Objective-C (PROGRAMMING. . . , 2016) que até então, era a única linguagem utilizada pelos desenvolvedores para dispositivos Apple.

B.0.2.1 Arquitetura iOS

Segundo (ROCHA; NETO, 2014), a arquitetura iOS é formada por quatro camadas: *CoreOS, CoreServices, Media e Cocoa Touch*, como é mostrado na Figura 7 e cada uma delas fornece vários *frameworks* que auxiliam no desenvolvimento de aplicativos. Fonte (ROCHA; NETO, 2014).

Nas camadas superiores, estão as tecnologias e serviços mais sofisticados, e são elas que os desenvolvedores devem utilizar, pois o acesso à camadas inferiores não é permitido pelas aplicações.

A Camada *Cocoa Touch* fornece interação com os principais *frameworks* para desenvolvimento. Ela define ainda, a infra-estrutura básica para as tecnologias fundamentais do sistema, tais como, multitarefa, serviços de notificação *push*, etc.

A camada *Media* contém as tecnologias de gráfico, áudio e vídeo. As principais tecnologias são: *UIKit, AVFoundation, CoreImage e CoreAnimation.*

A camada *Core Services* apresenta os serviços fundamentais do sistema que todos os aplicativos usam. Mesmo que o desenvolvedor não a invoque diretamente, muitas partes do sistema são construídas baseadas nele. As principais tecnologias são: *grand central dispatch, in-app purchase, SQLite e XML support*

A camada *CoreOS* apresenta as características de baixo nível que foram utilizadas na implementação de outras tecnologias. Em geral, utiliza-se essa camada quando o desenvolvedor deseja lidar explicitamente com segurança ou comunicação com *hardware* externo. As principais tecnologias são: *Accelerate e External Accessory*

Segundo (ABOUT. . . , 2016), existe uma série de interações entre o aplicativo e o sistema que devem ser levadas em consideração durante o desenvolvimento. Algumas são

apresentadas abaixo:

- **Aplicativos devem fornecer suporte à recursos chave:** Existe uma série de informações sobre o aplicativo que o sistema precisa ter acesso garantindo que ambos permaneçam estáveis.

- **Os aplicativos tem um fluxo de execução bem definido:** Durante o ciclo de vida (iniciação e término), o estado do aplicativo pode alternar entre primeiro plano, plano de fundo, pode ser encerrado e reiniciado, dormir temporariamente, ser interrompido por uma ligação, etc. Cada vez que ele transita entre estes estados, o comportamento pode ser alterado e deve ser previsto. Os aplicativos em primeiro plano podem fazer inúmeras coisas, enquanto que os que estão em plano de fundo, quase nada.

- **Comunicação entre aplicativos deve seguir caminhos bem definidos:** Por segurança, os aplicativos no iOS são executados em um *sandbox* e possuem interações limitadas com outros aplicativos. Para fazer com que aplicações troquem mensagens, é necessário seguir as regras que definem como fazer isso de forma segura.

B.0.3 Vulnerabilidades do Sistema iOS

Segundo (BANKS; EDGE, 2015), o iOS é um dos sistemas operacionais mais seguros que existem. Vários fatores podem contribuir para esse elevado nível de segurança, como: os usuários e desenvolvedores não possuem acesso à pontos críticos do sistema; as aplicações utilizam o conceito de *sandbox*; as senhas utilizadas são armazenadas no *iCloud Keychain*, etc. De qualquer forma, a necessidade de criar novos e melhorar cada vez mais os mecanismos já existentes é latente.

Atualmente, a utilização do *smartphone* está além de ser somente para um telefone portátil; dessa forma, tanto os fabricantes quanto os desenvolvedores fornecem uma série de funcionalidades nunca antes imaginadas. Segundo (SANTOS,),com esse abrangente conjunto de vantagens, é um trabalho extremamente difícil garantir a segurança das informações contidas nos dispositivos e prover meios que garantam a sua integridade.

Levando em consideração algumas das vunerabilidades conhecidas, tanto desenvolvedores quanto proprietários dos *smartphones* podem evitar grandes problemas. O mesmo autor diz que os ataques virtuais podem ser divididos em duas categorias: Passivos e Ativos.

- **Passivos:** São ataques onde a informação é analisada ou copiada na tentativa de se identificar padrões de comunicação ou de seu conteúdo.

- **Ativos:** São ataques que interferem de alguma forma no funcionamento do sistema.

Existem ainda os ataques físicos, baseados no furto de informações, dispositivos de armazenamento, hardware, etc. Para mitigar os efeitos desse tipo de ataque, é necessário que exista um mecanismo de criptografia e segurança dos dados. O iOS conta com o *Find my iPhone* (BLOQUEIO. . . , 2016) que, além de localizar o dispositivo, permite que ele seja bloqueado.

Como é um software, a qualquer momento que a Apple reconhecer alguma falha, ela pode corrigir e lançar uma atualização; e isso ocorre com certa frequência. Em uma pesquisa rápida na internet, é possível encontrar vários casos onde a Apple lança atualizações para corrigir potenciais problemas.

B.0.4 Backup de Dados

Em especial no iOS, quando um dispositivo é conectado à um computador com o software *iTunes* instalado, por padrão, é feito um backup dos dados contidos nele automaticamente. Esses dados podem ser salvos ou no computador quanto no *iCloud*, de acordo com as configurações. Existem alguns softwares que disponibilizam algumas informações contidas nesse backup, então é importante evitar conectar o dispositivo em computadores não confiáveis.

Segundo (ICLOUD. . . , 2016), o backup do *iCloud* facilita a configuração ou restauração de um dispositivo iOS; para a isso, são necessárias as informações de um dispositivo antigo.

B.0.5 Recuperação de Dados

Como citado anteriormente, o backup dos dados pode ser salvo tanto no próprio computador quanto no *iCloud*. A partir do backup, os dados podem ser facilmente recuperados pelo *iTunes*, bastando conectar o dispositivo ao computador e utilizar o software para recuperá-los mediante a utilização de e-mail e senha cadastrados.

Segundo a Apple (BLOQUEIO. . . , 2016), em caso de perda ou roubo do dispositivo, o usuário pode a qualquer momento efetuar o bloqueio de ativação, impedindo que outra pessoa utilize o dispositivo. Além de ser uma segurança aos dados, que podem ser apagados a partir do iCloud, a utilização do dispositivo fica comprometida, uma vez que o mesmo seja bloqueado. Caso o usuário já tenha um dispositivo e tenha adquirido um novo, os dados podem ser facilmente transferidos por esse backup. Basta que ambos dispositivos estejam com a mesma versão de iOS.

B.0.6 Características Relevantes das versões do iOS

Em seu site oficial, a Apple não disponibiliza um comparativo entre versões, apenas anuncia o que há de novo em cada lançamento de versão. A Tabela 7 apresenta algumas

Tabela 7 – Características entre as versões do iOS.

Versão	Características	Lançamento
iPhone OS 1, 2 e 3	- Versão móvel do OSX utilizado nos desktops; - Telas Sensíveis a Toques - Loja de Aplicativos AppStore - Funcão copiar e colar - MMS	2008 e 2009
iOS 4	- Função Multitarefa no sistema	2010
iOS 5	- iCloud - Central de Notificações - Edição de fotos - Integração com o *Twitter*	2011
iOS 6	- Não perturbe - Aplicativo Mapas e Acesso Guiado - *Passbook* - *Facetime* utilizando dados móveis	2012
iOS 7	- Mudança na interface gráfica - Acesso rápido à calculadora, lanterna - Airdrop - Siri	2013
iOS 8	- Melhorias no aplicativo Mensagens e Fotos - Lançamento do aplicativo Saúde - Integração do iPhone com o AppleWatch - *TouchID*	2014
iOS 9	- Otimização da bateria - Modo baixo consumo de energia - Atualizações em segundo plano	2015
iOS 10	- Melhorias no aplicativo Mensagens, Mapas, Fotos - Utilização da Siri em aplicativos	2016

características, baseado nas notícias de *release* da própria Apple.

B.0.7 Aplicações

Segundo (ABOUT. . . , 2016), os aplicativos iOS seguem o padrão de projeto MVC e o *delegation*. Ou seja, as aplicações desenvolvidas para esta plataforma, apesar de terem certa flexibilidade na arquitetura do código, deverão seguir esses dois padrões, mesmo em baixo nível; entendê-los é crucial para a criação de uma aplicação.

Para a criação de um aplicativo, deve-se levar em consideração a integração dos *frameworks* disponíveis com o código produzido.

A linguagem de programação utilizada pode ser tanto o Objective-C quanto o Swift.

Figura 8 – Representação do padrão de projeto MVC.

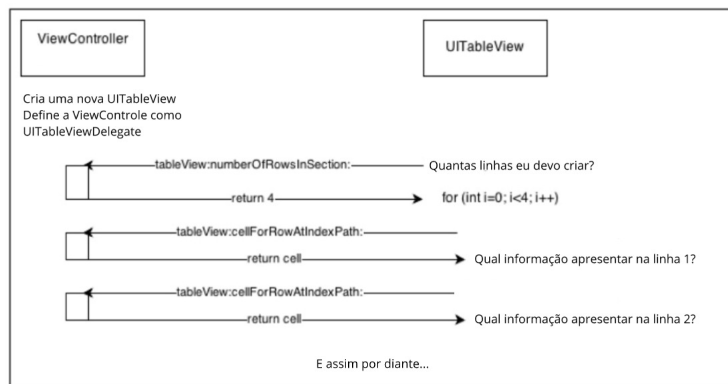

Figura 9 – Pseudo código para implementação de uma *UITableView*.

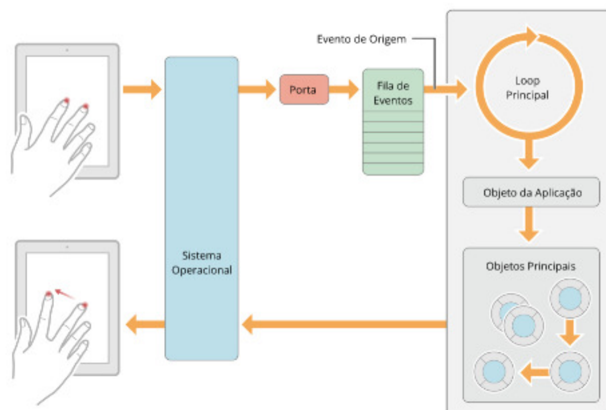

Figura 10 – Eventos na *Main Thread*.

B.0.7.1 Estrutura de uma Aplicação

Como citado anteriormente, as aplicações seguem os padrões de projeto MVC e *Delegate*. A própria Apple recomenda a utilização de outros padrões de projeto, tais como: *Singleton, Decorator, Abstract Factory, Adapter, Façade*, etc. Como acontece em qualquer outra linguagem, o programador deve entender e saber como usá-los para aproveitar ao máximo as vantagens que os Padrões de Projeto podem proporcionar.

- **MVC:** É um padrão estrutural, que classifica os objetos de acordo com o seu propósito geral, facilitando assim, um código limpo e claro. É separado em 3 camadas: *Model, View e Controller*. A camada *Model* é responsável pela leitura, escrita e validação de dados. A camada *View* é responsável pela interação do usuário. E, a camada *Controller*, processa as entradas da *View*, e faz a ligação entre a *View* e a *Model*. A Figura 8 apresenta um esquema visual de como o padrão funciona. (IOS. . . , 2016a)

- ***Delegate***: É um padrão estrutural, faz parte do *Decorator*. É um mecanismo no qual um objeto age em nome de, ou em coordenação com, outro objeto. Ele facilita a criar elementos comuns de aplicações que sem ele, seria extremamente trabalhoso. A Figura 9 apresenta um pseudo código para a criação de uma *UITableView*. Fonte: (IOS. . . , 2016a).

B.0.7.2 Como a aplicação se comporta com o SO

A primeira linguagem utilizada para a criação de aplicativos iOS foi o Objective-C. É uma linguagem baseada em C puro, e executa as operações na *main thread*. Conforme o usuário vai interagindo com a aplicação, os eventos são enviados por uma porta especial configurada pelo UIKit e colocadas em uma fila. Assim que a *main thread* estiver disponível, ela receberá os eventos um por vez. A Figura 10 mostra a arquitetura do

loop principal de execução e como os eventos resultam em ações tomadas pelo aplicativo. Fonte: (ABOUT..., 2016). Outros eventos podem ter comportamento ligeiramente diferentes, principalmente os que tratam do estado da aplicação no SO.

Como se trata de um dispositivo móvel, ao se receber uma ligação ou mesmo ao se operar com bateria em nível criticamente baixo, existem mecanismos que salvam o estado da aplicação, para que quando a mesma voltar para o primeiro plano, os dados criados não sejam perdidos.

B.0.7.3 Como a aplicação se comporta com outras aplicações

Historicamente, a Apple tenta restringir que as aplicações conversem entre si. Principalmente a partir do iOS 4, a utilização do conceito de *sandbox* implica que as aplicações sejam isoladas e todos os seus dados sejam acessíveis apenas por elas mesmas. Mas isso recentemente mudou. Segundo (INTER-APP..., 2016), as aplicações podem conversar entre si indiretamente, desde que estejam instaladas no mesmo dispositivo, através do AirDrop ou utilizando URLs. Porém, dependendo do tipo de dado e finalidade, ele pode ser compartilhado através do framework *UIDocumentInteractionController*.

Caso o dispositivo receba algum arquivo via *AirDrop* (fotos, documentos, URLs, etc.), é possível abri-lo na aplicação através do *delegate* que implementa o seguinte método: `application:openURL:sourceApplication:annotation:`.

Para trocar informações através do uso de URLs, é necessário: utilizar um protocolo específico, criar a URL no formato apropriado e pedir ao sistema para abri-lo. E, para implementar o suporte para troca de informações, é necessário criar um esquema e definir quais tipos de arquivos serão aceitos. Como fazer isso varia bastante de acordo com o formato. Maiores detalhes podem ser encontrados em (DOCUMENT..., 2016a)

B.0.8 Desenvolvimento para Dispositivos iOS

Atualmente, os dispositivos móveis que a Apple fabrica são: iPhone, iPad, iPod Touch, Apple Watch e Apple TV. Em geral, o desenvolvimento para essas plataformas difere no que diz respeito aos *frameworks* disponíveis. Por exemplo, para desenvolver aplicativos para o Apple Watch, é necessário entender o WatchOS, para a AppleTV o TVOS e para, os outros dispositivos, o iOS; deve-se entende como cada dispositivo funciona e quais são os recursos disponíveis em cada plataforma. Por exemplo, eventos de toque não fazem sentido no TVOS, enquanto nas outras plataformas é algo essencial.

Como citado anteriormente, no desenvolvimento de aplicativos para dispositivos móveis da Apple é utilizado Objective-C (PROGRAMMING..., 2016) ou Swift (SWIFT, 2016). O ambiente de programação é o Xcode (DOCUMENT..., 2016b) disponível para a plataforma MACOS, que só pode ser instalada em dispositivos Apple.

Existe ainda a possibilidade de desenvolver aplicações *Cross-Dev*, ou seja, multi plataforma, geralmente baseadas em *HTML* e *JavaScript*. Nesse caso, uma das tecnologias mais famosas é o Ionic (IONIC, 2016).

Após criado o aplicativo, é necessário submeter para avaliação prévia antes de a Apple disponibilizar o aplicativo na loja. Para isso, é necessário utilizar uma conta de desenvolvedor. Atualmente são vendidos dois pacotes: Individual e Empresarial (APPLE. . . , 2016), com valores partindo de US$ 99/ano. Ambos pacotes incluem o desenvolvimento para todas as plataformas Apple.

B.0.9 Recursos e Limitações para Desenvolvedores iOS

Como se trata do desenvolvimento para um fabricante específico que preza muito pela qualidade, alguns procedimentos e métodos devem ser adotados para que o aplicativo seja aprovado e disponibilizado na loja. A princípio, é necessário estar de acordo com o *iOS Human Interface Guidelines* (IOS. . . , 2016b), que é um documento que diz exatamente o que pode ou não ter e o que o aplicativo pode fazer. Existe ainda o suporte da Apple, em vários idiomas, que qualquer dúvida ou problema pode ser resolvido pelo site ou chamada de voz.

Os desenvolvedores Apple tem acesso à uma infinidade de tutoriais e códigos de exemplo para rotinas comuns e todos os vídeos de palestras e apresentações durante o evento anual *WWDC*. Softwares e testes beta, capacidades avançadas, e um software que apresenta métricas dos aplicativos publicados também estão disponíveis. (PROGRAM. . . , 2016)