UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

EMANUELE NUNES DE LIMA FIGUEIREDO JORGE

SENSORWATERMARK: UM ARCABOUÇO DE SOFTWARE PARA APLICAÇÃO DE MARCA D'ÁGUA UTILIZANDO TÉCNICAS DE OFUSCAÇÃO DE CÓDIGO E DE INCORRUPTIBILIDADE PARA REDE DE SENSORES SEM FIO

RIO DE JANEIRO
2015

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Universidade Federal do Rio de Janeiro

Emanuele Nunes de Lima Figueiredo Jorge

SENSORWATERMARK: um arcabouço de software para aplicação de marca d'água utilizando técnicas de ofuscação de código e de incorruptibilidade para rede de sensores sem fio

DISSERTAÇÃO DE MESTRADO

Instituto de Matemática

Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais

Emanuele Nunes de Lima Figueiredo Jorge

SENSORWATERMARK: um arcabouço de software para aplicação de marca d'água utilizando técnicas de ofuscação de código e de incorruptibilidade para rede de sensores sem fio

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, do Instituto de Matemática, e Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais da Universidade Federal do Rio de Janeiro, como requisito parcial à obtenção do título de Mestre em Informática

Orientadora: Luci Pirmez, D.Sc

Rio de Janeiro

2015

CIP - Catalogação na Publicação

NJ82s

Nunes de Lima Figueiredo Jorge, Emanuele
SensorWatermark: Um arcabouço de software para aplicação de marca d’água utilizando técnicas de ofuscação de código e de incorruptibilidade para Rede de Sensores Sem Fio / Emanuele Nunes de Lima Figueiredo Jorge. -- Rio de Janeiro, 2015.
91 f.

Orientador: Luci Pirmez.
Dissertação (mestrado) - Universidade Federal do Rio de Janeiro, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Programa de Pós-Graduação em informática, 2015.

1. Marca d'água de Software. 2. Proteção de Propriedade Intelectual . 3. Proteção de Software. 4. Ofuscação de Código. 5. Incorruptibilidade. I. Pirmez, Luci , orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a).

Emanuele Nunes de Lima Figueiredo Jorge

SENSORWATERMARK: um arcabouço de software para aplicação de marca d'água utilizando técnicas de ofuscação de código e de incorruptibilidade para rede de sensores sem fio

Dissertação de Mestrado submetida ao Corpo Docente do Programa de Pós-Graduação em Informática da Universidade Federal do Rio de Janeiro e à banca externa convidada como parte dos requisitos necessários para obtenção do título de Mestre em Informática.

Aprovada em: Rio de Janeiro, ___ de _____ de _____.

_____

Profa. Luci Pirmez (Orientadora)

D.Sc., COPPE/UFRJ, Brasil

_____

Prof. José Ferreira de Rezende

Dr., Université Pierre et Marie-Curie, França

_____

Prof. Luiz Fernando Rust da Costa Carmo

Dr., Université Toulouse III Paul Sabatier, França.

_____

Prof. Davidson Rodrigo Boccardo

D.Sc., UNESP, Brasil

Rio de Janeiro

2015

Dedico este trabalho a todos da minha família, especialmente a memória póstuma do meu pai Anilton – meu grande herói -, à minha mãe Rozangela, meu filho Gabriel e meu marido Ricardo que foram e sempre serão meu porto seguro.

AGRADECIMENTOS

Primeiramente a Deus por ter me dado força e perseverança para superar este desafio.

Aos meus pais, Anilton e Rozangela que me mostraram desde cedo que conhecimento e educação eram os bens mais preciosos que alguém poderia ganhar. Agradeço pelos ensinamentos, exemplos e dedicação durante toda a minha vida. Para vocês, o meu amor eterno.

Ao meu filho Gabriel, anjo abençoado que ilumina a minha vida e alegra cada segundo do meu dia devido à sua existência.

Ao meu marido Ricardo, amor da minha vida, pelo incentivo, apoio e pela compreensão durante todos os momentos que estive ausente devido a necessidade da dedicação acadêmica.

A toda a minha família. Não citarei nomes, para não esquecer de ninguém. Agradeço a vocês que souberam tolerar e compreender todos os momentos de ausência.

Aos meus amigos e amigas que sempre compreenderam a importância do mestrado para mim. Em especial para minha comadre Ana Paula Fagundes, por estar do meu lado em todos os momentos da minha vida. A distância não nos separa. Ao Professor Welsing Pereira pela amizade sincera e pela ajuda incansável na coordenação do curso. A Lyssandra Garcia e a Vanessa Ribeiro por todo o apoio e força.

Aos colegas do Laboratório de Redes Sem Fio e Multimídia (Labnet) do PPGI/UFRJ. Em especial, ao Thomaz Ávila, ao Gabriel Caldas, ao Vitor Pires, ao Ítalo Cruz, ao Paulo Soares, ao Álvaro Robles e ao Joffre Gavinho. Agradeço a todos pela amizade nesses últimos anos e pelas palavras de coragem e incentivo.

Aos irmãos que Deus colocou em minha vida e escolhi para conviver: Claudio Miceli de Farias que me ensinou que o importante não é vencer todos os dias, mas lutar sempre. Muito obrigada por toda a ajuda, ensinamento e principalmente pela sua amizade. Sei que consegui chegar até aqui porque você lutou junto comigo. Igor Leão dos Santos por me escutar sempre com sua serenidade e paciência. Rafael de Oliveira Costa que quando eu mais precisei, me ajudou a seguir em frente. Tiago Cruz de França que me fez enxergar que eu sou tão capaz quanto qualquer outro. Elton Costa que me apresentou a solução para os problemas.

À Prof.ª Dr.ª Luci Pirmez, minha orientadora e exemplo profissional. Não tenho palavras para agradecê-la pela oportunidade, pelos ensinamentos, por tudo que fez por mim. A senhora foi uma mãe. Muito Obrigada!!!

Também aos demais professores do PPGI, em especial a: Flávia C. Delicato, e Luiz F. Rust da Costa Carmo, pelas sugestões e análises significativas sobre o presente trabalho.

Ao Prof. Davidson Boccardo pelo tempo dedicado fazendo sugestões para o bom desenvolvimento do trabalho.

Ao IFRJ – Instituto Federal do Rio de Janeiro – campus Duque de Caxias por permitir o meu crescimento profissional.

Só tenho a agradecer por todos que contribuíram com o desenvolvimento do meu aprendizado. Vocês me ensinaram muito e serei eternamente grata a vocês.

Com vocês, queridos, divido a alegria desta experiência.

“Você não sabe o quanto eu caminhei, pra chegar até aqui.
Percorri milhas e milhas antes de dormir, eu nem cochilei.”
[Cidade Negra - Toni Garrido, Lazão, Bino Farias, Ras Bernardo, Da Gama, Alessandre Massau]

RESUMO

JORGE, Emanuele Nunes de Lima Figueiredo. **SensorWatermark**: um arcabouço de software para aplicação de marca d'água utilizando técnicas de ofuscação de código e de incorruptibilidade para rede de sensores sem fio. 2015. f. Dissertação (Mestrado em Informática) - Programa de Pós-Graduação em Informática, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2015.

Um dos principais desafios no campo das Redes de Sensores sem Fio (RSSF) relaciona-se à segurança dos seus nós. Isso ocorre porque tais nós, denominados sensores, são dispostos em áreas desprotegidas onde estão vulneráveis a serem capturados por pessoas mal intencionadas (atacantes). Uma vez que esses sensores são capturados, um atacante pode aclamar direitos sobre o seu código e, inclusive, utilizá-lo em outros projetos. Nesse contexto, propõe-se uma marca d'água de software que é gerada por meio da utilização de técnicas de ofuscação de código e de incorruptiblidade a fim de dificultar ataques Man-at-the-End para proteger a autoria do software embarcado nesses sensores e, com isso, desestimular o roubo de propriedade intelectual. A marca d'água proposta é definida como sendo uma sequência ordenada de ofuscações de código aplicadas em um software, onde tal sequência é responsável por identificar quem é o autor desse software, caso a marca d'água seja alterada a utilização da técnica de incorruptiblidade possibilita a restauração da mesma. Tais ofuscações, são capazes de tornar esse código mais difícil de ser analisado, além de dificultar a localização da marca d'água, já que ela confunde-se com as demais instruções do software. Realizaram-se experimentos que possibilitaram medir a verificabilidade da marca d'água proposta, a furtividade, a sobrecarga imposta ao software após inserir a marca d'água e a resistência a ataques.

Palavras-chave: Rede de Sensores Sem Fio. Proteção de Propriedade Intelectual. Marca d'água de Software. Proteção de Software. Ofuscação. Incorruptibilidade.

ABSTRACT

JORGE, Emanuele Nunes de Lima Figueiredo. **SensorWatermark**: um arcabouço de software para aplicação de marca d'água utilizando técnicas de ofuscação de código e de incorruptibilidade para rede de sensores sem fio. 2015. f. Dissertação (Mestrado em Informática) - Programa de Pós-Graduação em Informática, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2015.

One of the main challenges in the field of Wireless Sensor Networks (WSN) is related to the security of their nodes. This is because such nodes, called sensors, are arranged in unprotected areas where they are vulnerable to capture, reverse engineer and tamper with by malicious people (attackers). Once such sensors are captured, an attacker can distribute the software code or even use the code in his projects without being traceable. In this context we propose a "code obfuscation-based software watermarking framework" for resistant against attacks Man-At-The-End and for protecting the authorship of the software embedded in the sensors, and thereby discouraging the theft of intellectual property. The proposed software watermarking framework is defined as an ordered list of obfuscations code applied in a software sequence, where this sequence is responsible for identifying who the author of this software. Such obfuscations are able to make this more difficult to analyze code, and hinder the location of the watermark, since it overlaps with other software instructions. Experiments were performed that allowed to measure the credibility of the watermark proposal, the stealth and the overhead imposed by software after inserting the watermark.

Keywords: Wireless Sensor Networks. Intellectual Property Protection. Software Watermarking. Software Protection. Obfuscation. Tamper-Proofing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ASCII	*American Standard Code for Information Interchange*
BSA	*Business Software Alliance*
INPI	Instituto Nacional de Propriedade Industrial
MATE	*Man-At-The-End*
NESC	Network Embedded Systems C
PC	*Personal Computer*
PI	Propriedade Intelectual
RSSF	Redes de Sensores sem Fio

SUMÁRIO

1 INTRODUÇÃO

A pirataria e a falsificação de software (OSTERGARD, 2000) são problemas de grandes proporções e de escala global que violam os direitos de propriedade intelectual (PI) dos desenvolvedores e fornecedores de software e, assim, ameaçam sua viabilidade de mercado (OSTERGARD, 2000). A pirataria de software fornece aos atacantes a oportunidade não autorizada de utilizar, replicar e distribuir comercialmente o software disponível ou protegido por direitos autorais (OSTERGARD, 2000). Em outras palavras, a pirataria de software é a tentativa de fazer cópias ilegais de uma parte ou de todo software original e vender essas cópias ilegais para os consumidores. A falsificação de software fornece aos atacantes a oportunidade de alterar e aumentar o código do software original de maneira indesejada, ou seja, com a inclusão de lógica maliciosa, *backdoors* e vulnerabilidades exploráveis (OSTERGARD, 2000). O resultado da falsificação de software é o empacotamento de conteúdos pirateados tal que o resultado do produto de software ilegal parece tão genuíno quanto o original (OSTERGARD, 2000). No entanto, independentemente se a ameaça em questão é de pirataria de software ou de falsificação, a intenção de ambos é semelhante: copiar o software sem pedir formalmente pela permissão do autor, provocando assim um aumento da distribuição de software ilegais.

Com o rápido crescimento da indústria de software nas últimas décadas observou-se um aumento na distribuição de softwares ilegais e consequentemente a perda de receita dessas empresas. Nesse novo cenário, onde a informação digital pode ser copiada e transmitida com grande facilidade, a importância da existência de métodos para salvaguardar o investimento da indústria de software faz-se necessária para dificultar a pirataria e a falsificação de software.

No ano de 2011, a *Bussiness Software Alliance* (BSA) publicou um estudo (SHADOW MARKET, 2011) que mostra que a taxa de pirataria global de software para computadores pessoais estava em torno de 42%. A taxa de pirataria de software de um país é definida como sendo a quantidade de programas sem licença dividida pelo total instalado em todos os computadores de cada país. O valor monetário dos softwares pirateados (“prejuízos” dos fornecedores de software) subiu de 58,8 bilhões de dólares em 2010 para 63,4 bilhões de dólares em 2011 (SHADOW MARKET, 2011). Tais fatos apresentados revelam a necessidade de investigação e desenvolvimento de soluções eficazes para apoiar a luta contra a pirataria de software. De acordo com essa percepção, observou-se um aumento do interesse dos desenvolvedores de software no desenvolvimento de novas técnicas de proteção de

propriedade intelectual, como forma de garantir o uso legal do software. As metodologias recentes geralmente cobrem uma variedade de técnicas, como a proteção legal por direitos autorais, patentes, licenças e proteção tecnológica (EBERHARDT *et al.,* 2007).

Do ponto de vista da proteção tecnológica, algumas técnicas de proteção de propriedade intelectual de software têm sido desenvolvidas nos últimos anos para dificultar a pirataria de software (YANG *et al*. 2009). Os objetivos da utilização de tais técnicas são: (i) facilitar a verificação da autenticidade do software e (ii) dificultar a subversão dos processos de distribuição de software. (OSTERGARD, 2000).

Apesar do problema da pirataria de software ser difundido em sistemas convencionais, existe uma preocupação crescente para desenvolvedores de sistemas embarcados sobre a proteção da propriedade intelectual de seus softwares, principalmente quando o sistema está localizado em ambientes inseguros e hostis, como é frequentemente o caso das redes de sensores sem fio (RSSF).

Uma RSSF é composta por dezenas a milhares de dispositivos de baixo custo, alimentados por baterias e de tamanho reduzido, capazes de realizar sensoriamento, processamento e transmissão de informação através de enlaces sem fio. Sensores são alimentados por baterias não recarregáveis e devem operar sem assistência humana por longos períodos de tempo. Tais redes podem ser utilizadas em diversos tipos de aplicações como, por exemplo: monitoramento de alvos militares, detecção de incidentes ambientais e muitas outras (LEVIS e GAY, 2009).

Se por um lado as RSSFs trazem novas e amplas perspectivas para várias aplicações, por outro trazem uma série de desafios. Os principais desafios estão relacionados à limitação de recursos dos sensores e as vulnerabilidades associadas à comunicação sem fio e à organização *ad-hoc* (características inerentes das RSSFs). Outro desafio está relacionado ao fato de que as RSSFs geralmente estão dispostas em áreas abertas, desprotegidas e às vezes até hostis tornando a vigilância dos nós dessa rede uma tarefa praticamente impossível. Com isso, os nós da RSSFs são vulneráveis a ataques (*Man-At-The-End* - MATE) (BOCCARDO *et al*., 2010).

Esse tipo de ataque pode ocorrer em cenários não supervisionados, onde a presença humana é limitada, na qual um atacante é capaz de capturar um nó dessa rede e de efetuar a leitura da sua memória. Nesse tipo de ataque, um atacante deseja capturar um ou mais desses sensores, com o intuito de analisar tanto o software quanto o hardware desses nós a fim de obter alguma vantagem. Por exemplo, um atacante de posse de um ou mais nós sensores pode aclamar-se como o autor do software embarcado nesses sensores ou extrair seus módulos

proprietários e utilizá-los em outros projetos, infringindo o direito de propriedade intelectual deste software. Daí a necessidade de prover meios para garantir ou rastrear a propriedade intelectual do software embarcado. O desafio passa a ser maior em RSSF, uma vez que os nós sensores das RSSFs estão dispostos em ambientes hostis.

Uma das estratégias tradicionais de proteção de propriedade intelectual de software é por meio da inserção de uma marca d'água no software de forma codificada. Uma marca d'água de software caracteriza-se como sendo uma informação única que deve ser inserida no código de um software com o intuito de reconhecer quem é o autor desse software, atestando quem é o detentor dos direitos sobre o mesmo (BOCCARDO *et al.* 2010). O trabalho de Collberg e Nagra (COLLBERG e NAGRA, 2010) menciona várias formas de codificação e decodificação de marca d'água. Uma das estratégias mais comuns de marca d'água de software é a inclusão do nome do autor em um ponto específico desse software (COLLBERG e NAGRA, 2010). Tal estratégia é vantajosa porque não aumenta significativamente o tamanho do software e nem o seu tempo de execução. Contudo, tal técnica carece, na maioria das vezes, de outras técnicas para agregar furtividade e resiliência. Sem furtividade, um atacante pode facilmente localizar tal marca d'água, podendo ser capaz de modificá-la ao alterar o nome do autor ou simplesmente removê-la. Sem resiliência, um atacante pode aplicar transformações semanticamente equivalentes no código a fim de destruir a marca d'água.

Alguns trabalhos da literatura propõem estratégias de combinar a marca d'água com outras técnicas de proteção de software, como a ofuscação de código (ZENG *et al*. 2011), a fim de dificultar a localização da marca d'água, ou seja, de torná-la furtiva, desestimulando um atacante a realizar a engenharia reversa, por exigir muito tempo ou muito poder computacional. A ofuscação de código pode ser caracterizada como a substituição e/ou inserção de um conjunto de instruções em um software, sem modificar o comportamento original do software, ou seja, mantendo as funcionalidades originais do software. Assim, o emprego da técnica de ofuscação de código com a estratégia de marca d'água de software possibilita a criação de uma marca d'água furtiva, ou seja, que é mais difícil de ser localizada pelo fato do código ser de menor compreensão (código ofuscado). Outros trabalhos, propõem a combinação de técnicas de incorruptibilidade com o esquema de marca d'água (KHIYAL *et al.,* 2010), a fim de dificultar a destruição da marca d'água. A incorruptibilidade de código consiste na inserção de travas lógicas no código da aplicação visando a verificação da integridade da marca d'água. A aplicação conjunta de tal técnica no esquema de marca d'água possibilita recuperar a marca d'água caso a mesma seja modificada, em uma tentativa de destruição da marca, dificultando um atacante a realizar um ataque de transformação de

código. Diante disso, a adoção de soluções de aplicação de marca d'água que sejam furtivas e resistentes aos ataques de transformação de código, tornam-se necessárias.

1.1 OBJETIVO

O presente trabalho propõe um arcabouço, denominado SensorWatermark, considerado furtivo, resistente à violação e eficiente, uma vez que pode ser aplicado para RSSF. O SensorWatermark tem como objetivo principal um esquema de aplicação de marca d'água que utiliza técnicas de ofuscação de código e de incorruptibilidade, a fim de dificultar a localização e a modificação da marca d'água de software embarcado nos nós sensores de uma Rede de sensores sem fio (RSSF). O esquema de marca d'água proposto nesse trabalho é definido como sendo uma sequência ordenada de ofuscações de código aplicada em um software, onde tal sequência é responsável por identificar quem é o autor desse software. Dessa forma, o SensorWatermark é dito ser ***verificável***, porque é capaz de detectar corretamente todas as marcas d'água inseridas em softwares distintos. O SensorWatermark é dito ser ***eficiente,*** porque o esquema de marca d'água proposto nesse trabalho não impacta de forma negativa no consumo de recursos do dispositivo sensor já que essas ofuscações são baseadas em substituições de instruções já existentes e/ou inserem poucas instruções no software. O SensorWatermark é dito ser ***resistente*** a ataques de transformações de código porque em caso de corrupção da marca d'água, a marca d'água original pode ser restaurada, por meio de mecanismos de resposta. O SensorWatermark é dito ser ***furtivo***, porque é difícil de localizar a marca d'água uma vez que a marca d'água de software está atrelada à ofuscações que tornam o software menos inteligível.

Ao contrário de outras abordagens na literatura, como as de (ZENG et al., 2010),(CHEN e CHAOQUAN, 2012), e (XU *et al*., 2012), esse arcabouço é considerado resistente à ataques, porque depois que o software com marca d'água for submetido a ataques de transformação de código, as técnicas de incorruptibilidade utilizadas verificarão a realização da transformação e recuperarão o trecho do código alterado do software.

O SensorWatermark é considerado eficiente comparada ao trabalho de (KHIYAL *et al*., 2010) pois gera pouca sobrecarga em termos de consumo de recursos, pois as ofuscações empregadas substituem instruções já existentes e/ou inserem poucas instruções no software.

1.2 ORGANIZAÇÃO DO TRABALHO

Esse trabalho está organizado em 7 capítulos. O capítulo 2 apresenta os conceitos básicos sobre os temas abordados na concepção desta proposta, necessários para sua compreensão. São revistos conceitos básicos sobre propriedade intelectual, técnicas de proteção de propriedade intelectual, engenharia reversa, ofuscação de código, incorruptibilidade, RSSFs e segurança em RSSF. O capítulo 3 apresenta um levantamento bibliográfico inspirado em uma revisão sistemática dos trabalhos relacionados na literatura. O capítulo 4 apresenta a proposta do presente trabalho, o arcabouço SensorWatermark, detalhando seus componentes, sua operação e o seu modelo de ataque. O capítulo 5 descreve as questões envolvidas na implementação de um protótipo do SensorWatermark. O capítulo 6 detalha os experimentos realizados para avaliar o arcabouço SensorWatermark em termos da furtividade da marca d'água inserida, da sua credibilidade, do seu impacto sobre o consumo de recursos da RSSF, assim como a sua resistência à ataques. Finalmente, o capítulo 7 apresenta a conclusão e os trabalhos futuros.

2 CONCEITOS BÁSICOS

Nesse capítulo são apresentados os conceitos básicos necessários para a compreensão do trabalho proposto. Na Seção 2.1 e 2.2 o conceito de propriedade intelectual e as técnicas de proteção de propriedade intelectual de software são apresentados. Na Seção 2.3 e 2.4 são apresentados respectivamente os conceitos de engenharia reversa e ofuscação de código. Na seção 2.5 é apresentado a técnica de incorruptibilidade. Na Seção 2.6 é apresentado o conceito de Redes de Sensores Sem Fio. Na Seção 2.7 são levantados os requisitos de segurança em Redes de Sensores sem Fio. Na Seção 2.8 são apresentadas as considerações finais sobre o capítulo.

2.1 PROPRIEDADE INTELECTUAL

A propriedade intelectual é a área do direito que trata de proteger os direitos daqueles que criam obras originais. Esta área cobre tudo, desde peças originais até marcas de identificação de uma empresa. O objetivo das leis de propriedade intelectual é incentivar novas tecnologias, expressões artísticas e invenções ao promover o crescimento econômico. Quando as pessoas sabem que o seu trabalho inovador estará protegido e que eles podem se beneficiar deste trabalho, eles são mais propensos a continuar a produzir obras, desenvolver novas tecnologias e tornar os processos mais eficientes. Existem três principais mecanismos de proteção da propriedade intelectual: direitos autorais, patentes e marcas registradas (RENNER, 2015).

Na Subseção 2.1.1 é apresentado uma visão geral sobre direito autoral. Na subseção 2.1.2 é apresentado o conceito de patente e na Subseção 2.1.3 é apresentado a visão geral sobre marcas registradas.

2.1.1 Direito Autoral

O direito autoral oferece proteção para autoria de obras originais, fixados em um tangível meio de expressão, incluindo literária, musical, e obras de teatro, bem como fotografias, gravações de áudio visuais, software e outras obras intelectuais. A proteção de direitos autorais inicia-se como um método de informar aos outros que ele tem a intenção de exercer o controle sobre a produção, distribuição, exibição e desempenho da sua obra. A proteção de direito autoral dá aos proprietários o direito exclusivo de reproduzir o seu

trabalho, exibir publicamente, executar, e criar trabalhos derivados. Além disso, proprietários de direitos autorais têm o direito econômico de beneficiar-se financeiramente de seu trabalho e proibir os outros de fazê-lo sem a sua permissão (RENNER,2015). No presente trabalho, o mecanismo de proteção de propriedade intelectual utilizado será a técnica de aplicação de marca d'água de software, que possibilita a identificação do autor do software e do detentor do direito autoral. Esta técnica será apresentada na Subseção 2.2.1.

2.1.2 Patente

A patente pode ser concedida a uma invenção única considerada crucial para o sucesso de muitas organizações, instituições ou empresas (RENNER, 2015). As implicações da obtenção de uma patente relacionada a uma invenção original é que o titular de uma patente pode impedir que terceiros possam produzir, usar ou vender sua invenção por um período de anos, dependendo do tipo de invenção (RENNNER, 2015).

2.1.3 Marca Registrada

Uma marca registrada protege um produto com uma identificação correta sobre o produto e sua origem. O propósito inicial de proteção da marca registrada era tornar ilegal qualquer outro produto fazer-se passar pelo produto registrado. Assim, a proteção de marcas evoluiu como uma forma de proteção indireta do consumidor, assegurando que as decisões de compra são com base em informações corretas sobre o produto e até mesma confiança na marca (BESER, 1991).

2.2 TÉCNICAS DE PROTEÇÃO DE PROPRIEDADE INTELECTUAL DE SOFTWARE

As técnicas de proteção de propriedade intelectual de software são usadas para garantir que o software seja usado legalmente, ou seja, dificulta que os direitos de propriedade intelectual dos desenvolvedores e fornecedores de software sejam violados. A seguir, apresentamos uma técnica de proteção de propriedade intelectual, marca d'água de software.

2.2.1 Marca d'água de Software

A marca d'água de software é uma técnica utilizada para inserir uma informação secreta dentro do software (COLLBERG e NAGRA, 2010). Esta informação pode identificar o proprietário do software. Assim, quando uma utilização não autorizada deste software ocorre, detentores dos direitos autorais deste software podem ter evidências de pirataria, extraindo esta mensagem secreta de uma cópia não autorizada (ZHU *et al.* 2011). A marca d'água fornece elementos para identificar a utilização não autorizada de um software, mas não impede o uso indevido.

Segundo Collberg em (COLLBERG e NAGRA, 2010), um sistema de aplicação de marca d'água compreende 3 operações: inserção, extração e detecção.

Inserção (P,w,*chave*) $\rightarrow P_w$
Extração (P_w, *chave*) $\rightarrow w$
Detecção (P_w, *chave*, w) $\rightarrow$[0.0,1.0]

Tabela 1: Funções do esquema de aplicação de marca d'água

A operação de Inserção de marca d'água, apresentada na Figura 1, recebe um software P como entrada e o transforma em um software P_w por meio da inserção da marca d'água w. Como um atacante pode conhecer o algoritmo de inserção que está sendo utilizado para inserir a marca d'água, é necessário a existência de algum segredo que só seja conhecido pelo proprietário do algoritmo, mas não pelo atacante. Esse segredo é a chave utilizada na inserção da marca d'água. A chave também é utilizada pela operação de Extração para recuperar a marca d'água do software P_w. O software com a marca d'água inserida deve ter sua semântica preservada, isto é, P e P_w devem ter o mesmo comportamento. A operação de Detecção pega a marca d'água w como entrada e retorna a credibilidade de que o software P_w contém w.

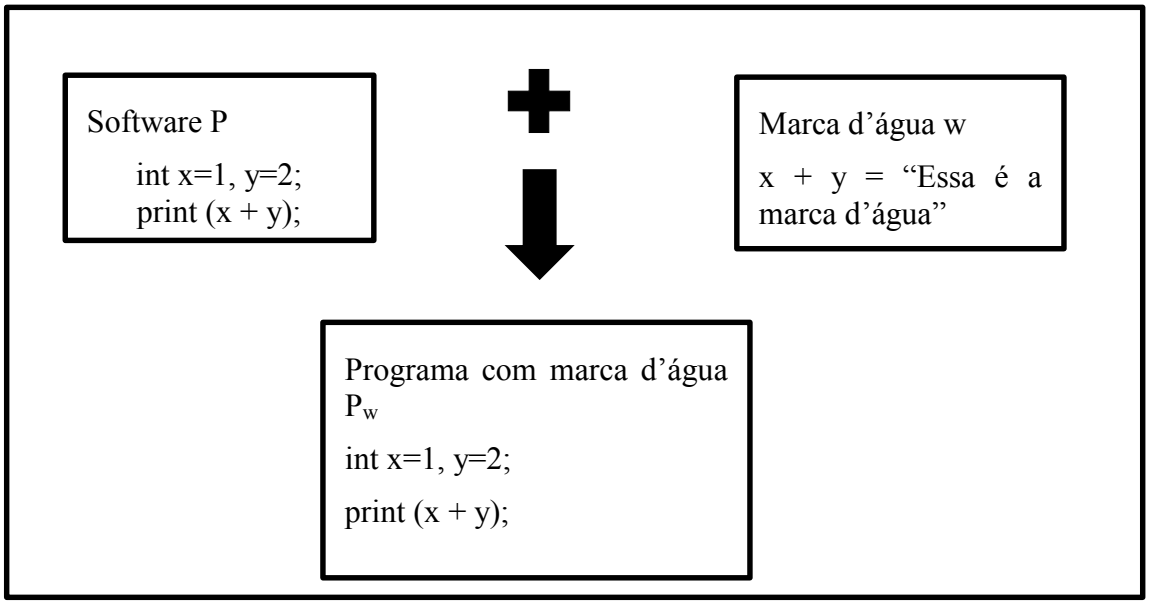

Figura 1: Processo de Inserção da Marca d'água

Segundo Zhu em (ZHU *et al.* 2011), uma marca d'água de software pode ser classificada por diferentes formas dependendo da sua função ou da sua propriedade. De acordo com o tipo, a marca d'água de software pode ser classificado como: marca d'água autoral, marca d'água de impressão digital, marca d'água de validação e marca d'água de licença. A ***marca d'água autoral*** tem por objetivo inserir uma informação secreta no software que identifica a quem pertence o direito autoral do software. A ***marca d'água de impressão digital*** é a informação inserida em todas as cópias do software que forem distribuídas. Sendo que, diferente da marca autoral que identifica o direito autoral do proprietário do software, a impressão digital identifica unicamente o comprador do software. É importante ressaltar que tendo em mãos uma cópia do software com uma marca d'água de impressão digital é possível extrair a marca d'água do software e rastrear o conteúdo até a pessoa que originalmente o comprou. A ***marca d'água de validação*** é usada para verificar se o software marcado é autêntico e não foi alterado ou falsificado, por exemplo como as *Applets* JAVA assinados digitalmente. Uma ***marca d'água de licença*** codifica, invisível e robustamente, a maneira pela qual o software pode ser usado pelo usuário final. Por exemplo, nesse caso uma marca pode ser a data a partir da qual a cópia do software pode ser executada.

Segundo Boccardo em (BOCCARDO *et al*. 2010), os extratores e os detectores da marca d'água podem ser classificados como ***cego*** ou ***informado.*** A marca d'água pode ser extraída por um algoritmo de extração ou verificada por um algoritmo de detecção que

comprove quem é o proprietário do software. Os algoritmos de extração ou detecção da marca d'água de software podem ser classificados como ***cego*** quando o programa original e a marca d'água não estão disponíveis, ou ***informado*** quando o programa original e/ou a marca d'água estão disponíveis. Um exemplo de uso de extratores ou detectores informados são os casos do rastreamento de transações em que o proprietário da obra original busque identificar quem distribuiu ilegalmente a obra. Como o usuário do extrator ou detector é o proprietário da obra, então esta obra estará disponível, à priori, durante o processo de extração. Em outros casos, a obra não estará disponível, como é o caso, por exemplo, da extração da marca d'água de um vídeo para fins de verificação de autenticidade. Neste caso, o vídeo é uma obra desconhecida, de forma que o extrator deverá ser capaz de obter a marca d'água sem o conhecimento da obra original. Isso porque a obra cujo a marca d'água está sendo verificada pode sofrer uma série de alterações antes do processo de extração, tanto alterações maliciosas, visando à própria corrupção da marca d'água, quanto alterações lícitas, tal como a própria inclusão da marca d'água. Assim, a obra original torna-se uma boa referência para o algoritmo de extração ou detecção da marca d'água.

Segundo Zhu (ZHU *et al.* 2011), as técnicas de marca d'água de software pode ser classificada como ***estática*** e ***dinâmica***. Segundo Boccardo (BOCCARDO *et al.*, 2010), na operação estática, a marca é armazenada no próprio código executável e o processo de inserção e extração é feito sem a necessidade de o programa ser executado ou interpretado. Uma marca d'água de software é considerada **estática** quando uma informação é inserida em uma área de dados ou no código. Para a extração da marca d'água o software não precisa ser executado. Uma marca d'água de código é inserida dentro da área de código do programa, isto é, a marca d'água é armazenada na seção do executável que contém as instruções, enquanto uma marca d'água de dados é inserida diretamente dentro da área de dados do programa, isto é, a marca d'água é armazenada em qualquer outra seção, tais como, cabeçalho, seção de variáveis, seção de informações de depuração e etc.

Na **dinâmica**, o programa ou parte dele precisa ser executado ou interpretado para que a marca d'água seja extraída. O processo de extração de uma marca d'água dinâmica utiliza-se de uma sequência especial de entrada para extrair a marca de um determinado estado do programa. Uma marca d'água de software é considerada **dinâmica** quando esta é inserida no estado de execução de um software. Mais precisamente, na marca d'água de software dinâmica, o que tem sido inserido não é a própria marca d'água, mas alguns códigos que possibilitem que a marca d'água possa ser expressada, ou extraída, quando o software com a marca d'água for executado.

Segundo Collberg em (COLLBERG e NAGRA, 2010), dependendo do aspecto em relação à visibilidade do software que está sendo protegido, a marca d'água inserida pode ser considerada ***visível*** ou ***invisível***. Uma marca d'água de software é considerada ***visível*** quando pode ser inserida por alguma entrada especial que fará o software gerar uma imagem legível como um logotipo, etc, para mostrar a existência de tais marcas d'água visíveis no software. Uma marca d'água de software é considerada ***invisível*** é aquele que não aparece como uma imagem legível para o usuário final, mas pode ser extraído por algum algoritmo sem o controle direto do usuário final.

Outra classificação existente de marca d'água é quando esta pode ser considerada ***robusta*** ou ***frágil***. Uma marca d'água de software é considerada robusta quando ela é difícil de ser modificada, isto é, ela continuará ilesa, mesmo que o software seja submetido a transformações semânticas. A marca d'água de software é considerada frágil quando ao alterar o software (o software for submetido a transformações semânticas) esta é sempre destruída.

Outro ponto importante, são as formas de ataque que a marca d'água de software pode ser submetida. Em (COLLBERG e NAGRA, 2010) e (ZHU *et al.* 2011), são definidas quatro principais formas para atacar uma marca d'água de software, são elas: ataque aditivo, ataque subtrativo e ataque distorcivo. Para este trabalho, somente será detalhado o ataque distorcivo.

O ***ataque aditivo*** incorpora uma nova marca d'água em um software com marca d'água, como ambas as marcas d'água estarão, agora, presentes no software, isto confundirá quando o proprietário do software tentar provar a propriedade de sua marca d'água original.

Com o ***ataque subtrativo***, atacantes removem a marca d'água do software marcado, sem afetar a funcionalidade do software com a marca de água. Para ser mais preciso, depois da localização da marca d'água e a remoção, o software atacado ainda deverá estar intacto o suficiente para ter algum valor.

O ***ataque distorcivo*** tem como objetivo modificar a marca d'água para evitar que ela seja extraída pelos proprietários dos direitos autorais do software e ainda mantenha a usabilidade do software. A desvantagem deste ataque é que muitas transformações distorcivas degradam o software, seja tornando-o maior, seja tornando-o mais lento.

Em (COLLBERG e THOMBORSON, 1999) e (ZHU *et al.,* 2011), são usados quatro critérios de avaliação para mensurar a qualidade de uma marca d'água de software. O critério de ***taxa de dados*** expressa a quantidade de dados que pode ser incorporado no interior do código do programa ao inserir a marca d'água. O critério ***resiliência*** é a capacidade de resistir contra transformações semânticas, enquanto o critério ***furtividade*** diz respeito de quanto a

marca d'água pode ser difícil de ser localizada comparando o software original com o software com a marca d'água. Finalmente, o critério de ***desempenho*** é a razão entre o tempo de execução do software com marca de água com o tempo de execução do programa original.

2.3 ENGENHARIA REVERSA

Essa seção apresenta a transcrição textual sintetizada de parte da dissertação de mestrado de (COSTA, 2012) páginas 19 e 20, maiores informações sobre esse conceito, remeter-se a dissertação de mestrado de (COSTA, 2012).

Segundo Costa em (COSTA, 2012), geralmente os programas são distribuídos através do seu código binário. Entretanto, como o código fonte não é disponibilizado, é difícil entender o comportamento interno de um programa já que existem situações que requerem esse entendimento a fim de reparar algum problema no programa ou adicionar novas funcionalidades. Nesses casos, é utilizada a engenharia reversa. A engenharia reversa de um programa é um processo de verificação e compreensão do software existente que tenta recriar o projeto e decifrar os requisitos atualmente implementados, apresentando-os em um nível ou grau mais alto de abstração a fim de facilitar a análise de um programa. Apesar dos benefícios que a engenharia reversa pode possibilitar, existem situações em que ela é utilizada com intuitos maliciosos. Pessoas maliciosas têm utilizado a engenharia reversa para comprometer a propriedade intelectual de um programa através da extração de dados confidenciais ou de algoritmos que poderão ser utilizado em outros projetos. Além disso, a engenharia reversa realizada de forma maliciosa pode permitir a um atacante modificar ou entender o funcionamento de um programa a fim de obter alguma vantagem, por exemplo, explorar uma vulnerabilidade.

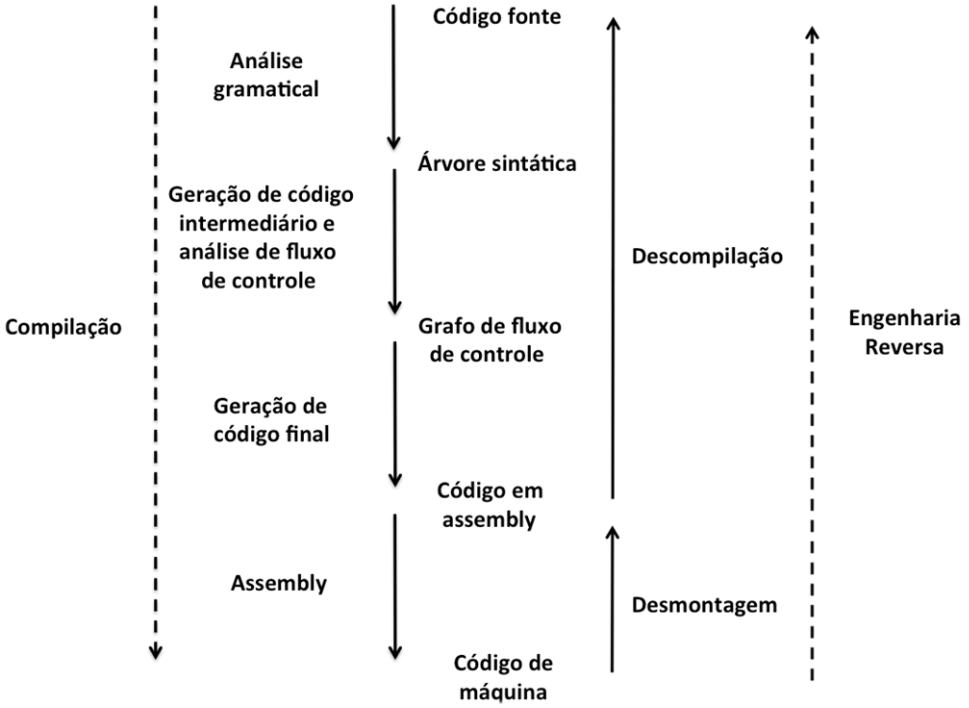

Figura 2: Fases de compilação e de engenharia reversa

A engenharia reversa é o caminho inverso da compilação e é dividida em duas fases: a fase de desmontagem e a fase de descompilação, como mostrado na Figura 2. A fase de desmontagem é a primeira fase da engenharia reversa onde os bytes do código executável de um programa são traduzidos para um código em *assembly*, mnemônico utilizado para representar as instruções que deverão ser executadas por um processador. A fase de descompilação por sua vez utiliza o código gerado pela desmontagem a fim de construir estruturas de alto nível que facilitem a análise de um programa.

A descompilação é realizada porque analisar o código *assembly* de um programa inteiro é uma tarefa que requer muito esforço já que um código em *assembly* é sintaticamente pobre em relação à identificação dos limites de funções, seus parâmetros e das instruções que invocam chamadas e retornos de função. A descompilação é totalmente dependente da desmontagem, pois é a partir do código em *assembly* que são realizadas análises de programa com o intuito de gerar estruturas de mais alto nível, tais como grafos de fluxo de controle e grafo de chamadas de função. Em suma, a descompilação é realizada com o intuito de criar um código fonte legível a fim de facilitar o entendimento de um programa.

2.4 OFUSCAÇÃO DE CÓDIGO

Essa seção apresenta a transcrição textual sintetizada de parte da dissertação de mestrado de (COSTA, 2012) sobre ofuscação de código nas páginas 27, 28, 29, 30, 32, 36, 37, e 38, maiores informações sobre esse conceito, remeter-se a dissertação de mestrado de (COSTA, 2012).

Segundo Costa em (COSTA, 2012), a ofuscação de código é uma técnica que visa produzir um programa semanticamente equivalente ao programa original, mas com alterações em sua sintaxe capazes de dificultar o entendimento de um programa. A ofuscação de código é utilizada para dificultar a engenharia reversa de um programa e com isso dificultar a extração de dados ou algoritmos proprietários. Uma das aplicações de ofuscação de código é para diversificar um programa. Esta diversidade consiste na criação de versões sintaticamente diferentes de um programa, porém semanticamente equivalentes. Esta diversidade é utilizada para dificultar ataques de colisão, no qual um atacante com posse de mais de uma cópia do programa, consegue descobrir a localização de uma determinada propriedade do programa, por exemplo, *fingerprint*, marca d'água, serial ou até mesmo a aplicação de uma ofuscação por substituição de instruções através da simples diferença das cópias do programa. No entanto, a diversidade dificulta a localização dessas propriedades, pois várias partes do programa serão identificadas como diferentes, não revelando assim, a propriedade desejada (BOCCARDO *et al*., 2010).

A ofuscação de código pode ser aplicada tanto no comprometimento da fase de desmontagem como na fase de descompilação da engenharia reversa. A maioria das técnicas de ofuscação atua na fase de descompilação, tornando a análise de um programa mais complexa. Tais técnicas podem, por exemplo, alterar o fluxo de controle de um programa, modificar a abstração de funções através de junção[1] ou separação[2] dessas ou alterar expressões simples por expressões complexas (BALAKRISHNAN e SHULZE, 2005). As técnicas para alterar o fluxo de controle de um programa modificam um programa com o intuito de reordenar os blocos básicos e arestas do grafo de fluxo de controle, bem como incluir blocos e arestas para dificultar a análise desse grafo. Outras técnicas realizam transformações no programa a fim de remover a abstração de funções de um programa. Pessoas analisam um programa de forma modularizada, analisando um programa por partes, geralmente analisando uma função por vez. A separação e junção de funções comprometem

[1]Original do Inglês, inlining
[2] Original do inglês, outlining

essa análise já que a delimitação entre as funções não estará mais presente. O uso da técnica de equivalência de expressões simplesmente altera como é feita, por exemplo, as atribuições de variáveis, ao invés de simplesmente atribuir uma constante a uma determinada variável, a ofuscação utiliza deslocamento de registradores, dificultando a análise do programa.

As técnicas de desmontagem baseiam-se na fragilidade das convenções de compilação a fim de comprometer a fase de desmontagem das ferramentas de engenharia reversa ao obrigar a tradução incorreta do código executável de um programa. Essas técnicas são realizadas através da substituição ou inclusão de instruções.

As principais técnicas de ofuscação desse tipo são: ofuscação de chamada (LAKHOTIA *et al.*, 2010), ofuscação de retorno (LAKHOTIA *et al.*, 2010), ofuscação de falso retorno (VIEGA *et al*, 2003) e ofuscação de salto incondicional (LINN e DEBRAY, 2003). É bom ressaltar que todas essas técnicas fazem o redirecionamento no fluxo de controle. A seguir são detalhadas as técnicas de ofuscação de chamada, técnica de ofuscação de retorno e técnica de salto incondicional.

2.4.1 Ofuscação de Chamada

De acordo com as convenções usadas pelos compiladores, as ferramentas de engenharia reversa identificam o início e o fim de uma função através de instruções de chamada e de retorno. O destino de uma instrução de chamada delimitaria o início de uma função e a instrução de retorno do fim de uma função. A ofuscação de chamada é uma técnica capaz de remover a abstração de funções de um programa removendo a instrução de chamada e, consequentemente, a delimitação do início de funções. Qualquer análise interprocedural realizada em funções que tiveram suas chamadas ofuscadas será comprometida, pois a mesma depende da delimitação das funções para gerar seus resultados. Caso toda chamada de função do programa fosse ofuscada, ferramentas de engenharia reversa enxergariam o programa todo como uma única função. Esse tipo de ofuscação também dificulta a análise manual de um programa visto que é mais fácil para um atacante analisar várias funções menores do que analisar poucas funções que contemplam toda a execução do programa.

Esse tipo de ofuscação é realizado substituindo as instruções de chamada de função de um programa, instruções *CALL*, por um conjunto de instruções capazes de realizar a mesma operação da instrução *CALL*. A semântica de uma instrução *CALL* é a seguinte: (i) armazena na pilha o endereço de retorno da função e (ii) redireciona o fluxo de controle do programa

para o endereço da função a ser executada. Esse comportamento pode ser realizado, por exemplo em uma arquitetura x86, substituindo uma instrução *CALL* por duas instruções *PUSH* e uma instrução *RET*. A primeira instrução *PUSH* é utilizada para armazenar na pilha o endereço de retorno da função, endereço para onde o fluxo de controle deverá ser redirecionado após o retorno da função. A segunda instrução *PUSH* é utilizada para armazenar no topo da pilha o endereço da função a ser executada. Por último a instrução *RET* é utilizada a fim de redirecionar o fluxo de controle para o endereço que está no topo da pilha, ou seja, o endereço da função a ser executada. Quando a instrução *RET* da função for executada, o fluxo de controle será redirecionado para o endereço armazenado no topo da pilha, endereço armazenado pela primeira instrução PUSH do conjunto de instruções utilizado para substituir a instrução *CALL*.

Main:	Main:
A1: PUSH 0xA	B1: PUSH 0xA
A2: PUSH 0xB	B2: PUSH 0xB
A3: CALL Add	B3: PUSH B6
A4: PUSH 0xC	B4: PUSH B14
A5: PUSH 0xD	B5: RET
A6: CALL Add	B6: PUSH 0xC
A7: JMP A9	B7: PUSH 0xD
A8: NOP	B8: PUSH B11
A9: RET	B9: PUSH B14
Add:	B10: RET
A10: MOV EAX, [ESP+4]	B11: JMP B13
A11: MOV EBX, [ESP+8]	B12: NOP
A12: ADD EAX, EBX	B13: RET
A13: RET	Add:
	B14: MOV EAX, [ESP+4]
	B15: MOV EBX, [ESP+8]
	B16: ADD EAX, EBX
	B17: RET

Figura 3: Exemplo de ofuscação de chamada

A Figura 3 é um exemplo de ofuscação de chamada. A coluna da esquerda mostra o programa original. Na coluna da direita apresenta como as chamadas de função foram

ofuscadas. As instruções localizadas entre B3 e B5 da direita ofuscam a chamada de função apresentada em A3 (esquerda). A instrução “PUSH B6” (B3) empilha o endereço de retorno da função e a instrução “PUSH B13” (B4) empilha o endereço inicial da função a ser executada. Quando a instrução RET (B5) for executada, o fluxo de controle é redirecionado para o endereço armazenado no topo da pilha, ou seja, o endereço da função ADD (B14). Da mesma maneira a chamada de função localizada em A6 é ofuscada, as instruções que realizam essa ofuscação se encontram entre B8 e B10.

2.4.2 Ofuscação de Retorno

A ofuscação de retorno é uma técnica de ofuscação que remove as instruções de retorno com o intuito de dificultar a identificação do fim legítimo de uma função. Quando a ofuscação de retorno é utilizada, o fim de uma função passa a ser o fim de outra função, representado pela próxima instrução RET. Com isso, a análise intraprocedural da função ofuscada retornará resultados menos precisos do que se fosse realizada a análise no grafo legítimo.

A ofuscação de retorno pode ser implementada substituindo as instruções de retorno por outras instruções que também atuam como uma instrução de retorno. Uma instrução de retorno redireciona o fluxo de controle para o endereço armazenado no topo da pilha. Uma ofuscação de retorno em uma arquitetura x86 pode ser realizada substituindo uma instrução RET por uma instrução POP seguida de uma instrução JMP. A instrução POP é utilizada para remover o endereço de retorno do topo da pilha e armazená-lo em um registrador. E a instrução JMP redireciona o fluxo de controle do programa para o endereço armazenado no registrador utilizado pela instrução POP.

Main: A1: PUSH 0xA A2: PUSH 0xB A3: CALL Add A4: PUSH 0xC A5: PUSH 0xD A6: CALL Add A7: JMP A9 A8: NOP A9: RET Add: A10: MOV EAX, [ESP+4] A11: MOV EBX, [ESP+8] A12: ADD EAX, EBX A13: RET	Main: C1: PUSH 0xA C2: PUSH 0xB C3: CALL Add C4: PUSH 0xC C5: PUSH 0xD C6: CALL Add C7: JMP C9 C8: NOP C9: POP ECX C10: JMP ECX Add: C11: MOV EAX, [ESP+4] C12: MOV EBX, [ESP+8] C13: ADD EAX, EBX C14: POP ECX C15: JMP ECX

Figura 4: Programa que exemplifica a ofuscação de retorno

A Figura 4 mostra um exemplo de ofuscação de retorno. A coluna à esquerda mostra o programa original. Na coluna da direita mostra como as instruções que foram utilizadas para substituir as instruções de retorno. As instruções de retorno do programa original (A9 e A13) foram substituídas pelas instruções “POP ECX” e “JMP ECX” como pode ser visto nas linhas C9-C10 e C14-C15.

2.4.3 Ofuscação de Salto Incondicional

A ofuscação de salto incondicional é uma técnica que tem como objetivo substituir uma instrução de salto incondicional por uma chamada de função com a finalidade de dificultar a análise de um programa. Neste tipo de ofuscação, a análise do programa torna-se mais difícil porque ao invés de um atacante descobrir de forma trivial para onde o fluxo de controle está sendo redirecionado, terá que analisar uma função, geralmente complexa. O retorno dessa função sempre é o endereço de destino do salto incondicional. Tais funções são denominadas funções de redirecionamento.

A finalidade das funções de redirecionamento é redirecionar o fluxo de controle para o endereço legítimo, ou seja, o endereço para onde o fluxo de controle seria redirecionado caso a instrução de salto incondicional fosse mantida no programa. Entretanto, a principal característica desse tipo de função é que elas são complexas de forma a dificultar ao máximo um atacante disposto a realizar a engenharia reversa, pois a análise dessa função obrigará ao

atacante despender mais tempo e esforço para descobrir que essa função simplesmente redireciona o fluxo de controle para outro trecho do programa.

Na arquitetura x86, esse tipo de ofuscação pode ser realizado substituindo as instruções JMP por instruções CALL que invocam uma função de redirecionamento. No entanto, para garantir que tais funções redirecionem corretamente o fluxo de controle é necessário adicionar o destino da instrução de salto incondicional como parâmetro para a função de redirecionamento a fim de garantir que o programa tenha seu fluxo de controle redirecionado corretamente.

A Figura 5 apresenta um exemplo que substitui a instrução JMP localizada em A7 do programa original (coluna à esquerda) por uma chamada de função como mostrado na coluna à direita em E8. Essa instrução realiza a chamada da função Bf localizada em E14. Para garantir que o programa seja redirecionado corretamente pela função Bf é necessário informar o destino da instrução JMP que foi substituída. Com isso em mente foi adicionado a instrução "PUSH E9" (E7), ou seja, colocamos no topo da pilha o endereço da próxima instrução a ser executada RET (localizada em E9).

Main: A1: PUSH 0xA A2: PUSH 0xB A3: CALL Add A4: PUSH 0xC A5: PUSH 0xD A6: CALL Add A7: JMP A9 A8: NOP A9: RET Add: A10: MOV EAX, [ESP+4] A11: MOV EBX, [ESP+8] A12: ADD EAX, EBX A13: RET	Main: E1: PUSH 0xA E2: PUSH 0xB E3: CALL Add E4: PUSH 0xC E5: PUSH 0xD E6: CALL Add E7: PUSH E9 E8: CALL Bf E9: RET Add: E10: MOV EAX, [ESP+4] E11: MOV EBX, [ESP+8] E12: ADD EAX, EBX E13: RET Bf: E14: MOV ECX, [ESP+4] E15: JMP ECX

Figura 5: Programa que exemplifica a ofuscação de salto Incondicional

Neste exemplo, criamos uma função de redirecionamento que simplesmente redireciona o fluxo de controle para o endereço armazenado no topo da pilha. Em E14 a instrução "MOV ECX, [ESP+4]" (E14) copia o endereço armazenado no topo da pilha para o registrador ECX e em E15 o fluxo de controle é redirecionado para o endereço armazenado no registrador ECX, instrução "JMP ECX". No entanto para usufruir dessa técnica de ofuscação

é necessário que a lógica que determina o endereço de retorno seja a mais complexa possível a fim de dificultar a análise por um atacante.

2.5 INCORRUPTIBILIDADE

Segundo Collberg (COLLBERG e NAGRA, 2010), conceitualmente, as técnicas de incorruptibilidade realizam duas tarefas, a saber: detecção de modificações e resposta a detecção. Na tarefa de detecção, a técnica de incorruptibilidade monitora o programa, para ver se o seu código ou dados foram modificados, e o ambiente de execução esteja funcionando de uma forma hostil. Na resposta à detecção de modificações, uma vez que a técnica de incorruptibilidade tenha determinado que ocorreram modificações, um mecanismo de resposta entra em ação e aborta o programa de maneira conveniente. As respostas às alterações no programa podem variar da terminação do programa até uma punição ao usuário. Um exemplo dessa punição é se o programa detectar que o usuário está tentando usar um número de série pirateado, ele apaga um diretório do usuário.

É importante que os procedimentos de detecção e resposta das técnicas de incorruptibilidade estejam separados o máximo possível, tanto espacialmente, com os procedimentos distantes um do outro no executável, quanto temporalmente, com os procedimentos distantes um do outro no traçado de execução.

Segundo Boccardo (BOCCARDO, 2010), um exemplo comum da utilização dessas técnicas são os sistemas de gerenciamento de direitos digitais (DRM) que possuem tipicamente dispositivos reprodutores de mídias digitais com uma chave criptográfica embarcada e mídias criptografadas por esta chave criptográfica. Um adversário com acesso ao código do dispositivo reprodutor poderia modificá-lo para reproduzir mídias de outras regiões, assim como ter acesso a mídia em texto claro.

Segundo Boccardo (BOCCARDO, 2010), existem estratégias de incorruptibilidade para detectar violações a integridade do código, a saber: a auto-verificação de trechos de código do programa ou a inspeção lógica do programa. A inspeção lógica pode ser feita nos dados do programa ou no fluxo de execução. Dado que um desenvolvedor tenha conhecimento dos possíveis estados que seu programa pode atingir, na inspeção lógica o desenvolvedor pode inserir rotinas para verificar se uma variável ou o resultado de uma função está em conformidade com o programa. A auto-verificação de trechos de código do programa pode ser classificada em estática ou dinâmica. Na estática, a verificação é feita em

tempo de carregamento do programa, na dinâmica a verificação é feita em tempo de execução.

No que se refere as respostas fornecidas quando da detecção de violações a integridade do código pela estratégia de incorruptibilidade, elas podem ser: terminar o programa, restaurar, degradar resultados, degradar o desempenho, reportar o ataque e até mesmo uma punição ao usuário, como já mencionado no início da seção, um exemplo dessa punição é se o programa detectar que o usuário está tentando usar um número de série pirateado, ele apaga um diretório do usuário. Segundo Collberg (COLLBERG, 2010), na ação de terminar o programa, o programa será encerrado, mas deverá passar um tempo entre a detecção e o encerramento, para impedir que o atacante encontre facilmente a localização do código de detecção. Na ação de restauração, o programa é restaurado para seu estado correto pelo conserto do código alterado e reajuste de qualquer estrutura de dados corrompida. Na ação de degradação de resultados, é retornado resultados incorretos do programa, Os resultados poderão se deteriorar ao longo do tempo, para evitar que o atacante perceba que ele foi descoberto. Na ação de degradação de desempenho, o programa executa mais lentamente ou consome mais memória. Na ação de reportar o ataque, é possível avisar ao servidor da rede sobre o ocorrido, caso uma conexão com a internet esteja disponível. Na ação de punição do usuário, o usuário pode perder os seus arquivos pessoais.

Na Subseção 2.5.1, é apresentado a visão geral das técnicas de incorruptibilidade de auto-verificação de trechos de código do programa. Na subseção 2.5.2 é apresentado a visão geral das técnicas de incorruptibilidade de inspeção lógica.

2.5.1 Auto – Verificação

As técnicas de incorruptibilidade de auto-verificação de trechos de código do programa utilizam o algoritmo de introspecção para detectar a violação de integridade de um programa através do uso de *hash* (resumo criptográfico). O *hash* é inserido no código do programa, durante ou antes da execução de algum trecho de código, como apresentado na Figura 6. O cálculo *hash* de um determinado trecho de código é executado. Em seguida, o resultado do cálculo *hash* executado é comparado com o valor de *hash* esperado. Caso o valor seja diferente do valor esperado, o programa pode acionar a resposta que lhe foi programada.

Segundo Collberg (COLLBERG, 2010) a ideia da técnica de incorruptibilidade de auto-verificação é terem múltiplos verificadores de trechos de código, e que os verificadores

chequem uns aos outros, de forma que nunca baste apenas desabilitar um deles, será necessário desabilitar muitos ou todos eles.

```
ini = endereço_inicial;
fim= endereço_final;
h=0;
Bloco = Trecho(ini, fim)
        H_novo = calcula_hash(Bloco_novo)
        Se h é diferente de H_novo
                Restauramarcadagua( );
```

Figura 6: Exemplo de Algoritmo de Auto-verificação

Segundo Boccardo (BOCCARDO, 2010), uma técnica de ataque contra auto-verificadores de *hash* consiste em varrer o código a procura de instruções que comparam um cálculo de *hash* com um valor aleatório. Em programas que são distribuídos de forma distinta, um ataque de diferença poderia ser utilizado para localizar as exatas posições dos cálculos de *hash*. Uma estratégia para conter este problema consiste na duplicação de cada região que está sendo efetuado o cálculo de *hash* para que ao invés de comparar o resultado com um valor aleatório, comparar o resultado do cálculo de uma região com o cálculo da região duplicada. Uma desvantagem desta estratégia é a duplicação do código do programa. As funções que calculam o *hash* também podem ser alvos de ataques de varredura por padrão, assim, é extremamente importante que estas funções estejam escondidas (o quanto possível) no código.

2.5.2 Inspeção lógica

Segundo Collberg (COLLBERG, 2010), as técnicas de incorruptibilidade de auto-verificação de trechos de código possuem dois problemas fundamentais. Primeiro, as técnicas de auto-verificação realizam operações que são altamente incomuns, isto é, programas reais tipicamente não leem seu próprio segmento de código. Segundo, as técnicas de auto-verificação só verificam a validade do próprio código e o fato de estas técnicas não tratarem alterações no comportamento ou dados do programa. Um exemplo apresentado em (BOCCARDO,2010), expõe que um adversário pode temporariamente alterar o valor de um registrador que contém o resultado de retorno de uma função, antes que a função realize o

retorno, para que não haja a necessidade de alteração no código do programa. Métodos que inspecionam a corretude dos dados e do fluxo de controle visam a atender as deficiências dos métodos de auto-verificação de trechos de código que utilizam cálculo *hash*.

Segundo Boccardo em (BOCCARDO, 2010), outra técnica de incorruptibilidade consiste em remotamente detectar e responder a violações de integridade. Este cenário visa à proteção de um programa que executa em um local não confiável (lado do cliente) monitorado por comunicação com um programa que executa em um local confiável (lado do servidor). O servidor além de fornecer recursos aos pedidos do cliente, oferece mecanismos para verificar a integridade do programa e responder à violações. Em um sistema cliente-servidor em que o cliente espera recursos advindos do servidor, uma maneira fácil de responder a uma violação de integridade consiste na interrupção da comunicação. A tarefa de verificação não é básica quanto a tarefa de resposta pois o servidor não acessa diretamente o código do lado do cliente.

O servidor poderia requisitar o *hash* de um determinado trecho de código através do canal de comunicação, porém nada garante que a resposta do cliente seria legítima. A resposta pode ser forjada pela modificação do código cliente ou do ambiente de execução ou pela interceptação e substituição das mensagens de rede.

Soluções para verificar integridade de modo remoto são baseadas no compartilhamento de código entre o cliente e do servidor, levando em consideração que códigos mais críticos estejam no servidor. O balanceamento do compartilhamento reflete na carga computacional e na latência da comunicação. Em situações que grande parte do código está contida no servidor significaria uma carga computacional alta no servidor e uma latência de comunicação alta para o cliente. Caso contrário, ou seja, a maior parte de código contida no cliente refletiria uma latência de comunicação baixa para o cliente e uma carga computacional baixa para o servidor. Contudo, nesta situação é difícil o servidor garantir que o código do cliente não foi modificado. As técnicas para incorruptibilidade remota são niveladas em um balanceamento intermediário entre carga computacional e latência de comunicação.

2.6 REDES DE SENSORES SEM FIO

As Redes de Sensores sem Fio são sistemas compostos por centenas de milhares de dispositivos de baixo custo e tamanho reduzido, alimentados por baterias e dotados de capacidades distribuídas de processamento, sensoriamento e comunicação sem fios. RSSFs

típicas são compostas por um ou mais nós sorvedouro (também chamados de estações base) e vários nós sensores distribuídos por uma grande área geográfica de interesse (a área de monitoramento). Os dados são coletados por nós sensores a partir da área de monitoramento, e então transmitidos para o nó sorvedouro, tipicamente utilizando um protocolo de comunicação de múltiplos saltos (*multihop*).

Uma RSSF é um tipo de rede ad hoc utilizada para monitorar um ambiente, ou seja, coletar grandezas físicas tais como, temperatura, luminosidade e umidade. Os nós de uma RSSFs são dispositivos, denominados sensores, capazes de transformar propriedades físicas de um ambiente em sinais elétricos. Por exemplo, um nó sensor de uma RSSF é capaz de transformar o grau de luminosidade de um ambiente em sinais elétricos que em seguida são convertidos em sinais digitais para que possam ser manipulados por computadores. Os sensores são dispositivos que possuem recursos limitados, ou seja, pouco poder de processamento, pouca memória disponível e são alimentados por bateria. Por conta do baixo custo de produção desses sensores, as RSSFs podem ser compostas por um grande número de nós, frequentemente da ordem de centenas a milhares desses dispositivos que são dispostos de forma distribuída a fim de fornecer informações para diversas aplicações.

Exemplos de aplicações que podem se beneficiar do uso de RSSF são: o Monitoramento estrutural (SHM) (XU *et al.*, 2004), o monitoramento do habitat (MAINWARING *et al.*, 2002), da vida selvagem (LIU *et al.*, 2004) e ambiental (WERNER-ALLEN *et al.*, 2006), monitoramento de condição de máquinas (GAO e FAN, 2006), sistemas de vigilância (ARORA *et al.*, 2004), acompanhamento médico (SHNAYDER *et al.*, 2005), rastreamento (CHEN *et al.*, 2010), entre outros.

Em uma RSSF, cada nó sensor engloba componentes de software e hardware. O hardware do nó sensor possui quatro componentes principais: (i) um subsistema de sensoriamento, incluindo um ou mais sensores (com conversores analógico-digitais associados) para aquisição de dados, (ii) um subsistema de processamento, incluindo um microcontrolador e memória, possibilitando a execução de protocolos complexos e também o processamento de dados local pela aplicação, (iii) um subsistema de comunicação sem fio, para comunicação dos dados, e (iv) um subsistema de fornecimento de energia. Os componentes de software consistem na lógica da aplicação, protocolos e sistemas operacionais, que em conjunto coordenam as tarefas da RSSF.

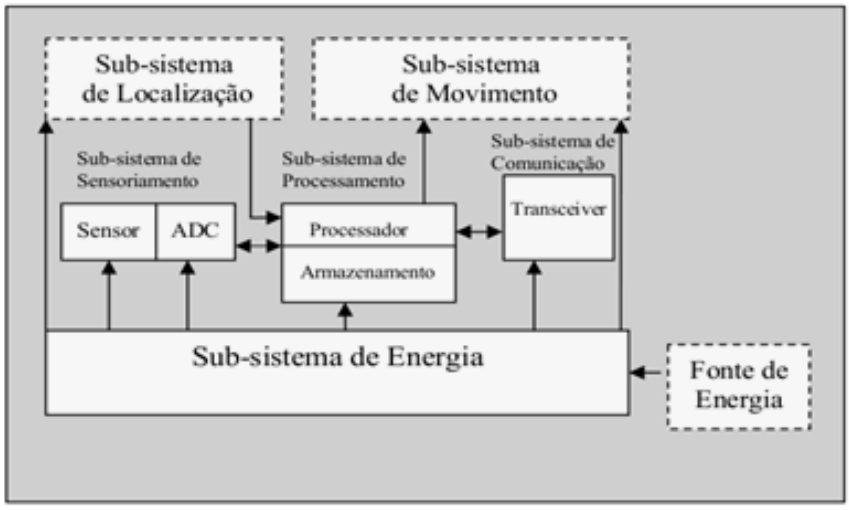

Figura 7: Componentes de Hardware de um nó sensor de uma RSSF

Uma *RSSF*, de forma geral, é constituída por apenas dois tipos de nós: nós sensores comuns e nó sorvedouro, também chamado de estação base. Os nós comuns são responsáveis por captar os dados do ambiente e repassá-los para a estação base que por sua vez armazena esses dados em local seguro e repassa esses dados para uma aplicação ou então encaminha esses dados para outras redes já que a estação base pode servir como ponto de conexão entre a *RSSF* e as demais redes.

As restrições de hardware e de energia dos sensores fazem com que o projeto de uma *RSSF* seja muito influenciado por sua aplicação. Segundo Delicato (DELICATO, 2005) um projeto de uma *RSSF* visa atender apenas os requisitos mínimos da aplicação, de forma a consumir menos energia e, assim, maximizar seu tempo de vida. Outra restrição em relação ao desenvolvimento de aplicações para *RSSFs* é em relação à limitação de recursos computacionais dos sensores, ou seja, as aplicações devem ser desenvolvidas levando em consideração à disponibilidade de memória e o poder de processamento dos sensores. Outros aspectos que influenciam um projeto de *RSSFs* são: tolerância a falhas, escalabilidade, topologia de rede, meio de transmissão, consumo de energia e segurança.

2.7 SEGURANÇA EM REDES DE SENSORES SEM FIO

Essa seção apresenta a transcrição textual sintetizada de parte da dissertação de mestrado de (SALMON, 2011) sobre o conceito de segurança em Redes de Sensores sem Fio, nas páginas

24 e 25, maiores informações sobre esse conceito, remeter-se a dissertação de mestrado de (SALMON, 2011).

Uma rede ou um sistema de comunicação pode ser considerado seguro se atender aos seguintes requisitos de segurança: (i) confidencialidade, que é a garantia de proteção de uma informação armazenada ou transmitida quanto a divulgação a uma entidade não autorizada; (ii) autenticação, que é o método de comprovação da identidade de um parceiro de comunicação ou autor de mensagem; (iii) integridade, que é a garantia de proteção de informações armazenadas ou em trânsito contra a modificação por uma entidade não autorizada; (iv) irretratabilidade, que é a garantia de que determinada entidade que tenha transmitido ou recebido uma mensagem não alegue que não a transmitiu ou não a recebeu; (v) disponibilidade, que representa a garantia de que determinado recurso ou o próprio sistema esteja sempre disponível para as entidades autorizadas; (vi) controle de acesso, que representa a garantia de que somente as entidades autorizadas tenham acesso a um determinado recurso e que essas autorizações não sejam modificadas indevidamente; e (vii) auditoria, que é a garantia do armazenamento de informações sobre a utilização de recursos do sistema (VIANNA 2006, SU *et al.* 2007).

Corroborando com os requisitos anteriores, para que um sistema seja considerado seguro, a segurança deve estar integrada em todos os aspectos do projeto de um sistema (PERRIG *et al.* 2004), o que se torna ainda mais evidente no contexto de uma RSSF. Nas RSSFs, a garantia da segurança pode ser dificultada por vários fatores, tais como: (i) restrições de recursos (energia, memória e processamento), afetando o requisito de disponibilidade; (ii) possibilidade de captura ou destruição dos sensores, o que pode afetar todos os requisitos de segurança (iii) utilização de comunicação sem fio, afetando os requisitos de confidencialidade e controle de acesso; e (iv) necessidade de garantir a escalabilidade em redes com uma grande quantidade de sensores, o que afeta diretamente o requisito de disponibilidade (SHI *et al.* 2004). Estas restrições inerentes às RSSFs tornam inviável a aplicação direta das soluções de segurança propostas para redes sem fio de computadores tradicionais (SILVA, 2005).

2.7.1 Ataques em Redes de Sensores sem fio

As RSSFs geralmente estão dispostas em áreas abertas, desprotegidas e às vezes até hostis tornando a vigilância dos nós dessa rede uma tarefa praticamente impossível. Com isso, os nós da RSSFs são vulneráveis à diversos tipos de ataques (SALMON *et al.*, 2010), como por

exemplo, o ataque Buraco Negro (*Black Hole*) (KARLOF e WAGNER 2003), o ataque de Encaminhamento Seletivo (*Selective Forwarding*) (KARLOF e WAGNER 2003), o de Inundação da rede (*Flooding*) (MARGI et al. 2009), o ataque de Buraco de Verme (*Wormhole*) (MARGI *et al.* 2009), o ataque Nós irmãos (*Sybil Nodes*) (KARLOF e WAGNER 2003) e o ataque de comprometimento de nó (*Man-At-The-End* (MATE)) (BOCCARDO *et al.,* 2010). Para este trabalho, somente é detalhado o ataque de comprometimento de nó.

O ataque de comprometimento de nó (*Man-At-The-End* (MATE)) (BOCCARDO *et al.,* 2010), ocorre em cenários não supervisionados onde a presença humana é limitada, quando um atacante for capaz de capturar um nó dessa rede e de efetuar a leitura da sua memória. Nesse tipo de ataque, um atacante deseja capturar um ou mais desses sensores, com o intuito de analisar tanto o software quanto o hardware desses nós a fim de obter alguma vantagem. Por exemplo, um atacante de posse de um ou mais nós sensores pode aclamar-se como o autor do software embarcado nesses sensores ou extrair seus módulos proprietários e utilizá-los em outros projetos, infringindo o direito de propriedade intelectual deste software. Daí a necessidade de prover meios para garantir ou rastrear a propriedade intelectual do software embarcado nos nós sensores das RSSFs.

2.8 CONSIDERAÇÕES FINAIS

Nesse capítulo foram apresentados os conceitos básicos de engenharia reversa, ofuscação de código, Incorruptibilidade, Propriedade intelectual, redes de sensores sem fio e segurança de redes de sensores sem fio. Para cada um dos conceitos apresentados existem técnicas que visam a proteção da propriedade intelectual de software. Tais técnicas podem ser combinadas e são a principal inspiração para o arcabouço de software para aplicação de uma marca d'água de software que utiliza técnicas de ofuscação de código e técnicas de incorruptibilidade para a proteção da propriedade intelectual de um software, apresentado a seguir no Capítulo 4.

3 REVISÃO BIBLIOGRÁFICA

Esse capítulo apresenta uma parcela significativa da literatura disponível acerca de técnicas de proteção de propriedade intelectual de software.

A coleta de referências aqui apresentadas apoia-se em um levantamento inspirado em revisão sistemática, apresentando os principais trabalhos que respondiam as questões de pesquisas estabelecidas no processo de busca e que depois possibilitaram relacionar os trabalhos encontrados na literatura sobre as soluções mencionadas no trabalho proposto. Na seção 3.1 é apresentada o processo do levantamento bibliográfico inspirado em uma revisão sistemática. A seção 3.2 apresenta as técnicas de proteção de propriedade intelectual de software. Na seção 3.3 é apresentado os ataques sofridos pelo software que protege a propriedade intelectual. Finalmente, na seção 3.4 são apresentados os trabalhos relacionados ao trabalho proposto.

3.1 LEVANTAMENTO INSPIRADO EM REVISÃO SISTEMÁTICA

O levantamento dos trabalhos relacionados foi inspirado em uma revisão sistemática de literatura baseada em uma metodologia previamente introduzida na ciência da computação pelo trabalho *de Kitchenham, B. et al* (KITCHENHAM *et al.,* 2009). O trabalho de *Kitchenham, B. et al*, fornece um conjunto bem definido de passos realizados de acordo com um protocolo pré-definido. Nas seções, 3.1.1, 3.1.2, 3.1.3, 3.1.4 e 3.1.5 são apresentadas todas as etapas do processo adotado para realizar a revisão sistemática da literatura. Nas seções 3.2 e 3.3 são apresentados os trabalhos encontrados na literatura que abordam as técnicas e ataques de proteção de propriedade intelectual de software. Finalmente, na seção 3.4 são apresentados os resumos dos trabalhos relacionados.

3.1.1 Questões de pesquisa

Diferentemente das técnicas tradicionais de segurança da informação que tentam evitar o acesso a essas informações, as técnicas de proteção de software são utilizadas para proteger o software que o atacante já tenha obtido acesso ao mesmo. As técnicas de proteção de propriedade intelectual de software são um caso particular das técnicas de proteção de software que visam garantir a propriedade intelectual de um software. O objetivo principal da

nossa revisão é fazer um levantamento dos trabalhos existentes na literatura sobre as técnicas de proteção de propriedade intelectual de software e dos ataques que podem vir a sofrer um software que necessita proteger a sua propriedade intelectual.

O primeiro passo da nossa revisão sistemática da literatura é a tradução do objetivo da análise para as seguintes questões de investigação:

Q1: Existem técnicas que protejam a propriedade intelectual de um software?

Q2: Quais são os ataques existentes que comprometem a propriedade intelectual de software?

3.1.2 Processo de Busca

Esta etapa consiste em efetuar um processo de busca automatizada na internet para recuperar toda a literatura relevante para responder as questões de pesquisa mencionadas na seção anterior. Esse processo é composto por duas etapas. Na primeira etapa, as bases de dados que podem fornecer os estudos relevantes para a análise devem ser especificadas, e a segunda etapa foi decidir em qual base de dados procurar.

Nesta revisão, assume-se que todas as obras publicadas estão disponíveis on-line uma vez que este é um senso comum em ciência da computação. Nesta revisão, as bases de dados eletrônicas utilizadas são: Scopus, IEEE Xplore e ACM Digital Library.

Após a definição das bases de dados, passou-se à definição de grupos de termos de pesquisa conforme apresentado na tabela 2. Cada grupo contém termos de busca que são ou sinônimos (diferentes formas da mesma palavra), ou termos que têm significado semântico semelhante ou afim dentro do campo. Nesta busca, foi assumido somente os documentos da subárea da ciência da computação e da engenharia. Os termos utilizados na confecção da consulta são apresentados na tabela 2

.

	Grupo1	Grupo2	Grupo3	Grupo 4
Termo1	*Software*	*Intelectual property*	*Protection*	*attacks*
Termo 2	*Program*		*Security*	*analysis*
Termo 3	*Code*			
Termo 4	Application			

Tabela 2: Termos de pesquisa

A definição da *string* de busca foi formada através da combinação dos termos presentes na tabela 2. O efeito da *string* de busca foi a de que todos os artigos incluíram pelo menos alguma combinação dos termos do primeiro e do segundo grupo, com algum termo de um dos grupos restantes (Grupos 3 e 4). Um exemplo de *string* de busca possível é "*Code Intelectual Property Protection*". Só um termo de cada grupo poderia ser usado por vez, evitando assim pesquisas que contivessem termos como "*Program Application*" por exemplo. Para modelar essas regras, usou-se a representação com a Tupla [*Gn, Tn*], onde *Gn* representa o n-ésimo grupo e *Tn* o n-ésimo termo daquele grupo. Portanto [G1, T2] apresentaria a *string* "*Program*". Entre os termos dentro de um mesmo grupo usou-se o conector OU, indicando que só um deles seria utilizado por vez. Entre as combinações requeridas (entre grupos) o AND. A forma geral da string de busca é apresentada abaixo:

RQ1 - (([G1, T1] OR [G1, T2] OR [G1, T3] OR [G1, T4]) AND ([G2, T1]) AND ([G3, T1] OR [G3, T2]))

RQ2 - (([G1, T1] OR [G1, T2] OR [G1, T3] OR [G1, T4]) AND ([G2, T1]) AND ([G3, T1] OR [G3, T2]) AND ([G4, T1] OR [G4, T2]))

3.1.3 Critérios de inclusão e exclusão

O objetivo desta etapa é estreitar progressivamente o número de artigos encontrados na etapa de pesquisa para uma coleção de artigos de alta qualidade considerada relevante para responder à questão de pesquisa. Para diminuir o número de artigos encontrados na pesquisa sobre proteção de propriedade intelectual de software, utilizou-se alguns critérios de seleção de artigo. Critérios de seleção de artigos são utilizados para determinar quais artigos são incluídos ou excluídos em um levantamento de revisão sistemática da literatura. Em outras palavras, os critérios de seleção de qualidade são compostos por critérios de inclusão e critérios de exclusão.

Os critérios de inclusão e exclusão de artigos são usados para excluir os artigos que não contribuem para responder à questão de pesquisa. A seguir, vamos descrever as três etapas do processo de exclusão de artigos utilizado na nossa revisão sistemática da literatura.

Na primeira etapa, simplesmente eliminou-se os artigos que foram encontrados na etapa de pesquisa com base na informação fornecida no resumo. Os artigos só foram mantidos para posterior processamento se o resumo satisfizesse o critério de inclusão principal: o foco principal do artigo são as técnicas e ataques à proteção de propriedade intelectual. Os artigos com poucas informações no resumo foram mantidos temporariamente na lista para serem avaliados na próxima etapa. Note-se que, nesta etapa, não foi considerada a qualidade dos trabalhos.

Na segunda etapa, eliminou-se os artigos que não estavam diretamente relacionados com os termos de busca (apresentados na tabela 2) sobre técnicas e ataques à proteção de propriedade intelectual. Esses artigos que embora tivessem os termos da *string* de busca no resumo, técnicas e ataques à proteção de propriedade intelectual representava um tema menos importante do trabalho.

Na terceira etapa, os artigos restantes foram submetidos a uma triagem de qualidade, onde foram eliminados os estudos que não atendem aos critérios de qualidade seguintes:

QC1 - Existe uma declaração clara dos objetivos da pesquisa?

QC2 - É proposto um sistema / algoritmo que pode ser aplicado em um cenário real?

QC3 - Os estudos de simulação / experiência são cuidadosamente analisados e explicados, e os resultados dos testes reforçam as ideias apresentadas nos trabalhos?

3.1.4 Análise e Coleta de dados

O objetivo do processo de coleta de dados foi o de reunir os dados necessários para responder as questões de pesquisa de uma forma eficiente. Para garantir a qualidade dos dados, foram instituídos os seguintes critérios:

1. Obras são publicadas nas áreas de ciência da computação e da engenharia.
2. A linguagem para publicação deve ser em Inglês.
3. Os trabalhos são publicados no período de 2008- Fev/2015.

Após o processo acima, a extração de dados foi posteriormente processada para extrair informações-chave como parte da etapa de síntese da revisão. Da mesma forma para (KITCHENHAM *et al.*, 2009), os dados extraídos de cada estudo foram os seguintes:

- Informações dos autores

  - nome dos autores
  - Instituição e
  - o país onde ele está situado.
- A fonte (revista ou conferência) e referência completa.
- Resumo do estudo, incluindo as principais questões de investigação e as respostas.
- Os aspectos técnicos relacionados com a proteção de propriedade intelectual de software que foi abordado no trabalho, incluindo modelagem, as soluções propostas, avaliação da qualidade.
- Os resultados e conclusões.

3.2 RESULTADOS DA BUSCA DE TÉCNICAS DE PROPRIEDADE INTELECTUAL

Esta subseção apresenta os resultados da revisão sistemática da literatura sobre a busca de técnicas de proteção de propriedade intelectual de software. Existem 557 artigos totalizados que foram identificados como potencialmente relevantes depois da pesquisa a base de dados. Antes de aplicar os critérios de inclusão e exclusão, os artigos duplicados foram removidos a partir do primeiro conjunto de resultados preliminares. Então, todos os resumos dos artigos restantes foram armazenados para processamento posterior. Depois de aplicar o primeiro critério apresentado na Seção 3.1.3, o conjunto de trabalhos foi reduzido a 29 artigos. Foram excluídos 528 durante a revisão dos resumos. Durante a seleção dos artigos completos, foi necessário recuperar 29 artigos, onde 4 artigos não puderam ser recuperados, deixando um total de 25 artigos para a triagem de qualidade.

Durante a triagem de qualidade, 11 artigos foram excluídos, deixando um total de 14 artigos para a próxima etapa de processamento, que é conhecida como a análise e coleta de dados apresentados na Seção 3.1.4. Nesta etapa, as regras mais restritas foram aplicados aos artigos restantes. Eventualmente, havia um total de 14 artigos incluídos em nossos estudos de pesquisa, sendo 3 artigos de esquemas de marca d'água de software, 7 artigos de esquema de marca d'água de software combinados com ofuscação de código, 3 artigos de esquema de marca d'água combinado com *tamper-proofing* e 1 esquema de marca d'água utilizando esteganografia. Um fluxograma do processo de busca é mostrado na Figura 8, formatado de acordo com as diretrizes de instrução PRISMA (MOHER *et al.,* 2009). Como mencionado anteriormente, o número final de artigos incluídos neste estudo foi de 14. Mesmo sem responder diretamente à questão de pesquisa nesta etapa, esta questão 1 ainda será utilizada

para classificar os artigos incluídos. Há 14 artigos que estão intimamente relacionados com a questão de pesquisa listada.

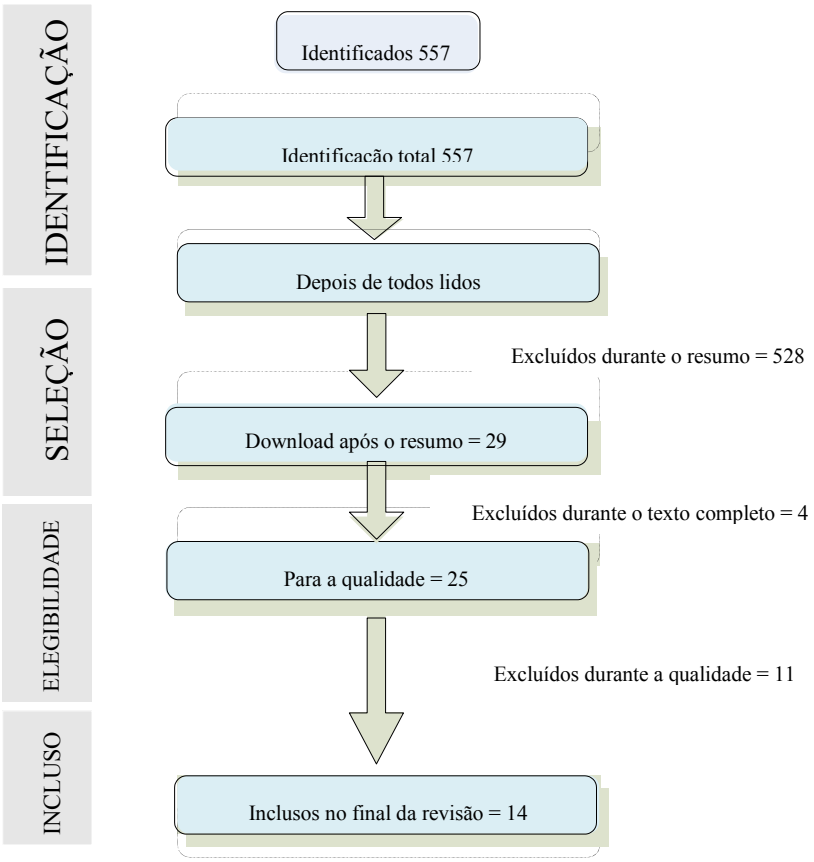

Figura 8 - Fluxograma do processo da revisão sistemática

Na subseção 3.3, são apresentadas as técnicas encontradas na literatura que são usadas para proteger a propriedade intelectual de software com o objetivo de responder a RQ1. Na subseção 3.4, são apresentados os ataques que podem sofrer o software que protege a propriedade intelectual com o objetivo de responder a RQ2. Finalmente, na subseção 3.5, são apresentados os resumos dos trabalhos relacionados.

3.3 TÉCNICAS DE PROTEÇÃO DE PROPRIEDADE INTELECTUAL DE SOFTWARE (RQ1)

As técnicas para proteger a propriedade intelectual de software têm por objetivo dificultar a cópia e a distribuição ilegal de um software. Foram encontradas 2 técnicas para proteger a

propriedade intelectual de software: marca d'água de software (CHEN e CHAOQUAN, 2012), ALITAVOLI *et al.,* 2013) e (MAHALAXMI *et al.,* 2012) e marca de nascença de software (WANG *et al*., 2012) e (MA *et al.,* 2012). Também foram encontrados na literatura trabalhos que combinavam uma dessas duas técnicas citadas anteriormente com técnicas capazes de esconder a própria marca d'água, como: esteganografia (JEON *et al.,* 2012), *tamper-proofing* (KHIYAL *et al.,* 2010) e ofuscação de código (ZENG *et al*., 2011), (XU *et al*., 2012), (CHEN e CHAOQUAN, 2012), (ARTHY *et al.,* 2013) e (EBERHARDT *et al.* 2007).

3.3.1 Marca d'água de Software

De acordo com o levantamento na literatura recente, podem ser vistos nos trabalhos de (CHEN e CHAOQUAN, 2012), (ALITAVOLI *et al.*, 2013) e (MAHALAXMI *et al.*, 2012) reiterando o que foi apresentado na seção 2.1.1, que a marca d'água de software é definida como sendo a técnica que incorpora um identificador único em um trecho de software, que serve para desencorajar os ladrões de software, fornecendo um meio de identificar o proprietário do software e / ou a origem do software roubado (BOCCARDO *et al.,* 2010). Esta técnica ajuda os proprietários de software a proteger a propriedade intelectual do software, sendo capaz de provar o direito autoral do software.

Assim, a marca d'água pode ser usada para garantir a propriedade do software, onde cada cópia legal do software é marcada com o mesmo identificador, ou seja, a marca de autoria. Além disso, a marca d'água pode ser usada para rastrear a origem de distribuição ilegal, onde cada cópia legal do software é, respectivamente, marcada com um identificador único, ou seja, a marca de impressão digital, que identifica o comprador original (COLLBERG e NAGRA, 2010).

Uma das estratégias mais comuns para a marca d'água de software envolve a inclusão do nome do autor em um ponto específico no programa (COLLBERG e NAGRA, 2010), (WANG *et al*., 2012). Esta estratégia é vantajosa porque não aumenta significativamente a dimensão do software, nem o tempo de execução. No entanto, esta estratégia não é muito recomendada porque um atacante pode facilmente mexer com a marca d'água, alterando o nome do autor ou simplesmente removê-lo. Portanto, uma marca d'água de software deve ser furtivo, ou seja, difícil de ser localizado dentro do código, evitando a sua adulteração. A solução para fazer a marca d'água ser furtiva é a combinação da marca d'água com alguma técnica que sejam capazes de esconder a própria marca d'agua.

3.3.2 Marca de Nascença de Software

Podem ser vistos nos trabalhos de (WANG *et al*. 2012) e (MA *et al.* 2012) que a técnica de marca de nascença de software pode ser definida como uma técnica relativamente nova, e uma técnica menos popular de detecção de roubo de software. Uma marca de nascença de software é a técnica responsável por extrair informação a partir de características de um programa executável. Ao comparar as marcas de nascença do software suspeito e do software original, pode-se detectar a pirataria do software. Pode-se pensar que, se dois programas têm a mesma marca de nascença, os dois programas têm a mesma origem.

As características únicas do software que podem ser usadas para distinguir o binário de um software dos outros binários, são focadas em instruções, APIs (*Application Program Interface*) ou as relações internas no programa representados pelo grafo de fluxo de controle ou o grafo de fluxo de dados. Essas características possibilitam calcular a semelhança entre os dois softwares e impedir a distribuição ilegal. Quando se suspeita de um binário a ser ilegalmente modificado ou roubado, a similaridade pode ser medida entre o software original e o software suspeito, comparando as marcas de nascença dos dois softwares. Os trabalhos encontrados na literatura que apresentam soluções de marca de nascença de software não são detalhados, pois marca de nascença está fora do escopo do nosso trabalho.

3.3.3 Marca d'água combinado com outras técnicas de proteção de software

Foram encontrados na literatura trabalhos sobre aplicação de marca d'água de software com ofuscação de código (ZENG *et al*., 2011), (XU *et al*., 2012), (CHEN e CHAOQUAN, 2012), (ARTHY *et al.,* 2013) e (EBERHARDT *et al.* 2007), marca d'água combinada com *tamper-proofing* (KHIYAL *et al.,* 2010), e marca d'água combinada com esteganografia (JEON *et al.,* 2012). Na seção 3.4, são detalhados somente os trabalhos que combinam a marca d'água com ofuscação de código e aquele que utiliza técnicas de incorruptibilidade (*tamper- proofing*), pois estas técnicas são empregadas na nossa solução.

3.4 ATAQUES DE PROTEÇÃO DE PROPRIEDADE INTELECTUAL DE SOFTWARE (RQ2)

Nos trabalhos de (COLLBERG *et al*. 2011), (ALITAVOLI *et al.* 2013), (PALSBERG *et al*. 2000) e (NAGRA *et al*. 2002) são apresentados os principais ataques que uma marca d'água de software pode sofrer, reiterando os conceitos apresentados na seção 2.2.1.

Os principais tipos de ataques definidos em (COLLBERG *et al*. 2011), (ALITAVOLI *et al.* 2013) e (NAGRA *et al*. 2002) são: ataque subtrativo, ataque distorcivo, ataque aditivo. No ataque subtrativo, o atacante estima a localização aproximada da marca d'água e tenta remove-la, enquanto o que resta do código continua a ser útil, a marca d'água não é mais reconhecível. No ataque distorcivo, um adversário faz a distorção uniforme através de um programa, com a intenção de que o proprietário do software já não possa reconhecer a sua marca. No ataque aditivo, o atacante acrescenta sua própria marca d'água ou substitui a marca d'água original, tornando impossível detectar qual marca d'água foi aplicada pela primeira vez e, portanto, qual é a única autêntica.

Os autores em (PALSBERG *et al*. 2000) definem que os ataques (aditivos, distorcivos, subtrativos) definidos por (COLLBERG *et al*. 2009) também podem ser chamados de ataques de transformações semânticas. As transformações semânticas podem ser obtidas, por exemplo, por meio da ofuscação. Tais transformações não alteram o comportamento de um programa, mas fazem alterar a sintaxe do programa. Como tal, eles podem facilmente remover ou distorcer marcas d'água que estão embutidos na estrutura do programa, por exemplo, nos comentários, nas constantes de cadeia, na ordem de instruções.

3.5 TRABALHOS RELACIONADOS

Os seguintes trabalhos na literatura propõem técnicas de marca d'água combinados com a técnica de ofuscação de código para dificultar a localização da marca d'água (ZENG *et al.* 2010), (XU *et al.* 2012), (CHEN e CHAOQUAN, 2012), (ARTHY *et al.,* 2013) e (EBERHARDT *et al.* 2007). No trabalho (KHIYAL *et al.,* 2010), é apresentado o trabalho de aplicação de marca d'água que utiliza ofuscação de código e técnicas de incorruptibilidade (*tamper-proofing*). As soluções de marca d'água apresentada por tais trabalhos diferem da apresentada no trabalho proposto, pois essas soluções não combinam a técnica de ofuscação de código com a técnica de aplicação de marca d'água, para criar a própria marca d'água, e sim para apenas esconder a marca d'água.

Em (ZENG *et al.* 2010) foi proposto um esquema de marca d'água baseada na frequência de cada tipo de instrução de um software. Um vetor contendo a ocorrência de cada instrução é criado e inserido no software. A identificação única caracterizada pela marca d'água é convertida em binário e armazenada em outro vetor. Em seguida, soma-se o conteúdo do vetor de ocorrência das instruções com o conteúdo do vetor que caracteriza a marca d'água, resultando em um novo vetor que será utilizado como a chave da marca d'água. A Inserção da marca d'água no código de um software é inserida segundo a chave da marca d'água criada, realizando substituição ou inserção de instruções. No caso de encontrar instruções que podem ser substituídas, o código é modificado por meio da substituição de instruções equivalentes que foram estabelecidas nos padrões de substituições. Na situação em que instruções não podem ser substituídas por instruções equivalentes, é necessária a inserção de código por meio de predicados opacos. Predicados Opacos são trechos de códigos que podem ou não ser executados inseridos no fluxo de controle do programa. A nossa proposta difere do trabalho de (ZENG *et al.* 2010) pois no nosso trabalho o processo de inserção da marca d'água é utilizado a sequência ordenada de ofuscações caracterizada como a própria marca d'água e o do ZENG é utilizando uma sequência de frequência de instruções. Com isso, o nosso trabalho tem a vantagem de dificultar a localização da marca d'água o que não é verdade em (ZENG *et al.* 2010). Este fato foi comprovado nos resultados de furtividade do capítulo 6.

A ideia de Xu em (XU *et al*., 2012) é que a marca d'água é convertida em dígitos binários. Em seguida, essa sequência binária é passada em uma função de ofuscação, que depois de ofuscada é armazenada em um vetor. A função de ofuscação é definida pela diferença de altura entre as declarações de desvio condicional presentes no código, onde a altura é definida como o número de instruções entre a instrução de salto e seu alvo. Um vetor contendo essas diferenças de altura é criado para que seja identificado os endereços onde serão inseridas as informações da marca d'água. Uma das diferenças entre a nossa proposta e o trabalho de Xu é que no nosso trabalho o processo de inserção da marca d'água é utilizando a sequência ordenada de ofuscações caracterizada como a própria marca d'água, enquanto em (XU *et al*., 2012) o processo de inserção da marca d'água é dado através da diferença de altura entre instruções de desvio condicional presentes no software o que é fácil do atacante identificar. O processo de inserção do nosso trabalho torna a marca d'água furtiva, comprovado nos resultados do capítulo 6.

Chen e Chaoquan (CHEN e CHAOQUAN, 2012) propuseram uma marca d'água de software baseado em ofuscação para software desenvolvido na plataforma .NET. A marca

d’água é convertida em uma sequência de bits. Em seguida, conhecendo o tamanho desta sequência, escolhe-se aleatoriamente uma dentre as combinações possíveis desta sequência. Esta combinação estabelece a sequência dos blocos do código de linguagem intermediário gerado pela máquina virtual da plataforma .NET. Na solução utilizada, o código é dividido em blocos de tamanhos variáveis, cujo tamanho é escolhido aleatoriamente, preservando a semântica do software. A ordem original destes blocos é ofuscada. A marca d’água é inserida, bit por bit, ao fim de cada bloco em uma instrução falsa. Em seguida, é adicionada também uma instrução de desvio incondicional entre os blocos para fazer com que o software com a marca d’água funcione como o software original. Finalmente, um novo executável é criado. Diferentemente de (CHEN e CHAOQUAN, 2012), o presente trabalho utiliza várias técnicas de ofuscação de código que substituem instruções no código que são semanticamente equivalentes. Na proposta de (CHEN e CHAOQUAN, 2012), a ofuscação é dado por meio da troca da ordem dos blocos de código.

O trabalho de (ARTHY e SARANYA, 2013) propõe um esquema de marca d’água que não altera o documento de texto, ao invés disso utiliza características do próprio texto para gerar a marca d’água. O usuário escolhe como palavra-chave uma palavra contida no texto. Em seguida, a palavra-chave é localizada em todas as posições ao longo do texto. Então, é calculado o comprimento das palavras anterior e posterior a cada aparição da palavra-chave no texto. Todos esses tamanhos são concatenados gerando a marca d’água. Diferentemente do trabalho de (ARTHY e SARANYA, 2013), o presente trabalho gera a marca d’água a partir da sequência de ofuscações e não através de características de um texto.

Os autores em (EBERHARDT *et al.,* 2007), propõem um esquema que combina ofuscação de código com marca d'água de software, a fim de proporcionar uma solução contra cópia do software. A solução proposta está relacionada principalmente em produtos de software móvel, onde pode-se contar com o hardware baseado na proteção de integridade do sistema operacional.

No trabalho de (KHIYAL *et al*., 2010), foi proposto uma técnica baseada na técnica de codificação de constantes, para melhorar o nível de proteção da marca d'água de software de um grafo dinâmico e com o objetivo de realizar a análise que avalia o efeito da integração de duas técnicas de proteção de software, tais como marca d'água de software e incorruptibilidade. A ideia básica por trás da técnica de codificação de constantes é a substituição de valores constantes por chamadas de função. O valor de retorno pelas chamadas de função é dependente dos valores das variáveis ponteiros na estrutura de dados dinâmica da mesma forma da estrutura da marca d'água. A análise experimental que foi

realizada, conclui que o processo de codificação de constantes aumenta significativamente o tamanho: (i) do código, (ii) espaço da pilha utilizada e (iii) o tempo de execução, mas ao mesmo tempo o código é resistente à adulteração e a ataques de transformações semânticas no programa. Diferentemente do trabalho de (KHIYAL *et al*., 2010), o trabalho proposto não tem um aumento significativo no tamanho do código e no tempo de execução, comprovado nos resultados do capítulo 6.

4 DESCRIÇÃO DO ARCABOUÇO SENSORWATERMARK

Nesse capítulo, é apresentado o ***SensorWatermark***, um arcabouço de aplicação de marca d'água de software utilizando técnicas de ofuscação de código e de incorruptibilidade para Redes de Sensores Sem Fio. O objetivo do ***SensorWatermark*** é inibir o roubo da propriedade intelectual do software. A marca d'água inserida pelo esquema proposto nesse trabalho é definido como sendo uma sequência ordenada de ofuscações de código aplicadas em um software, onde tal sequência é responsável por identificar quem é o autor desse software. O ***SensorWatermark*** insere uma marca d'água que dificulta a violação dos direitos de propriedade intelectual de um software embarcado nos nós sensores e caso a marca d'água seja alterada, as técnicas de incorruptiblidade implementadas possibilitam a identificação da alteração e a restauração da marca d'água. Com isso, um atacante disposto a piratear um software em um nó de uma RSSF deverá dispor de mais tempo e esforço já que a localização das informações de marca d'água podem ser difíceis de serem encontradas.

SensorWatermark combina as técnicas de ofuscação de código com a de incorruptibilidade, de forma a tornar a marca d'água inserida pelo arcabouço ***SensorWatermark*** resistente a ataques de transformações de código. As operações de inserção e detecção da marca d'água do ***SensorWatermark*** são realizadas *offline*, ou seja, antes que o código binário tenha sido implantado em um sensor.

Este capítulo está organizado em três seções: Na seção 4.1, é apresentada uma visão geral do arcabouço ***SensorWatermark***. Na seção 4.2, é apresentada a arquitetura lógica desse arcabouço. Na seção 4.3, é descrito a operação do ***SensorWatermark***. Finalmente, na seção 4.4 é apresentado o modelo de ataque do ***SensorWatermark***.

4.1 VISÃO GERAL DO ARCABOUÇO SENSORWATERMARK

A descrição dos procedimentos principais do ***SensorWatermark*** está dividida em duas operações, são elas: inserção da marca d'água (Procedimento 1, figura 9) e detecção da marca d'água (Procedimento 2, figura 9).

Antes da realização de qualquer procedimento é necessário que o proprietário do software a ser instalado no nó sensor da RSSF configure os parâmetros iniciais para a realização de ambos os procedimentos.

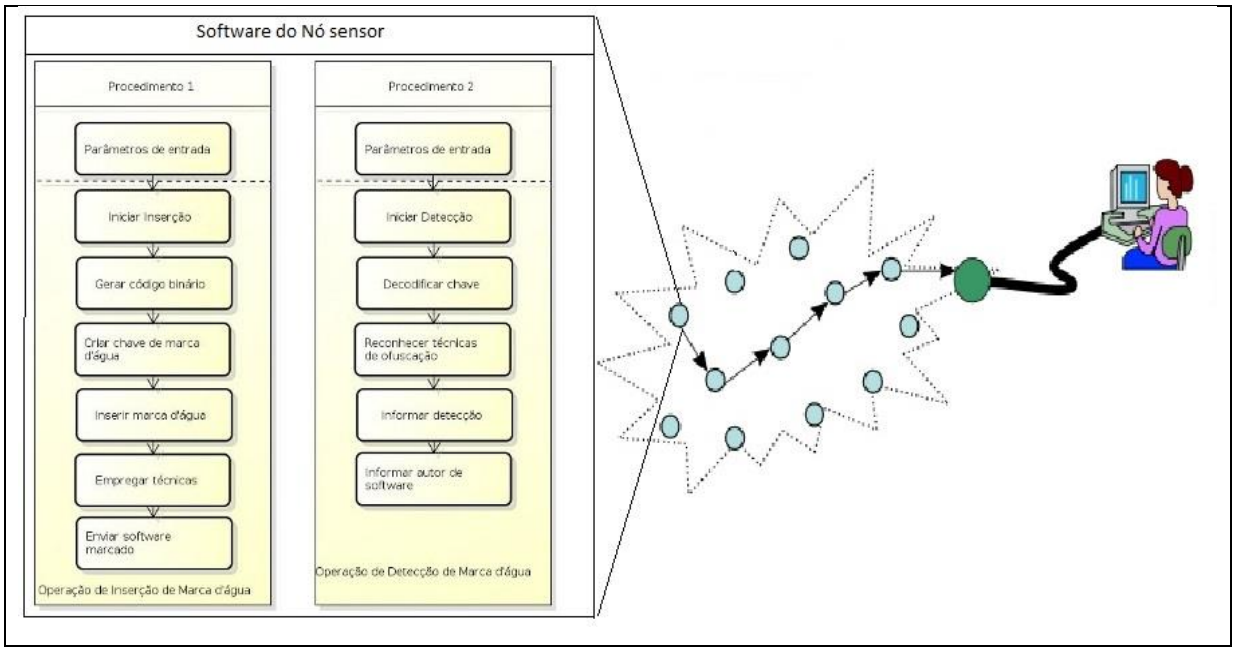

Figura 9 - Visão geral do arcabouço SensorWatermark

O ***SensorWatermark*** foi concebido com o intuito de permitir a inserção e detecção de uma marca d'água de software utilizando técnicas de ofuscação de código e de incorruptibilidade para redes de sensores sem fio. A inserção de marca d'água se faz necessária quando houver a necessidade de identificar um software através de uma marca d'água. Durante o procedimento de inserção da marca d'água, iniciado pelo proprietário do software, são fornecidas as informações de entrada para que a operação de inserção seja iniciada, como o código fonte e o proprietário do software. Em seguida, por meio do código fonte é gerado o código binário. Em seguida, solicita-se a criação de uma chave de marca d'água para posteriormente identificar o proprietário do software. Após a criação da chave, são selecionados os endereços do código onde serão empregadas as técnicas de ofuscação. Após as técnicas de ofuscação serem empregadas nos endereços selecionados, são inseridas travas lógicas antes de cada ofuscação para checar a integridade de cada trecho do código que foi ofuscado. Por fim, é retornado o software com a marca d'água inserida.

A detecção da marca d'água se faz necessária quando houver a necessidade por parte do proprietário de verificar se um software possui a marca d'água desse proprietário. No procedimento de detecção da marca d'água, iniciado pelo proprietário do software, são fornecidas as informações de entrada para que a operação de detecção seja iniciada, como o software marcado e a chave da marca d'água. Em seguida, é feita a decodificação da chave de marca d'água, verificando se nos endereços do código informados pela chave, foram empregados as técnicas de ofuscação indicadas também na chave. Se na verificação dos

endereços de código informados pela chave de marca d'água não for possível identificar a técnica de ofuscação indicada na chave, a trava lógica inserida no código na operação de inserção da marca d'água restaura o trecho do código, recuperando assim a marca d'água do proprietário. Finalmente, será retornado o autor do software.

4.2 ARQUITETURA LÓGICA

Nessa seção é apresentada a arquitetura lógica do ***SensorWatermark***, ilustrada na Figura 10. A arquitetura lógica é composta de 8 componentes (Gerenciador de Propriedade intelectual, Compilador, Gerador de Chave, Insersor de Marca d'água, Detector de Marca d'água, Ofuscador, Incorruptibilidade e Reconhecedor de Ofuscação) e duas estruturas de dados (Regras de Ofuscação e Chave da marca d'água). Os componentes do ***SensorWatermark*** relacionam-se através das seguintes interfaces: *genKey*, *embedWM*, *detectWM*, *reqObf*, *reqTP*, *recogObf* e *compile*.

As informações de entrada fornecidas pelo usuário (Modo de Operação, Plataforma de Hardware, Código Fonte e Autor do Software) e as informações de saída fornecidas para o usuário (Chave da marca d'água e o Código Binário) são descritas na seção 4.2.1. As descrições das estruturas de dados e dos componentes do ***SensorWatermark*** estão descritas respectivamente nas seções: Seção 4.2.2 e Seção 4.2.3.

4.2.1 Descrição de Entrada e Saída

As informações de entrada fornecidas pelo usuário são: Modo de Operação, Plataforma de Hardware, Código Fonte e Autor do Software. O **Modo de Operação** indica se o usuário quer inserir ou detectar uma marca d'água no software. A **Plataforma de Hardware**, indica a plataforma de hardware do sensor. O **Código Fonte** é o software em que se deseja inserir a marca d'água. No **Autor do Software** é armazenada a identificação do autor de um software, que é usado para identificar a sequência binária armazenada em *BinStr*.

As informações de saída providas para o usuário são: Chave da Marca d'água e Código Binário. A **Chave da Marca d'água** identifica a chave que compõe uma determinada marca d'água, e é composta por uma sequência de ofuscações de código e as respectivas localizações dentro do software onde tais ofuscações foram aplicadas (inserção de marca d'água) ou que foram aplicadas (detecção de marca d'água). O **Código Binário** contém o código binário do software com a marca d'água.

4.2.2 Estrutura de dados

As estruturas de dados definidas para serem utilizadas pelo ***SensorWatermark*** são: estrutura de dados Regras de Ofuscação e estrutura de dados Chave da Marca d'água.

A estrutura de dados **Regras de Ofuscação** especifica as técnicas de ofuscação suportadas pelo ***SensorWatermark*** e as regras que definem como empregar cada uma destas técnicas para uma ou mais plataformas de hardware. Em outras palavras, essa estrutura de dados especifica o conjunto de instruções que devem substituir uma dada instrução localizada em um ponto específico do software, para empregar uma técnica de ofuscação para uma dada plataforma de hardware específica. Esta estrutura de dados deve ter as seguintes informações: (i) a identificação da técnica de ofuscação, (ii) a plataforma de hardware, (iii) o conjunto de instruções a serem ofuscadas e (iv) o conjunto de novas instruções com a mesma semântica para substituir uma instrução inicial. Esta estrutura deve ser preenchida por um especialista de segurança capaz de personalizar cada técnica de ofuscação para uma determinada plataforma de hardware.

A estrutura de dados **Chave da Marca d'água** define a sequência de ofuscações de código (com base na substituição ou inserção de instruções) a ser aplicada no código do software. Essa sequência de ofuscações de código caracteriza uma marca de água. Cada item desta estrutura de dados possui as seguintes informações: (i) a sequência de dígitos binários que identifica o proprietário do software (ii) para cada um dos dígitos binários (campo anterior) é informada a localização do código binário onde a técnica de ofuscação deve ser empregada e (iii) para cada um dos dígitos binários (campo anterior) é informado a técnica de ofuscação de código que deve ser aplicada no código binário usado como entrada.

4.2.3 Componentes

Os componentes definidos para serem utilizados pelo ***SensorWatermark*** são: (i) **Gerenciador de Propriedade intelectual, (ii) Compilador, (iii) Gerador de Chave, (iv) Insersor de Marca d'água, (v) Detector de Marca d'água, (vi) Ofuscador (vii) Incorruptibidade e (viii) Reconhecedor de Ofuscação.**

O **Gerenciador de Propriedade Intelectual** é o principal componente deste arcabouço e é responsável por: (i) realizar a inicialização do procedimento de inserção e detecção de marca d'água do ***SensorWatermark***, (ii) coordenar as ações dos outros

componentes deste arcabouço (iii) receber as informações de entrada do usuário e (iv) fornecer as informações de saída para o usuário. Se a informação de entrada (Modo de Operação) for igual a Inserção de Marca d'água, o **Gerenciador de Propriedade Intelectual** aguarda as seguintes informações fornecidas pelo usuário: o Código Fonte do software que irá receber a marca d'água, o autor do software que identifica o proprietário do software e a plataforma do hardware utilizado. Por meio das informações recebidas, o componente **Gerenciador de Propriedade Intelectual** coordena as ações dos outros componentes para que a marca d'água seja inserida no software. Se a informação de entrada, modo de Operação for igual a Detecção de Marca d'água, o **Gerenciador de Propriedade Intelectual** aguarda as seguintes informações fornecidas pelo usuário: o Código binário marcado e a chave da marca d'água gerada. Então, o componente **Gerenciador de Propriedade Intelectual** coordena as ações dos outros componentes para detectar se existe uma marca d'água no software marcado.

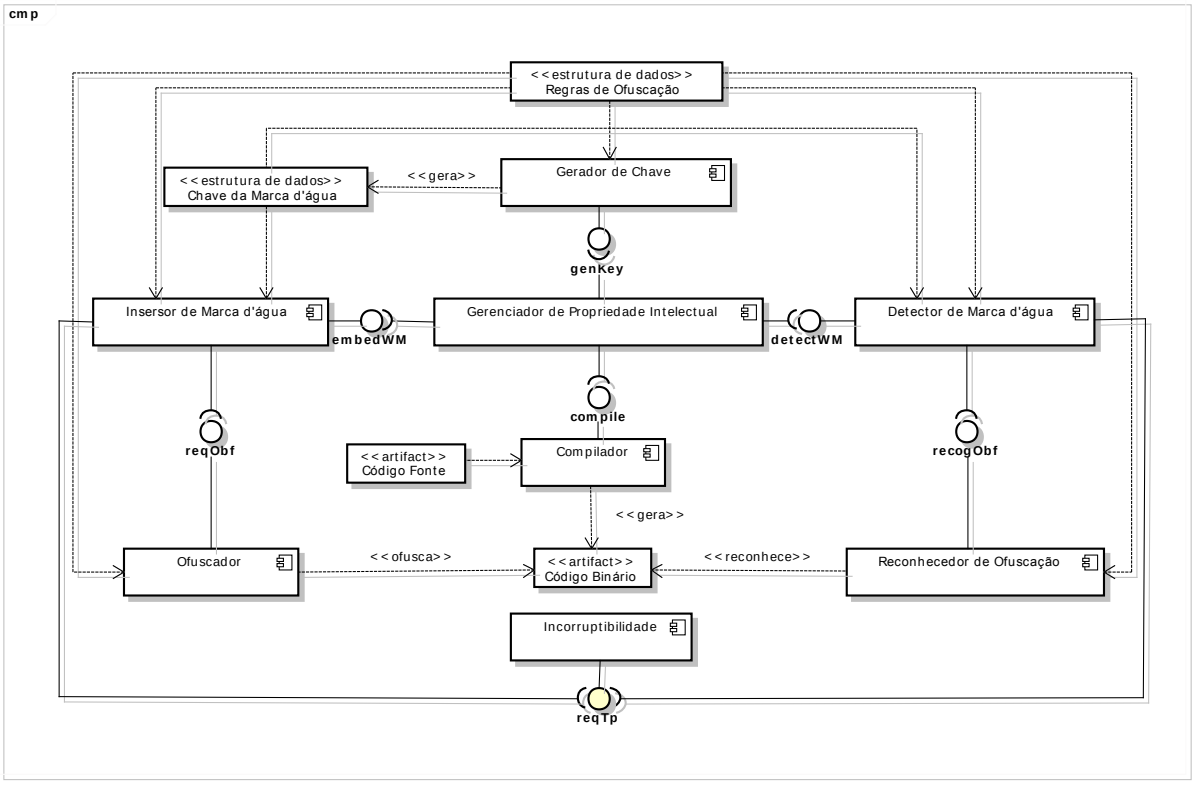

Figura 10: Arquitetura Lógica

O componente **Compilador** é o responsável pela compilação do código fonte para uma dada plataforma de hardware. O **Compilador** deve manipular o código fonte para inserir instruções NOPS para garantir que haja espaço para inserir uma marca d'água. Depois disso, o compilador gera um código binário para uma dada plataforma de hardware. Para fornecer tais funcionalidades, o Compilador provê uma interface, denominada ***compile***, que permite

que o **Gerenciador de Propriedade Intelectual** solicite que o software seja compilado em qualquer plataforma de hardware. A interface requer os seguintes parâmetros: Plataforma de Hardware e Código Fonte.

O componente **Gerador de Chave** é o responsável pela geração da chave de Marca d'água. Esta chave é gerada de acordo com a identificação do autor do software (Autor de Software) e deve ser capaz de detectar unicamente uma marca d'água, ou seja, detectar unicamente a sequência de ofuscações empregadas em um software, a fim de comprovar que o autor do software é o proprietário do software. O componente **Gerador de Chave** provê uma interface chamada ***genKey***, cujo parâmetro informa a identificação do autor de Software, a fim de gerar a Chave de Marca d'água.

O componente **Ofuscador** é o responsável por realizar as ofuscações no Código Binário para uma determinada plataforma de hardware. Para realizar tais ofuscações, esse componente provê a interface ***reqObf,*** cujos parâmetros são: Código Binário, Plataforma de Hardware, e a técnica de ofuscação que deve ser empregada no Código Binário, em conjunto com os endereços do Código Binário onde tal técnica deve ser empregada. Para o emprego de uma técnica de ofuscação, o **Ofuscador** utiliza a estrutura de dados denominada de *Regras de Ofuscação.*

O componente **Incorruptibilidade** é responsável por inserir guardas para verificar a integridade (autenticidade) da marca d'água de software via introspecção de código (COLLBERG e NAGRA, 2010). Este componente é responsável por verificar se o trecho de código que corresponde à marca d'água não se alterou. Em caso de ser alterado alguma parte do trecho de código, as instruções adulteradas são restauradas graças a Chave de Marca d'água (que contém as instruções ofuscadas juntamente com o endereço do Código Binário, onde tais instruções devem ser restauradas). O componente fornece uma interface chamada ***reqTp***, requer os seguintes parâmetros: Código Binário, Plataforma de Hardware, técnica de ofuscação de código e o primeiro endereço da instrução a ser verificado.

O componente **Insersor de Marca d'água** é o responsável por inserir uma marca d'água no código binário de acordo com a chave de marca d'água. Este componente determina qual a sequência de técnicas de ofuscação de código que devem ser aplicadas no código binário. O componente Insersor de Marca d'água fornece a interface denominada ***embedWM,*** que requer os seguintes parâmetros: Código Binário, Plataforma de Hardware, e Chave de Marca d'água.

O componente **Detector de Marca d'água** é responsável por detectar se um Código Binário possui uma determinada marca d'água. Se todas as ofuscações de código

determinadas pela chave de marca d'água são reconhecidas, o **Detector de Marca d'água** informa quem é o Autor de Software. O componente provê a interface denominada ***detectWM***, que requer os seguintes parâmetros: Código Binário, Plataforma de Hardware e Chave de Marca d'água.

O componente **Reconhecedor de Ofuscação** é o responsável por reconhecer, para uma determinada plataforma de Hardware, se uma técnica de ofuscação foi empregada em um determinado endereço do código binário. Para reconhecer todas as ofuscações, esse componente consulta a estrutura de dados denominada *Regras de Ofuscação*, identificando quais instruções devem ser encontradas a partir do endereço fornecido. Se tais instruções forem encontradas, o **Reconhecedor de Ofuscação** informa que o reconhecimento da técnica de ofuscação foi bem-sucedido, e, caso contrário, reporta uma falha. O componente provê a interface denominada ***recogObf***, que requer os seguintes parâmetros: Código Binário, Plataforma de Hardware, técnica de ofuscação e o endereço da primeira instrução a ser verificada.

4.3 OPERAÇÃO

Esta seção apresenta a operação do ***SensorWatermark,*** ilustrado na Figura 12, de acordo com a escolha do modo de operação, que pode ser de inserção ou detecção da marca d'água. Como já mencionado no início desse capítulo, o ***SensorWatermark*** opera em tempo de projeto, ou seja, antes do código binário com marca d'água ser implantado nos dispositivos.

Inicialmente, o componente **Gerenciador de Propriedade Intelectual** recebe do usuário o modo de operação. Se o modo de operação é "inserção de marca d'água de software", então o procedimento de inserção de marca d'água é iniciado. O usuário fornece as seguintes informações de entrada para o **Gerenciador de Propriedade Intelectual**: Código Fonte, Autor de Software e Plataforma de Hardware. Em seguida, o **Gerenciador de Propriedade Intelectual** solicita ao **Compilador**, através da interface *compile*, a compilação do código fonte para uma plataforma de hardware específica, gerando um código binário. Tendo finalizado a compilação, o **Compilador** envia a informação do **Código Binário** através da interface *compile* para o **Gerenciador de Propriedade Intelectual**. O **Gerenciador de Propriedade Intelectual**, por sua vez, solicita ao **Gerador de Chave**, através da interface *genKey*, a criação da chave da marca d'água, dado a informação do Autor do Software. O **Gerador de Chave** cria a chave da marca d'água (detalhes no capítulo 5), armazena-a na estrutura Chave de Marca d'água e envia-a para **Gerenciador de Propriedade Intelectual**

através da interface *genkey***.** O **Gerenciador de Propriedade Intelectual** de posse da informação da Chave de Marca d'água, solicita ao componente **Insersor de Marca d'água**, através da interface *embedWM*, a inserção da marca d'água no Código Binário. O **Insersor de Marca d'água**, por sua vez, solicita ao componente **Ofuscador**, através da interface *reqObf*, ofuscar as instruções nos endereços selecionados do Código Binário, de acordo com as informações especificadas na *Chave de Marca d'água*. Para ofuscar as instruções nos endereços selecionados do Código Binário, o **Ofuscador** consulta a estrutura de dados Regras de Ofuscação para verificar a regra de ofuscação para a plataforma de hardware especificada e para identificar o conjunto de instruções que devem substituir a instrução original. Após a aplicação de todas as técnicas de ofuscação no Código Binário, o **Insersor de Marca d'água** solicita ao componente **Incorruptibilidade** a inserção das guardas para verificação dos trechos de código ofuscados. Para realizar esse procedimento, o componente Insersor de Marca d'água informa a sequência de ofuscações de código e a localização das instruções ofuscadas por meio da interface denominada *reqTp*. Em seguida, o componente **Incorruptibilidade** informa para o **Insersor de Marca d'água**, através do *reqTp*, o código binário com as guardas e com a marca d'água inserida. Depois, o componente **Insersor da Marca d'água** informa ao **Gerenciador de Propriedade Intelectual** as informações do código binário com a marca d'água inserida através da interface *embedWM*. Finalmente, o **Gerenciador de Propriedade Intelectual** retorna o código binário marcado para o usuário, pronto para ser implantado em um nó sensor, e a Chave de Marca d'água, que pode ser usada para comprovar a autoria deste software. Na Figura 11 é apresentado todas as etapas da inserção da marca d'água.

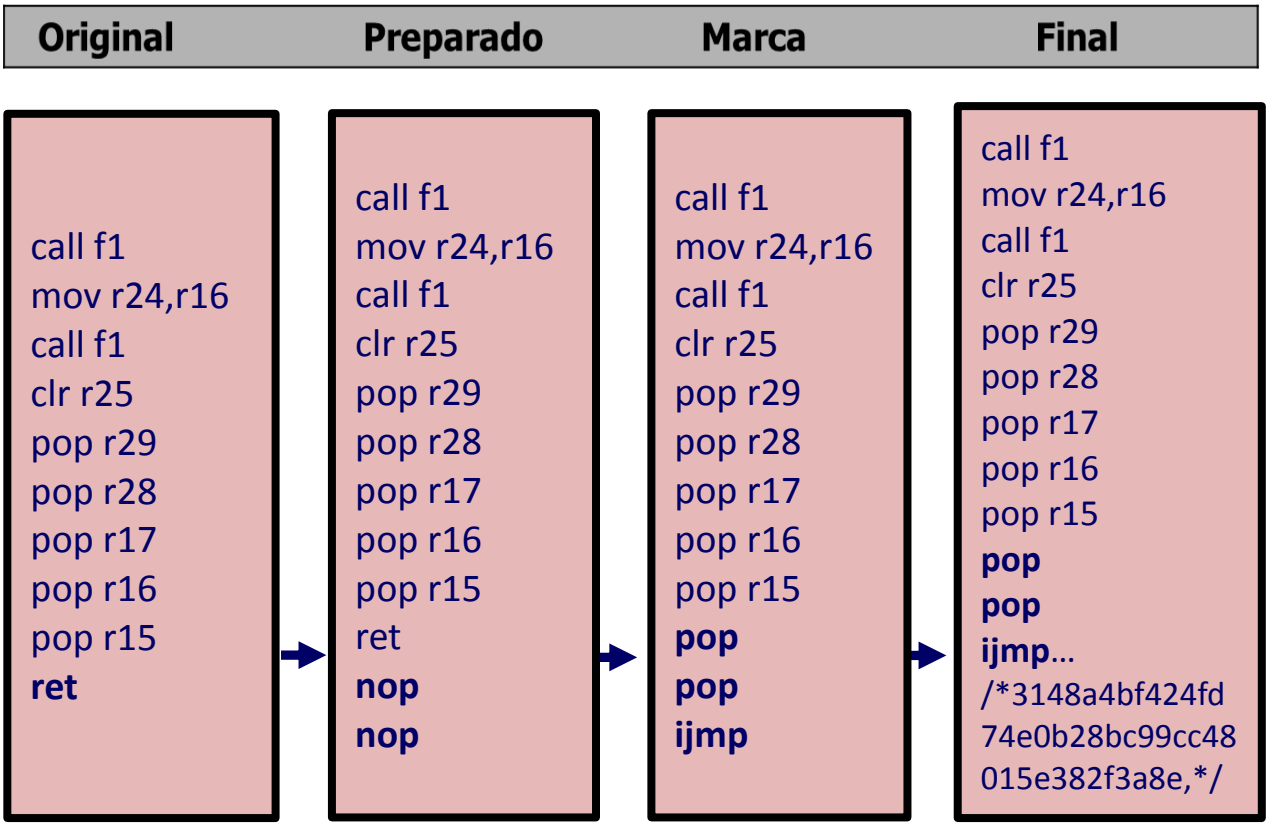

Figura 11: Operação de Inserção de marca d'água

Por outro lado, se o modo de operação é "detecção da marca d'água de software", ilustrado na Figura 13, o procedimento de detecção de marca d'água é iniciado. Neste caso, o usuário deve prover as seguintes informações de entrada para o **Gerenciador de Propriedade Intelectual**: Código Binário, Chave de Marca d'água e a Plataforma de Hardware. Em seguida, o **Gerenciador de Propriedade Intelectual** solicita ao componente **Detector de Marca d'água**, através da interface *detectWM*, para verificar se a marca d'água, especificada pela chave de marca d'água, pode ser reconhecida no Código Binário. Após o componente **Detector de Marca d'água** receber o código binário e a chave de marca d'água, este componente identifica os endereços das instruções ofuscadas e as técnicas de ofuscação utilizadas. Em seguida, o **Detector de Marca d'água** solicita ao componente **Reconhecedor de Ofuscação** o reconhecimento das técnicas de ofuscação empregadas no endereço informado através da interface *RecogObf*. O **Reconhecedor de Ofuscação**, por sua vez, consulta a estrutura de dados *Regras de Ofuscação*, tendo em mãos a chave da marca d'água, fornecendo as instruções ofuscadas para obter o conjunto de instruções originais para a plataforma de hardware especificada. Se o **Detector de Marca d'água** não reconhecer o autor

do software, através da chave de marca d'água, o **Detector de Marca d'água** solicita ao componente **Incorruptibilidade** através da interface *reqTp* a restauração da marca d'água do software, identificada através da Chave de Marca d'água. Depois, o componente **Incorruptibilidade** informa ao **Detector de Marca d'água**, através da interface *reqTp*, o Código Binário com a marca d'água restaurada para uma nova detecção. Finalmente, o componente **Detector de Marca d'água** informa ao componente **Gerenciador de Propriedade Intelectual**, através da interface *detectWM*, o autor do software.

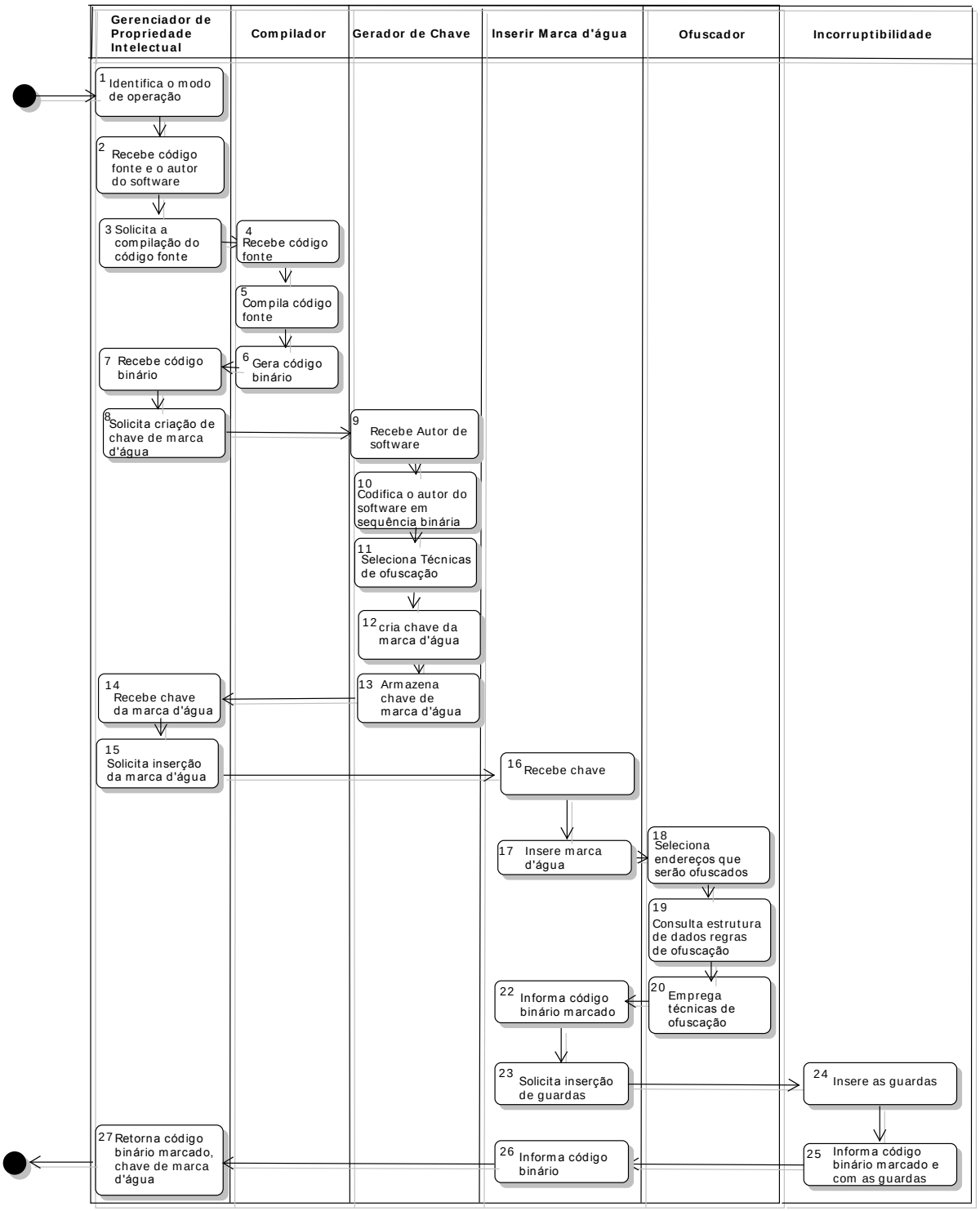

Figura 12: Diagrama da operação de inserção de marca d'água

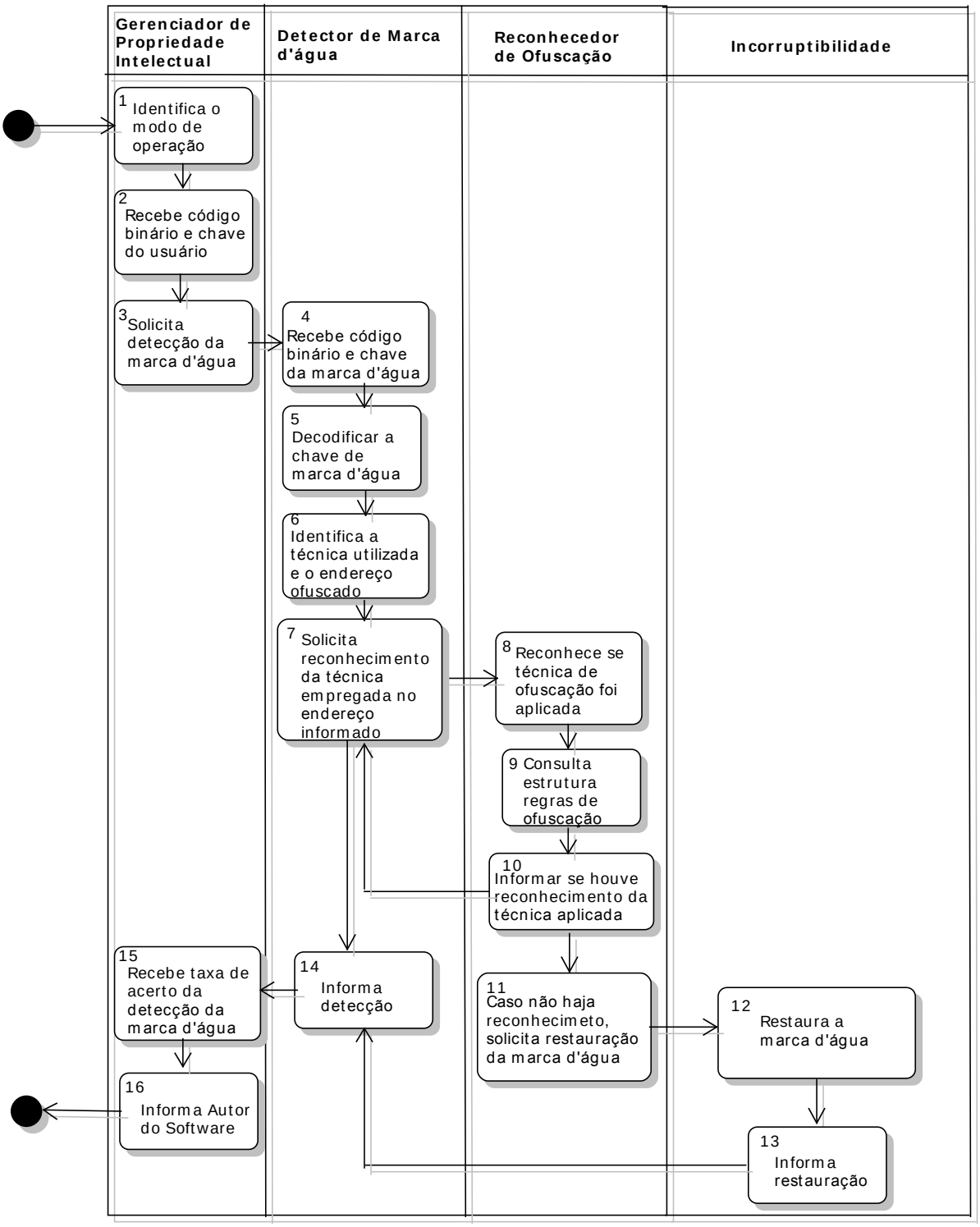

Figura 13**:** Diagrama da operação de detecção de marca d'água

4.4 MODELO DE ATAQUE

Esta seção apresenta o modelo de ataque utilizado pelo ***SensorWatermark***. Esse modelo teórico apresenta os ataques que possibilitariam um atacante quebrar a proteção de

propriedade intelectual de um software e as formas de proteção que faria com que o software resistiria a esses ataques.

A qualidade da marca d'água está relacionada à sua resistência contra ataques (COLLBERG et al 2011). Inicialmente, uma vez que um atacante tem a posse do dispositivo onde o programa com marca d'água foi implantado, ele pode retirar seu código binário a partir do disco ou cartão de memória e usar à vontade ferramentas de análise estática (*disassemblers* e *decompiladores*) e ferramentas de análise dinâmica (depuradores e emuladores) para descobrir e destruir a marca d'água incorporada antes de tentar distribuí-lo . No nosso trabalho, as ferramentas de análise estática usada pelo atacante são do tipo *disassemblers* denominado *IDAPRO.*

Supondo que um atacante disposto a distribuir *Pw* (programa com marca d'água) sem ser rastreado, quer dizer, sem que o proprietário possa comprovar a autoria do software, ele deve concentrar seus esforços em localizar e atacar a marca d'água original. Os ataques à marca d'água original são feitos por meio das funções do esquema de aplicação de marca d'água. Várias ferramentas podem ser utilizadas, como por exemplo: o *Hydan* (EL-KHALIL e KEROMITZ, 2003) ou *Sandmark* (COLLBERG *et al.*, 2003).

Portanto, o atacante pode comprometer *Pw* realizando os seguintes ataques: (i) aditivo, (ii) subtrativo e (iii) distorcivo (COLLBERG e NAGRA, 2010). No ataque aditivo, o atacante tenta inserir uma nova marca d'água *w'* em *Pw*, a fim de confundir a função de detecção, criando dúvidas em afirmar qual é a marca d'água verdadeira, *w* ou *w'*. Por outro lado, no ataque subtrativo, o atacante tenta remover *w* de *Pw*, sem modificar o comportamento original. Por fim, no ataque de distorção, o atacante emprega transformações de código, como otimização e ofuscação, em *Pw*, a fim de destruir *w* sem saber sua localização exata e mudando o seu comportamento original. Assim, quando as funções de reconhecimento ou de extração tentam identificar *w* em *P'w*' (programa atacado com marca d'água), eles não serão capazes de reconhecer ou retornar *w* em *P'w'* já que *w* tornou-se *w'*.

No caso do atacante utilizar um ataque subtrativo neste software (COLLBERG e NAGRA, 2010), seria realizado uma descompilação no código executável para tentar eliminar a marca d'água. Se por meio do código descompilado, a marca d'água puder ser localizada, o software ficará vulnerável e poderá ser atacado, permitindo a remoção da marca d'água original. No trabalho proposto, esse tipo de ataque é tratado utilizando ofuscações de código para a inserção da marca d'água, dificultando a localização da mesma. A marca d'água é transformada em uma sequência de ofuscações que são realizadas no código do programa.

Se o atacante utilizar um ataque aditivo (COLLBERG e NAGRA, 2010), tentando adicionar uma outra marca d'água em um software com uma marca d'água original, seria difícil provar qual a marca d'água é a original. Para tratar este problema, o trabalho proposto incorpora no código do programa com a marca d'água, a chave utilizada na função de inserção e extração do esquema de aplicação da marca d'água. Com isso, mesmo que um atacante consiga adicionar outra marca d'água no software, será possível comprovar qual é a marca d'água original, pois a chave contém a localização da mesma.

Finalmente, se o atacante utilizar um ataque distorcivo (COLLBERG e NAGRA, 2010) no software com a marca d'água, ataque este que realizam transformações no código do programa, seria possível destruir a marca d'água incorporada porque como o trabalho proposto já utiliza ofuscações para a inserção da marca d'água, poderia ser realizado outras ofuscações nas instruções que já teriam sido ofuscadas, destruindo assim a marca d'água original. Para tratar esse problema, o esquema proposto é combinado com a técnica de incorruptibilidade que detém a remoção da chave e verifica a integridade da mesma. Com isso, se o software for submetido a novas ofuscações no código e a chave tenha sido alterada, o programa irá detectar a alteração na marca d'água, realizando a restauração da mesma.

5 IMPLEMENTAÇÃO

Esse capítulo discute as principais questões envolvidas na implementação de um protótipo utilizado para avaliar o ***SensorWatermark***. Esse capítulo está organizado em três seções.

Na seção 5.1, é apresentado o ambiente de programação utilizado para desenvolver o protótipo. Na seção 5.2, é apresentado o componente Gerador de Chave do ***SensorWatermark***. Na seção 5.3, é apresentado o componente Ofuscador de ***SensorWatermark***. Finalmente, na seção 5.4, é apresentado o componente Incorruptibilidade do ***SensorWatermark***.

5.1 AMBIENTE DE IMPLEMENTAÇÃO DO PROTÓTIPO

A implementação do arcabouço ***SensorWatermark*** foi desenvolvida com a linguagem de programação Python 2.7.3 (PYTHON, 2014) no ambiente de desenvolvimento integrado denominado PyCharm. Python é uma linguagem de programação orientada a objetos e interpretada, com uma sintaxe muito simples, intuitiva e de fácil compreensão. A plataforma de hardware do sensor que utilizamos é o MICAz (CROSSBOW,2010). Essa plataforma utiliza um microprocessador de 8 bits (ATmega128) fabricado pela Atmel (ATMEL128,2014). Tal microprocessador possui poder de processamento de 4 MHz, memória programável de 128 *Kbytes*, 512 *Kbytes* de memória *flash*, um rádio de 2400 MHz, com uma taxa de 250 Kbps e alcance de 20 a 30 metros, alimentados por 2 baterias do tipo AA. As aplicações desenvolvidas para estes sensores foram escritas para o sistema operacional TinyOS, versão 2.1.2 usando a linguagem NesC (LEVIS e GAY, 2009), uma extensão da linguagem C, que implementa um modelo de programação orientado a eventos.

5.2 GERADOR DE CHAVE

Neste componente é implementado o processo de criação da chave de marca d'água, apresentado na Figura 14. O processo consiste de quatro passos: (i) transformação de nome de autor de software em código ASCII, (ii) Seleção de técnicas de ofuscação e endereços no código (iii) popular a estrutura Chave de Marca d'água e (iv) incorporar a chave de marca d'água no software. É apresentado um exemplo de criação de chave para um dado Autor de Software, na Figura 15.

O primeiro passo, denominado ***Transformação de nome de autor de software em código ASCII,*** consiste em transformar a cadeia de caracteres que identifica o autor de um software (*Autor do Software*) em um código binário (*BinStr*). O *BinStr* também indica o número de técnicas de ofuscações que devem ser empregadas a fim de inserir uma marca d'água em um determinado software. Para produzir o *BinStr*, cada caracter do código ASCII da sequência do Autor do Software é concatenada.

O segundo passo, denominado ***Seleção de técnicas de ofuscação e endereços no código***, consiste em selecionar as técnicas de ofuscação que devem ser aplicadas em determinadas instruções de um dado software. Os elementos de uma sequência de bits (*BinStr*) são numerados de zero até o número de bits da sequência menos um, na ordem da direita para a esquerda, (o bit mais à direita é numerado como zero). O Gerador de Chave lê *BinStr* a partir do seu bit mais à direita e seleciona para cada dígito binário uma técnica de ofuscação. Para selecionar uma técnica de ofuscação, as seguintes regras são obedecidas: (i) se o dígito de *BinStr* for igual a 0, selecione aleatoriamente uma técnica de ofuscação na estrutura **Regras de Ofuscação**, dentre as técnicas cujo o índice de busca na estrutura seja zero ou par; (ii) caso contrário, selecione aleatoriamente uma técnica de ofuscação na estrutura Regras de ofuscação, entre as técnicas que o índice de busca na estrutura são impares. Este mesmo processo é aplicado para selecionar as técnicas de ofuscação para todos os outros dígitos binários de *BinStr*.

O terceiro passo, denominado ***Preenchimento da estrutura Chave de Marca d'água***, consiste em armazenar na estrutura Chave de Marca d'água a informação (as técnicas de ofuscação e a localização das instruções onde cada uma dessas técnicas deve ser empregada) que foi obtida no passo anterior. O tamanho dessa estrutura deve ser igual ao número de dígitos em *BinStr*. Ressalta-se que se não houver instruções para que as técnicas sejam empregadas, será necessário a inclusão das mesmas no código por meio de instruções sem operação.

O quarto passo, denominado ***Incorporação da Chave de Marca d'água no software***, consiste em incorporar a chave de marca d'água dentro do software, a fim de o autor do software provar a autoria legitima do software. A estrutura Chave de Marca d'água deve ser incorporada a partir de uma sequência de predicados opacos (COLLBERG e NAGRA, 2010). Os predicados opacos são trechos de código que não afetam a execução do programa. Os predicados opacos são informados no processo de inicialização do ***SensorWatermark***.

É importante notar também que, como o Gerador de Chave escolhe aleatoriamente uma técnica de ofuscação, é possível criar um conjunto de chaves distintas para o mesmo

Autor do Software. Esse fato é positivo porque faz o autor do software ser capaz de usar chaves diferentes para várias cópias do mesmo software, tornando ainda mais difícil o atacante descobrir quem é o autor do software.

KeyGenerator(A)

Entrada: Autor (A),Programa (P), RegrasOfuscacao <indice,tecnica,equivalencia>

Saída: Chave de Marca d'água (K<digito, técnica,endereço>)

1. Cinst<inst,pos> ←Instruções em P que podem ser ofuscadas e suas posições

//transforma a string A em uma representação binária conforme a tabela ASCII

2. At←representação binária de A criada com base na tabela ASCII

3. Para cada indice j em At

4. idRegra←seleciona técnica de ofuscação com base no At[j]

5. p←escolherAleatoriamenteInstrucao(Cinst,RegrasOfuscacao[idRegra].tecnica)

6. endereco ←pegarEndereçoInstrucao($Cinst.pos_p$)

7. insereNoVetor(Kj, Atj, RegraOfuscacao[idRegra].tecnica,endereco)

8. Fim Para

Figura 14: Algoritmo do processo de criação da chave de marca d'água

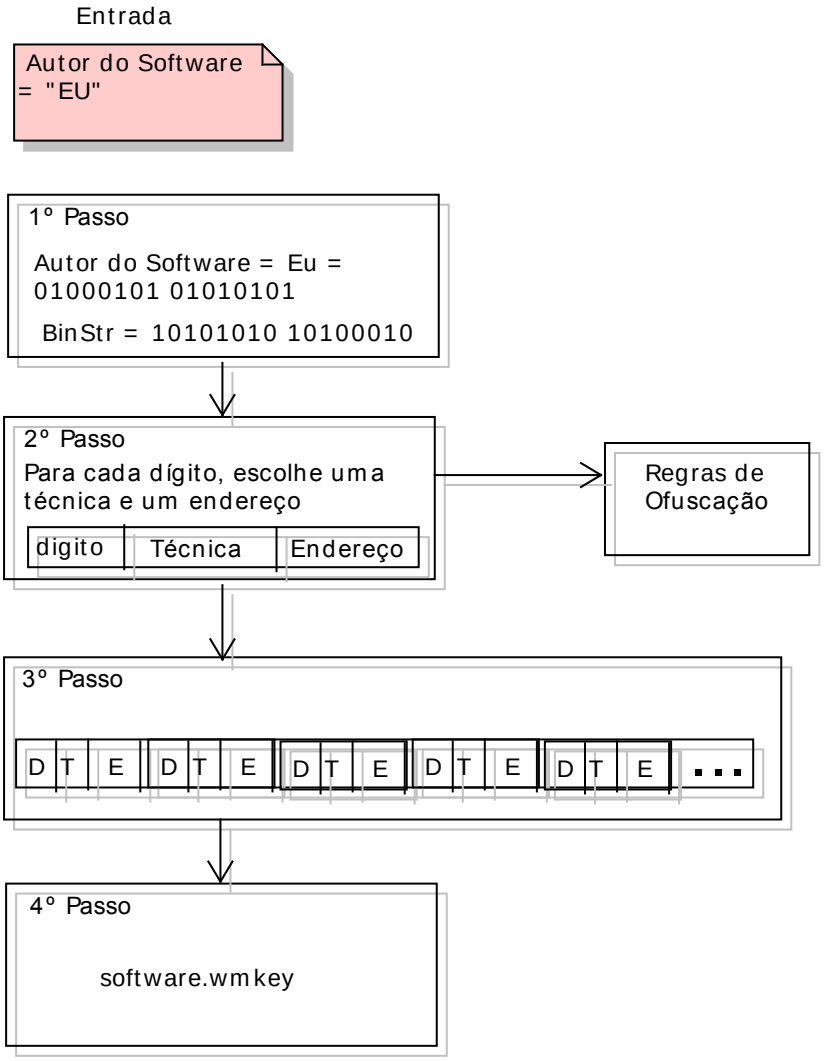

Figura 15: Criação de Chave

5.3 OFUSCADOR

O Ofuscador depende das informações armazenadas na estrutura de dados denominada de Regras de Ofuscação para aplicar as técnicas de ofuscação nos trechos de código localizados nos endereços especificados desse software. Em tempo de projeto, o especialista em segurança preenche a estrutura Regras de Ofuscação usado pelo componente Ofuscador, de acordo com as técnicas de ofuscação escolhidos para a plataforma de hardware selecionado. Portanto, para a Plataforma de Hardware selecionada neste trabalho (AVR), foi necessário preencher a estrutura de dados Regras de Ofuscação de acordo com as técnicas de ofuscação escolhidas: ofuscação de chamada (CallObf) (COSTA *et al.,*2010), ofuscação de retorno (RetObf) (COSTA *et al.,*2010) e Ofuscação de Salto Incondicional (JmpObf) (COSTA *et*

*al.,*2010). A instrução original CALL (call), RET (return) e JMP (salto incondicional) foram substituídos pelo conjunto de instruções {ldi / push / ldi / push / ldi / push / ldi / push / ret}; {pop / pop / jmp} e {ldi / ldi / push / push / call} respectivamente.

A Tabela 3 apresenta a estrutura de dados Regras de Ofuscação contendo as seguintes informações: plataforma de hardware adotada, Instrução original e o conjunto de instruções que devem ser utilizadas para a substituição da instrução original (Instruções ofuscadas).

Índice	**Técnicas de Ofuscação**	**Plataforma de Hardware**	**Instrução Original**	**Instrução Ofuscada**
0	Ofuscação de Chamada	AVR	CALL	ldi/push/ldi/push/ ldi/push/ldi/push/ret
1	Ofuscação de Retorno	AVR	RET	pop/pop/jmp
2	Ofuscação de Salto Incondicional	AVR	JMP	ldi/ldi/push/push/call

Tabela 3 - Regras de Ofuscação

O componente Ofuscador recebe a solicitação para ofuscar algumas instruções. Essa solicitação é enviada pelo componente Insersor de Marca d'água através da interface *reqObf*. Dada uma solicitação, o Insersor de Marca d'água envia um vetor contendo os endereços das instruções a serem ofuscadas e as técnicas de ofuscação que devem ser empregadas. As técnicas de ofuscação aplicadas nessas instruções permitem a criação do código fonte marcado.

A coluna da esquerda desta tabela (Código Original) apresenta um trecho de código original que contém as funções Main e Sum. Na função Main, as constantes 0xA e 0xB são empilhados e, em seguida, a chamada à função Sum é realizada, o que por sua vez, realiza a soma das constantes 0xA e 0xB. A coluna da direita desta tabela (Código ofuscado) mostra como as instruções responsáveis pela chamada (call) e pelo retorno (RET) da função foram ofuscados, de acordo com as informações da Tabela 4.

Código Original	**Código Ofuscado**
Main: L1: ldi r16, 0xA L2: push r16 L3: ldi r16, 0xB L4: push r16 **L5: call Sum** **L6: ret** Sum: L7: ldd r16, Y+3 L8: ldd r17, Y+4 L9: add r17, r16 L10: ret	Main: M1: ldi r16, 0xA M2: push r16 M3: ldi r16, 0xB M4: push r16 M5: **ldi r30, low(M14)** M6: **push r30** M7: **ldi r31, high(M14)** M8: **push r31** M9: **ldi r30, low(M17)** M10: **push r30** M13: **ret** M14: **pop r31** M15: **pop r30** M16: **ijmp** Sum: M17: ldd r16, Y+3 M18: ldd r17, Y+4 M19: add r17, r16 M20: ret

Tabela 4: Exemplos de como o SensorWatermark aplica as técnicas de ofuscação

Como exemplo, podemos considerar o autor do software como “UFRJ” e o *BinStr* contendo 32 dígitos (01010101010001100101001001001010). Pega-se o primeiro caracter representado pela letra “U”, representado pelos oito primeiros dígitos binário da sequência e exemplificamos o emprego de técnicas de ofuscação nos dois primeiros dígitos, que é representado pelo digito 0 e pelo digito 1, respectivamente. Foi selecionado um trecho de código do software original informado pela tabela 4, onde foi selecionado aleatoriamente as instruções CALL e RET que se encontram localizadas nos endereços L5 e L6, respectivamente. A instrução escolhida poderia ter sido retirada de qualquer trecho do código. Os dois primeiros dígitos do caractere “U” são 0 e 1, devido a isso, as técnicas de ofuscação que serão aplicadas nos endereços representados por esses dígitos serão selecionadas por meio da tabela 3 seguindo as regras estabelecidas pelo componente gerador de chave. Para o primeiro digito, onde a técnica selecionada na estrutura regras de ofuscação é a CallObf, primeiro a instrução de chamada (call), localizada no endereço L5, é substituída pelas instruções localizadas entre os endereços M5 e M13. As instruções ldi, push, ldi push localizadas entre M5 e M8 são responsáveis por empilhar o endereço de retorno da função a ser chamada (M14) e as instruções ldi, push, ldi push localizadas entre M9 e M12 são responsáveis por colocar no topo da pilha o endereço da função a ser chamada (M17). Por

último, o componente Ofuscador insere no endereço M13 a instrução ret que será responsável por redirecionar o fluxo de controle para o endereço localizado no topo da pilha, ou seja, M17 (endereço da função Sum). Após executar as instruções da função Sum, o fluxo de controle é então redirecionado para o endereço de retorno empilhado anteriormente (M14). Em seguida, pega-se o segundo digito da sequência que é representado pelo digito 1 e seleciona-se a técnica de ofuscação que será aplicada neste endereço, neste caso a técnica selecionada na estrutura regras de ofuscação é a RetObf, para isso a instrução de retorno (ret) localizada no endereço L6 é substituída pelas instruções pop, pop, ijmp. As instruções pop são responsáveis por armazenar nos registradores especificados (r30 e r31) o endereço de retorno da função, que está localizado no topo da pilha, e a instrução ijmp é responsável por redirecionar o fluxo de controle para o endereço especificado pelos registradores r31 e r31.

5.4 INCORRUPTIBILIDADE

Neste componente, é implementado o processo de detecção de violação da integridade do código, isto é, a inserção das guardas no código baseado na auto-verificação através do uso de *hash*. O processo consiste de dois passos, no modo de operação " Inserção de Marca d'água": (i) Inserção das guardas no código e (ii) Inserção dos verificadores para as guardas inseridas. Além disso, o processo consiste de um passo, no modo de operação " Detecção de Marca d'água": (i) Acionamento da restauração da marca d'água.

Quando o modo de operação é igual a "Inserção da Marca d'água", o primeiro passo do processo de incorruptibilidade é denominado como ***Inserção das guardas no código,*** apresentado na Figura 16, que consiste em preencher um vetor com os cálculos *hash* dos trechos do código onde foram aplicadas técnicas de ofuscação para a inserção da marca d'água. O endereço inicial de cada trecho de código a ser verificado é identificado pela chave de marca d'água. O endereço final de cada trecho de código a ser verificado é estabelecido pelo termino da aplicação da técnica de ofuscação de código que foi aplicada. O *hash* é calculado e o resultado é comparado com o valor de *hash* esperado.

InsereGuarda&Verificador (K)

Entrada: Chave de Marca d'água (K)

Saída: Hashs dos trechos com Marca d'água(H)

1. // calcula o Hash dos blocos que compões a chave e popula a estrutura com os Hashs

2. Para cada i em K

3. h←CalculaHash (At, k.endereco$_i$, K.instrucao$_i$))

4. H←Concatena(hi)

5. Fim Para

Figura 16: Algoritmo de inserção de guardas

O segundo passo, denominado ***Inserção dos verificadores para as guardas inseridas***, consiste em inserir outras guardas para proteger as guardas que foram inseridas para a verificação da integridade do trecho de código onde está a marca d'água. Esse passo permite a proteção mútua entre as guardas inseridas no código do programa.

Quando o modo de operação é igual a "Detecção da Marca d'água", o passo é denominado ***Acionamento da restauração da marca d'água***, apresentado na Figura 17, em que consiste em restaurar a marca d'água do software, substituindo o trecho de código adulterado pelo trecho de código original. Nesse passo, é recalculado o *hash* para cada trecho de código que foi empregada uma técnica de ofuscação para a inserção da marca d'água e comparado com o *hash* que foi calculado na inserção da marca d'água que encontra-se armazenado na estrutura de dados. No caso do resultado ser diferente, a marca d'água é restaurada por meio da chave de marca d'água.

RestauraMarca(P', K)

Entrada: Programa com marca (P'), Autor em Binario (At),Chave da Marca d'água (K<digito, técnica, endereço>), Hashes dos trechos com Marca d'água (H)

Saída: Programa com marca recuperada (P")

```
1. Para cada i in K
2.     endereco_final = 0;
3.     Bloco = null;
4.     Bloco_novo = null;
5.     endereço_inicial = k.endereçoi
2.     endereco_final=VerificaTecnica(k.endereçoi)
3.     Bloco = Trecho(endereco_inicial, endereco_final)
4.     Bloco_novo = Concatena(Bloco, At)
5.     H_novo = calcula_hash(Bloco_novo)
6.        Se Hi é diferente de H_novo
7.           P'' = restauramarcadagua(P', K)
8.        Fim_Se
9.  Fim_Para
```

Figura 17: Algoritmo do acionamento da restauração da marca d'água

6 EXPERIMENTOS E ANÁLISE DOS RESULTADOS

Nesse capítulo, são apresentados os experimentos que foram realizados para avaliar o ***SensorWatermark***. Os experimentos que foram realizados são: a análise da credibilidade, a análise da furtividade, a análise da resistência a ataques e o impacto do nó sensor da RSSF em termos de consumo de recurso após a inserção da marca d'água.

Este capítulo foi dividido em oito seções, a saber: Seção 6.1, que descreve o ambiente dos experimentos; Seção 6.2, que descreve as métricas usadas nos três conjuntos de experimentos; Seção 6.3, que descreve o cenário dos experimentos; Seção 6.4, experimentos para analisar a furtividade da marca d'água; Seção 6.5, experimentos para analisar a credibilidade da marca d'água; Seção 6.6, experimentos para analisar o impacto do nó sensor na RSSF em termos de consumo de recurso; Seção 6.7, experimentos para analisar a resistência a ataques que o software com a marca d'água pode sofrer.

6.1 AMBIENTE DOS EXPERIMENTOS

A descrição desse item foi subdividida em ambiente de simulação e ambiente com hardware de nós sensores reais. O primeiro trata de características intrínsecas do ambiente do simulador Avrora e o segundo trata das características do ambiente dos nós sensores reais. Em todos os experimentos realizados com o ***SensorWatermark***, tanto em ambiente simulado quanto em ambiente de hardware real, a RSSF foi composta por nós da plataforma de hardware denominada MICAz, da empresa MEMSIC (2013). O nó sensor do tipo MICAz foi escolhido devido a duas razões principais: a grande quantidade desses sensores adquirida em projetos de pesquisa e pela existência de diversos simuladores desenvolvidos por terceiros para esse tipo de sensor. A escolha da plataforma MICAz permitiu, portanto, a utilização da mesma plataforma de hardware tanto em ambiente real quanto em ambiente simulado.

Nos experimentos, foram utilizados os seguintes programas: 6lowpancli, Oscilloscope, RadioCountToLeds, RadioSenseToLeds, and BaseStation. Esses programas são distribuídos junto com o ambiente de desenvolvimento do TinyOS (LEVIS E GAY, 2009), versão 2.1.2, que usa a linguagem NesC (LEVIS E GAY, 2009). Os tamanhos das marcas d'águas incorporadas dentro dos programas são 1,3,6,8 e 10 bytes.

Os experimentos foram repetidos 30 vezes (DEGROT, 2002) e foi usado intervalo de confiança de 95% para os resultados. O *disassembler* escolhido para avaliar o arcabouço proposto foi o *IDA PRO*, versão 6.3, uma ferramenta comercial de disassemblers, que se

baseia em um algoritmo recursivo transversal. A linguagem de programação utilizada para desenvolver o ***SensorWatermark*** foi o Python 2.7.3.

6.1.1 Ambiente de Simulação

A Figura 18 apresenta uma ilustração esquemática do ambiente de simulação. Uma estação (computador *laptop* equipado com um processador Intel Core i7 2.10 GHz e 8 GB de RAM) foi usada para rodar as simulações. As simulações foram executadas com o simulador Avrora na versão 2.6 (TITZER 2005) que é um simulador de código aberto para RSSF.

Especificamente, a extensão AvroraZ (ALBEROLA e PESCH, 2008) foi empregada para analisar o consumo de energia e a comunicação para a plataforma MICAz. O modelo de energia usado pelo simulador Avrora é denominado "*Accurate Prediction of Power Consumption*" (AEON) (LANDSIEDEL et al., 2005) e é o modelo de energia que representa com maior precisão os ciclos de processamento dos sensores MICAz.

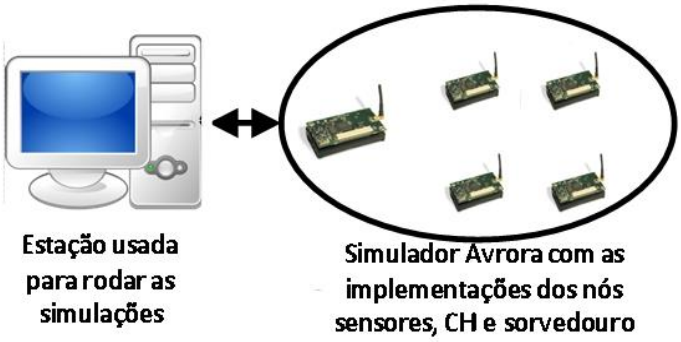

Figura 18: Ambiente de Simulação

6.1.2 Ambiente com nós sensores reais

A plataforma de hardware utilizada no ambiente com nós sensores reais foi também a plataforma MICAz, cuja descrição e razão de sua escolha já foram apresentadas no início da Seção 6.1. Os mesmos softwares utilizados pelo ***SensorWatermark*** nos experimentos no ambiente simulado, foram usados nos experimentos que fazem uso de nós sensores reais. Ou seja, os softwares com a marca d'água inserida pelo ***SensorWatermark*** foi instalado também nos nós sensores reais.

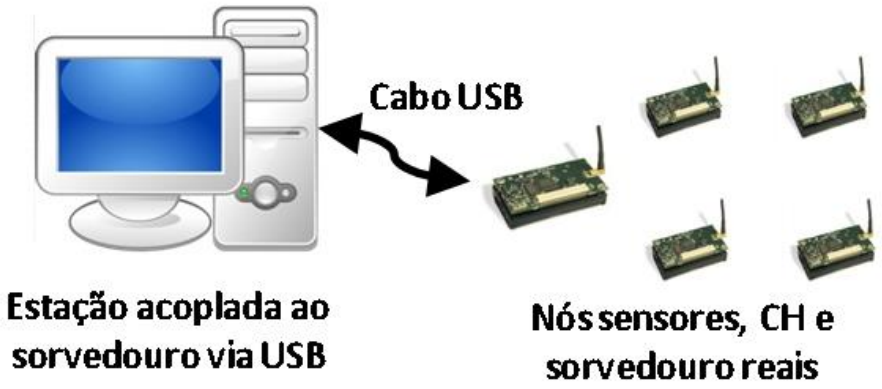

Figura 19: Ambiente com nós sensores reais

O ambiente com nós sensores reais foi montado em um ambiente controlado, dentro do Laboratório de Redes sem Fio (Labnet) do Programa de Pós-Graduação em Informática da UFRJ. O nó sorvedouro era composto de um nó MICAz acoplado via cabo USB a um computador *laptop* equipado com um processador Intel i7 2.10 GHz e 8 GB de RAM.

Apesar do modelo de energia do ambiente de simulação considerar todos os aspectos dos componentes do sensor, não é possível obter com facilidade os mesmos dados acerca do consumo de energia da rede de sensores reais, pois exigiria uma grande quantidade de trabalho aferir manualmente as medidas de cada um dos componentes de cada um dos nós sensores e em cada um dos cenários dos experimentos. Por esse motivo, e considerando que o modelo de energia das simulações foi suficientemente validado, as medições de consumo de energia pelos nós MICAz foram realizadas apenas através de simulações.

6.2 MÉTRICAS

As métricas utilizadas nestes experimentos foram divididas em quatro grupos: (i) métrica para avaliar a furtividade da marca d'água (ii) métricas para avaliar a credibilidade da marca d'água (iii) o impacto do esquema de aplicação de marca d'água proposto na RSSF em termos do consumo de recursos dos sensores e (iv) métrica para avaliar a resistência a ataques que o software com a marca d'água pode sofrer.

6.2.1 Métricas para avaliar a furtividade da marca d'água

A métrica utilizada para avaliar a furtividade foi a distância de Mahalanobis (LEE, 2010), que tem sido utilizada para a detecção de malware baseadas em estatísticas. A distância de Mahalanobis calcula a diferença entre dois vetores, compostos de variáveis aleatórias escalares do mesmo espaço de probabilidade, e informa a dissimilaridade entre eles. Neste trabalho, a distância de Mahalanobis é usado para medir a dissimilaridade entre o software com marca d'água e o software sem a marca de água que pode ser incorporado pelo ***SensorWatermark***.

$$md(\vec{N}, \vec{X}) = \sqrt{(\vec{N} - \vec{X})^T S^{-1} (\vec{N} - \vec{X})}$$

Equação 1: Distância de Mahalanobis

A equação 1 descreve como a distância de Mahalanobis deve ser calculado. Nesta expressão, cada elemento do Vetor N é uma função massa de probabilidade (pmf) dos opcodes encontrados no conjunto de software sem a marca d'água. Por outro lado, X é o vetor cujos componentes apresentam a ocorrência dos opcodes em um software com marca d'água. Finalmente, S é a matriz diagonal cujos elementos da diagonal principal são as variâncias do conjunto de programas utilizados para criar o vetor N. Portanto, quando a distância de Mahalanobis aumenta, mais anômalo é o software com a marca d'água comparado com o conjunto de software sem marca d'água. Tal anomalia pode indicar que um dado software está com marca d'água. Por outro lado, se a distância de Mahalanobis calculada entre um software com a marca d'água e o conjunto de software sem marca d'água for similar, isto significa que a marca d'água incorporada não dá evidências de que o software tem uma marca d'água incorporada e que tal marca d'água pode ser considerada furtiva.

6.2.2 Métricas para avaliar a credibilidade da marca d'água

As métricas utilizadas nos experimentos para avaliar a credibilidade do ***SensorWatermark*** em termos da sua capacidade de detectar uma marca d'água no software, foram: falsos positivos (FP), falsos negativos (FN), verdadeiros positivos (VP), verdadeiros negativos (VN). Os VP indicam que o ***SensorWatermark*** é capaz de detectar uma marca d'água incorporada em um software quando de fato a marca d'água está incorporada nesse software. Os FP indicam que o

SensorWatermark é capaz de detectar uma marca d'água incorporada em um dado software, mas que de fato, nenhuma marca d'água foi incorporada nesse software. Os VN indicam que o ***SensorWatermark*** não é capaz de detectar uma marca d'água incorporada em um dado software, quando de fato nenhuma marca d'água foi incorporada nesse software. Os FN indicam que o ***SensorWatermark*** informa que uma marca d'água não está incorporada em um dado software, mas de fato uma marca d'água foi incorporada nesse software. É importante que o ***SensorWatermark*** forneça uma baixa probabilidade de taxa de falsa detecção (FP + FN).

6.2.3 Métricas para avaliar o impacto da marca d'água

As métricas utilizadas nos experimentos para avaliar a sobrecarga do ***SensorWatermark*** em termos de consumo de recursos foram: Diferença de Ciclos de Processamento (DCP) e Diferença de Consumo de Memória (DCM). O DCP é definido como sendo a diferença (percentagem) entre os números de ciclos de processamento gastos para a execução do software com marca d'água e o mesmo software original e o DCM é definido como sendo a diferença (percentagem) entre a quantidade de memória consumida (em bytes) pelo software com marca d'água e o software original.

6.2.4 Métricas para avaliar a resistência a ataques

A métrica utilizada nos experimentos para avaliar a capacidade do software com uma marca d'água incorporada pelo ***SensorWatermark*** resistir a ataques de distorção é a taxa de sucesso. A taxa de sucesso é definida, neste caso, como a percentagem entre o número de tentativas falhas para distorcer a marca d'água do software em comparação com o número total de tentativas de ataques.

6.3 DESCRIÇÃO DO CENÁRIO DOS EXPERIMENTOS

Nos experimentos realizados com o ***SensorWatermark***, os seguintes parâmetros foram variados: (**i**) número de nós sensores e (**ii**) softwares utilizados nos experimentos.

6.4 EXPERIMENTOS PARA ANALISAR A FURTIVIDADE

Nesta seção, são apresentados os experimentos que foram realizados para verificar se as marcas d'água incorporadas pelo ***SensorWatermark*** em um dado software são furtivas.
Para calcular a distância de Mahalanobis (MD) foram utilizados trinta softwares distribuídos juntamente com o ambiente de desenvolvimento TinyOS. A Tabela 5 apresenta os valores da distância de Mahalanobis MD para softwares sem marca d'água inserida pelo ***SensorWatermark*** (nonsw), e softwares com marca d'água inserida pelo ***SensorWatermark*** (sw). A Tabela 5 apresenta também os valores da diferença (diferença md) obtidos entre os valores de MD para o software sem marca d'água (nonsw), e para o software com marca d'água (sw). Observa-se que na coluna (diferença md) da Tabela 5, em termos de percentagem, que as distâncias entre o software com a marca d'água, do mesmo software sem a marca d'água são muito pequenas. Portanto, podemos considerar que o software com a marca d'água inserida pelo ***SensorWatermark*** são semelhantes aos do software original. Assim, podemos considerar que a marca d'água incorporado pelo ***SensorWatermark*** é furtiva.

Software	nonsw (md)	sw (md)	(md) diferença
6lowpancli	46.5	48.9	0.05%
Oscilloscope	28.4	30.07	0.06%
RadioCountToLeds	26.01	28.34	0.09%
RadioSenseToLeds	26.01	28.34	0.09%
BaseStation	17.85	19.10	0.07%

Tabela 5: Resultados dos Experimentos de Furtividade

6.5 EXPERIMENTOS PARA ANALISAR A CREDIBILIDADE

Nesta seção, são descritos os dois conjuntos de experimentos realizados para avaliar a credibilidade do ***SensorWatermark*** em termos da sua capacidade para detectar se a marca d'água foi inserida em um dado software. Em ambos os conjuntos de experimentos, é avaliado se o ***SensorWatermark*** é capaz de detectar a marca d'água de cada um dos softwares

citados no início dessa seção. Em ambos os experimentos, para cada um desses software foi escolhido de forma aleatória o tamanho da marca d'água, de 1 a 10 bytes.

O primeiro conjunto de experimentos, o modo de operação do ***SensorWatermark*** foi definido como "inserção da marca d'água no software", e inseriu-se para cada software, marcas d'água que variavam seu tamanho de 1 a 10 bytes (1, 3, 6, 8, e 10 bytes). Em seguida, para cada software, definiu-se o modo de operação do ***SensorWatermark*** como "Detecção de Marca d'água", e foi dada uma chave de marca d'água do software original a fim de verificar se o ***SensorWatermark*** é capaz de reconhecer cada marca d'água originalmente inserida como válida ou não. Os resultados do primeiro conjunto de experimentos mostram que o ***SensorWatermark*** foi capaz de detectar corretamente todas as marcas d'água no respectivo software. Em todos os casos, os valores de FP + FN foram de 0%.

No segundo conjunto de experimentos, foi definido o modo de operação do ***SensorWatermark*** como " Inserção de uma marca d'água de software", e inseriu-se para cada software, marcas d'água originais que variavam de tamanho de 1 a 10 bytes (1, 3, 6, 8 e 10 bytes). Em seguida, o modo de operação do ***SensorWatermark*** foi definido como sendo "Detecção de marca d'água" e foi dada uma chave de marca d'água de software diferente do que foi utilizado para incorporar a marca d'água do software original, a fim de determinar se o ***SensorWatermark*** é capaz de reconhecer a marca d'água no respectivo software como válida (original) ou não. Os resultados mostram que para todos os softwares, o ***SensorWatermark*** informa que as marcas d'água não foram detectadas. Portanto, o ***SensorWatermark*** apresentou 0% de falsas detecções (FN). Estes resultados mostraram que a marca d'água incorporada pelo ***SensorWatermark*** é crível.

6.6 EXPERIMENTOS PARA ANALISAR O IMPACTO

Esta seção descreve os experimentos que foram realizados para avaliar a sobrecarga do ***SensorWatermark*** em termos de consumo de recursos. Para cada software foi realizado um experimento para avaliar a diferença de ciclos de processamento (DCP) e a diferença de consumo de memória (DCM) do mesmo software com marca d'água embutida em relação ao software original (sem marca d'água embutida). Para avaliar os valores de DCP e DCM do software com marca d'água em comparação com o respectivo software original foram realizados seis experimentos, um para cada software citado no início da seção.

DCM											
Software	**Tamanho original**	**Tamanho do software com marca d'água de**					**% de acréscimo**				
		1byte	3bytes	6bytes	8bytes	10bytes	1byte	3bytes	6bytes	8bytes	10bytes
6lowpancli	56.5	56.6	56.7	56.9	57.1	57.2	0.1	0.3	0.7	1.0	1.2
Oscilloscope	38.4	38.5	38.7	38.9	39.1	39.2	0.2	0.7	1.3	1.8	2.0
RadioCountToLeds	36.6	36.8	36.9	37.1	37.3	38.6	0.5	0.8	1.3	1.9	5.2
RadioSenseToLeds	37.8	37.9	38.1	38.3	38.5	38.7	0.2	0.7	0.7	1.8	2.3
BaseStation	43.0	43.1	43.3	43.5	43.6	43.7	0.2	0.6	0.6	1.3	1.6

Tabela 6: Diferença de consumo de memória

Para todos os experimentos, observa-se que os valores da DCP de todos os softwares com marca d'água em comparação com o respectivo software original eram menores que 1%, considerando-os como irrelevante. Logo, as operações realizadas pelo ***SensorWatermark*** para inserir uma marca d'água em um dado software não impactam negativamente em termos do consumo de processamento se comparada com o software original. Este fato ocorre, pois, o ***SensorWatermark*** adiciona apenas algumas instruções no software para inserir a marca d'água à base de ofuscação de código. Portanto, o ***SensorWatermark*** requer uma quantidade insignificante de ciclos de processamento para executar o software com marca d'água se comparado com o software original.

Para todos os experimentos, observa-se que os valores do DCM para todos os softwares com marca d'água comparados com o respectivo software original têm aumentos insignificantes conforme o tamanho da marca d'água aumenta nesses softwares. Os valores médios de DCM para os softwares, 6lowpancli, Oscilloscope, RadioCountToLeds, RadioSenseToLeds e BaseStation, são respectivamente 0,6%, 1,2%, 1,9%, 1,1%, e 0,8%. Portanto, conclui-se que a sobrecarga gerada pelo ***SensorWatermark*** em termos de quantidade de tamanho de memória e de ciclos de processamento para inserir uma marca d'água em um dado software pode ser considerada irrelevante se comparada ao software original.

6.7 EXPERIMENTOS PARA ANALISAR A RESISTÊNCIA A ATAQUES

Foram realizados experimentos de forma a verificar a capacidade da marca d'água aplicada pelo ***SensorWatermark*** de resistir a ataques distorcivos.

6.7.1 Resiliência contra ataques Distorcivos

Nesta Seção são apresentados os experimentos que foram realizados para verificar se as marcas d'água inseridas nos respectivos softwares mencionados no início do capítulo pelo SensorWatermark são robustas, ou seja, que as marcas d'água resistem aos ataques distorcivos.

Ataques de distorção envolve a aplicação de transformações no código, tais como ofuscações, eliminando assim qualquer marca d'água que dependa do código do software. Nesses experimentos (para cada software mencionado no início do capítulo), os ataques distorcivos foram feitas por uma ferramenta denominada TinyObf (o ofuscador apresentado em (COSTA *et al*, 2012). Nesses experimentos, os passos seguintes foram realizados para cada software: (i) o SensorWatermark insere a marca de água em um dado software (ii) a ferramenta apresentada em (COSTA *et al*, 2012) aplica técnicas de ofuscação em um dado software usando as técnicas CallObf, RetObf e JmpObf e (iii) o SensorWatermark tenta detectar a marca d'água após os ataques distorcivos.

Os primeiros resultados destes experimentos mostraram que as marcas d'água inseridas nos softwares pelo SensorWatermark eram resistentes a esses ataques porque o SensorWatermark usa técnicas de incorruptibilidade, fazendo a marca d'água resistente à alterações. Normalmente, quando transformações, como técnicas de ofuscação são aplicadas no software com marca d'água, a mesma pode ser destruída. Assim, quando os trechos de códigos ofuscados, que representam uma marca de água, são modificados, as guardas inseridas antes de cada trecho de código ofuscado identifica a alteração e restaura a marca d'água, usando as informações armazenadas na chave de marca d'água. No caso da marca de água ser alterada, a mesma é restaurada pelo SensorWatermark na operação de detecção. Para todos os experimentos, (para todo o software) a taxa de sucesso de resistir aos ataques distorcivos foram de 100%, uma vez que em todos os experimentos o SensorWatermark foi capaz de detectar a marca de água válida ou restaurar a marca d'água alterada.

7 CONCLUSÃO

O presente trabalho propôs um arcabouço de aplicação de marca d'água de software utilizando técnicas de ofuscação de código e de incorruptibilidade para Rede de Sensores Sem Fio denominado SensorWatermark, a fim de provar a autoria do software embarcado em nós sensores de uma RSSF. A marca d'água inserida pelo SensorWatermark é considerada furtiva, resistente à violação e eficiente. Essa proposta insere uma marca d'água que se baseia no uso de técnicas de ofuscação capazes de comprometer as ferramentas de engenharia reversa com o intuito de dificultar o comprometimento de nós.

A arquitetura proposta é genérica e pode ser personalizada para qualquer plataforma de hardware desde que as técnicas de ofuscação utilizadas para a inserção da marca d'água sejam personalizadas adequadamente para cada arquitetura de processadores. Em nosso estudo de caso, o SensorWatermark, foi personalizado para uma plataforma de hardware utilizada em RSSFs (MICAz) e a partir de nossos resultados esse arcabouço pode ser considerado uma solução de proteção de propriedade intelectual adequada para RSSFs para comprovar a autoria do software embarcado nos nós sensores.

As técnicas de ofuscação empregadas nesse arcabouço para inserção da marca d'água são baseadas na substituição e inserção de instruções. A utilização de tais técnicas não causa sobrecarga negativa nos sensores de uma RSSF porque substituem ou adicionam poucas instruções durante a ofuscação do programa e com isso o consumo de recursos do sensor não aumenta de forma considerável. Apesar disso, a combinação dessas técnicas é capaz de dificultar a localização da marca d'água no software.

A resistência a ataques foi comprovada a partir dos resultados obtidos dos programas submetidos à ataques de distorção. Na seção 6.7 mostra que as marcas d'água inseridas pelo SensorWatermark são resistentes a esses ataques, mesmo que a marca d'água seja alterada ou removida, as guardas inseridas antes dos trechos de código identificam a alteração e restauram a marca d'água. No caso de a marca de água ser alterada, ela será restaurada pelo SensorWatermark na operação de detecção, com a chave de marca d'água armazenada.

Outro benefício desse arcabouço é que a marca d'água inserida pelo SensorWatermark é considerada furtiva. As técnicas de ofuscação utilizadas para a aplicação da marca d'água colaboram para a inserção de uma marca d'água furtiva porque não dão indícios de que um programa está com uma marca d'água inserida como mostrado na seção 6.4. Os resultados mostram que um programa com uma marca d'água inserida pelo SensorWatermark

assemelha-se com o programa original sem marca d'água em relação à ocorrência de instruções. Tal fato é um benefício porque dificulta a localização da marca d'água.

As principais contribuições desse trabalho foram: em **primeiro** lugar, a descrição de uma arquitetura lógica de um arcabouço de aplicação de marca d'água de software utilizando técnicas de ofuscação de código e técnicas de incorruptiblidade capaz de inserir uma marca d'água furtiva, resistente à violação para RSSFs que pode ser generalizada para qualquer plataforma de hardware; em **segundo** lugar, um arcabouço que ao detectar que uma marca d'água foi alterada é capaz de recuperar a marca d'água original do software para comprovar a autoria do software e a marca d'água inserida pelo SensorWatermark é difícil de ser localizada, uma vez que a marca d'água de software está atrelada à ofuscações que tornam o software menos inteligível;

A partir dessas contribuições, dois artigos foram aceitos. O seguinte artigo foi publicado em, ICWN - International Conference on Wireless Networks – 2014 (Qualis B2), intitulado como: *TinyWatermark: a code obfuscation-based software watermarking framework for wireless sensor networks* (JORGE, *et al*. 2014), e o outro será apresentado no IWCMC - International Wireless Communications & Mobile Computing Conference – 2015 (Qualis B1), intitulado como: *SensorWatermark: a scheme of software watermark using code obfuscation and tamper-proofing for WSN* (JORGE, *et al*. 2015).

7.1 TRABALHOS FUTUROS

Como propostas de trabalhos futuros ressaltam-se os seguintes desdobramentos.
Primeiramente, sugere-se realizar a implementação do arcabouço com a aplicação de marca d'água dinâmica, a fim de proteger ainda mais a propriedade intelectual de um software embarcado em nós sensores de uma RSSF.

Em segundo lugar, sugere-se realizar a implementação do arcabouço em outras plataformas de RSSF, como a plataforma Contiki e RIoT, já que esses sensores utilizam outras arquiteturas. Novos experimentos podem ser realizados com essa nova implementação para comparar os resultados obtidos por elas com os resultados obtidos no presente trabalho, com a plataforma MICAz.

Por último, em terceiro lugar, sugere-se a implementação dos trabalhos relacionados que combinam as técnicas de aplicação de marca d'água com técnicas de ofuscação de código e incorruptibilidade. Novos experimentos podem ser realizados com a implementação dos

trabalhos relacionados, permitindo a comparação dos resultados com os resultados obtidos no presente trabalho.

REFERÊNCIAS

ALBEROLA, R. de P., PESCH, D. AvroraZ: extending avrora with an ieee 802.15.4 compliant radio chip model. In: WORKSHOP ON PERFORMANCE MONITORING AND MEASUREMENT OF HETEROGENEOUS WIRELESS AND WIRED NETWORKS, 3., 2008, Vancouver. **Proceedings**... New York: ACM, 2008, p. 43-50.

ALITAVOLI, Mohammad; JOAFSHANI, Mahdi; ERFANIAN, Aida. A novel watermarking method for Java programs. In: SYMPOSIUM ON APPLIED COMPUTING, 28., 2013, Coimbra. **Proceedings**... New York: ACM, 2013, p. 1013-1018.

ARORA, A. et al. A line in the sand: a wireless sensor network for target detection, classification, and tracking. **Computer Networks**, v. 46, n. 5, p. 605-634, dez. 2004.

ATmega128. Disponível em: <http://www.atmel.com/dyn/products. Acesso em: 01/2014.

BALAKRISHNAN, Arini; SCHULZE, Chloe. Code obfuscation literature survey. **CS701 Construction of compilers**, dez. 2005.

BOCCARDO, D. R.; NASCIMENTO, T. M. do; MACHADO, R. C. S.; PRADO, C. B. do; MORAES, F. P. de; CARMO, L. F. R. da C. Proteção, rastreabilidade e verificação de integridade de dispositivos com software embarcado. In: CONGRESSO INTERNO DO INMETRO, 1., 2010, Rio de Janeiro. **Anais**... Rio de Janeiro: INMETRO, 2010, [2] p.

BOCCARDO, D. R.; MACHADO, R. C. S.; CARMO, L. F. R. da C. Transformações de código para proteção de software. In: SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS, 10., 2010, Fortaleza. **Anais dos minicursos**... 2010, p. 103-148.

CHEN, Liang; ZHANG, Chaoquan. A novel algorithm for. NET programs watermarking based on obfuscation. In: INSTRUMENTATION & MEASUREMENT, SENSOR NETWORK AND AUTOMATION, 2012, Sanya. **Proceedings**… IEEE, 2012, p. 583-586.

CHEN, Min-Xiou; HU, Che-Chen; WENG, Wen-Yen. Dynamic object tracking tree in wireless sensor network. **EURASIP Journal on Wireless Communications and Networking**, v. 2010, p. 1-8, jun. 2010.

COLLBERG, Christian et al. Toward digital asset protection. **IEEE Intelligent Systems**, v. 26, n. 6, p. 8-13, 2011.

COLLBERG, Christian; NAGRA, Jasvir. **Surreptitious software**: obfuscation, watermarking, and tamperproofing for software protection. Upper Saddle River: Addison-Wesley, 2010.

COLLBERG, Christian; MYLES, Ginger; HUNTWORK, Andrew. Sandmark: a tool for software protection research. **IEEE Security & Privacy**, v. 1, n. 4, p. 40-49, 2003.

COLLBERG, Christian; THOMBORSON, Clark. Software watermarking: models and dynamic embeddings. In: PRINCIPLES OF PROGRAMMING LANGUAGES (POPL), 26., 1999, Texas. **Proceedings**... 2009, p. 311-324.

COSTA, Rafael de Oliveira. **TinyObf**: um arcabouço de ofuscação de código e proteção de dados para rede de sensores sem fio. 2012. 68 f. Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2012.

COSTA, Rafael de Oliveira et al. TinyObf: code obfuscation framework for wireless sensor networks. In: INTERNATIONAL CONFERENCE ON WIRELESS NETWORKS, 11., 2012, Las Vegas. **Proceedings**… 2012, p. 68-74.

CROSSBOW, "MICAZ Datasheet,"2010. CrossBow, Obtained through Internet: Disponível em:
<http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf>.
Acesso em: 4 abr. 2014.

DEGROOT, Morris H.; SCHERVISH, Mark J. **Probability and statistics**. 3. ed. Boston: Addison-Wesley, 2002.

DELICATO, Flávia Coimbra et al. A service approach for architecting application independent wireless sensor networks. **Cluster Computing**, v. 8, n. 2-3, p. 211-221, 2005.

EBERHARDT, Gergely et al. Copy protection through software watermarking and obfuscation. In: SALLAI, Gyula; DETREKÔI, Ákos. , v. 2, 2007.
Disponível em: http://www.hiradastechnika.hu/data/upload/file/2007/HT0701a.pdf. Acesso em: 14 set. 2015.

EL-KHALIL, R.; KEROMYTIS, A. D. Hydan: hiding information in program binaries. In: LOPEZ, J.; Qing, S.; Okamoto, E. (Ed.). **Information and Communications Security**. Berlin: Springer, 2004, p. 187-199. (Lecture Notes in Computer Science, 3269).

GAO, R.X.; FAN, Z. Architectural design of a sensory-node-controller for optimized energy utilization in sensor networks. **Instrumentation and Measurement**, v. 55, n. 2, p. 415-428, abr. 2006.

JEON, Cheol; CHO, Yookun. A robust steganography-based software watermarking. In: RESEARCH IN APPLIED COMPUTATION SYMPOSIUM, 2012, Texas. **Proceedings**… New York: ACM, 2012, p. 333-337.

JORGE, Emanuele N. de L. F. et al. SensorWatermark: a scheme of software watermark using code obfuscation and tamper-proofing for WSN. In: INTERNATIONAL WIRELESS

COMMUNICATIONS & MOBILE COMPUTING CONFERENCE, 11., 2015, Dubrovnik. **Proceedings**… 2015, p. 916-922.

JORGE, Emanuele N. de L. F. et al. TinyWatermark: a code obfuscation-based software watermarking framework for wireless sensor networks. In: INTERNATIONAL CONFERENCE ON WIRELESS NETWORKS, 13., 2014, Las Vegas. **Proceedings**… 2014, p.75-81.

KARLOF, C.; WAGNER, D. Secure routing in wireless sensor networks: attacks and countermeasures. In: SENSOR NETWORK PROTOCOLS AND APPLICATIONS, 1., 2003, Anchorage. **Proceedings**… IEEE, 2003, p. 113-127.

KHIYAL, Malik Sikandar Hayat et al. Evaluating effectiveness of tamper-proofing on dynamic graph software watermarks. **International Journal of Computer Science and Information Security**, v. 6, n. 3, p. 57-63, 2010.

KITCHENHAM, Barbara et al. Systematic literature reviews in software engineering: a systematic literature review. **Information and Software Technology**, v. 51, p. 7-15, 2009.

LAKHOTIA, A.; BOCCARDO, D. R.; SINGH, A.; MANACERO JR, A. Context-sensitive analysis without calling-context. **Journal of Higher-Order and Symbolic Computation**, v. 23, n. 3, p. 275-313, 2010.

LANDSIEDEL, O.; WEHRLE, K.; GÖTZ, S. Accurate prediction of power consumption in sensor networks. In: EMBEDDED NETWORKED SENSORS, 2., 2005, Sydney. **Proceedings**… IEEE, 2005, p. 37-44.

LEVIS, Philip; GAY, David. **TinyOS programming**. Cambridge: Cambridge University Press, 2009.

LINN, C.; DEBRAY, S. Obfuscation of executable code to improve resistance to static disassembly. In: COMPUTER AND COMMUNICATION SECURITY, 10., 2003, Washington. **Proceedings**… New York: ACM, 2003, p. 290-299.

LIU, T. et al. Implementing software on resource-constrained mobile sensors: experiences with impala and ZebraNet. In: MOBILE SYSTEMS, APPLICATIONS AND SERVICES, 2., 2004, Boston. **Proceedings**… New York: ACM, 2004, p. 256-269.

MA, LIANHONG et al. Instruction-words based software birthmark. In: MULTIMEDIA INFORMATION NETWORKING AND SECURITY, 4., 2012, Nanjing. **Proceedings**… IEEE, 2012, p. 909-912.

MAHALAXMI, D. S.; RAJU, S. V.; BABU, A. V. Attack model of version based software watermark. **International Journal of Engineering Inventions**, v. 1, n. 7, p. 20-21, out. 2012.

MAINWARING, A. et al. Wireless sensor networks for habitat monitoring. In: INTERNATIONAL WORKSHOP ON WIRELESS SENSOR NETWORKS AND APPLICATIONS, 1, 2002, Atlanta. **Proceedings**, New York: ACM, 2002, p. 88-97.

MARGI, C. et al. Segurança em Redes de Sensores Sem Fio In: SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS, 9., 2009, Campinas. **Anais dos minicursos**... Campinas: Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2009, p. 149-194.

MOHER, David et al. Preferred reporting items for systematic reviews and meta-analyses: the PRISMA statement. **Annals of Internal Medicine**, v. 151, n. 4, p. 264-269, 2009.

NAGRA, Jasvir; THOMBORSON, Clark; COLLBERG, Christian. A functional taxonomy for software watermarking. In: AUSTRALIAN COMPUTER SCIENCE CONFERENCE, 25., 2002, Melbourne. **Proceedings**… 2002, p. 177-186.

OSTERGARD JR, Robert L. The measurement of intellectual property rights protection. **Journal of International Business Studies**, v. 31, n. 2, p. 349-360, jun. 2000.

PALSBERG, Jens et al. Experience with software watermarking. In: COMPUTER SECURITY APPLICATIONS, 16., New Orleans, 2000. **Proceedings**… 2000, p. 308-316.

PERRIG, A.; STANKOVIC, J.; WAGNER, D. Security in wireless sensor networks. **Communications of the ACM**, v. 47, n. 6, p. 53-57, jun. 2004.

PYTHON TUTORIAL. Disponível em: <http://www.python.org/>. Acesso em: 13 maio 2014.

SALMON, Hélio Mendes et al. Sistema de detecção de intrusão imuno-inspirado customizado para redes de sensores sem fio. In: SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS, 10., 2010, Fortaleza. **Anais**… Fortaleza: Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2010, p. 269-282.

SARANYA, R.; ARTHY, R. Zero watermarking using sectional obfuscation scheme and stealthy code obfuscation technique, **International Journal of Advanced Research in Computer and Communication Engineering**, v. 2, n. 12, p. 4512-4516, dez. 2013.

SHADOW MARKET: 2011 global software piracy study. 9. ed. Bangkok: Business Software Alliance, maio. 2012. Disponível em: <http://globalstudy.bsa.org/2011/downloads/study_pdf/2011_BSA_Piracy_Study-Standard.pdf>. Acesso em: 13 set. 2015.

SHI, E.; PERRIG, A. Designing secure sensor networks. **IEEE Wireless Communications**, v. 11, n. 6, p. 38-43, 2004.

SHNAYDER, V. et al. **Sensor networks for medical care**. Cambridge: Division of Engineering and Applied Sciences, Harvard University, 2005. 14 p. (Technical Report TR-08-05).

SILVA, Ana Paula Ribeiro da. **Detecção de intrusos descentralizada em redes de sensores sem fio**. 2005. 81 f. Dissertação (Mestrado) – Universidade Federal de Minas Gerais, Belo Horizonte, 2005.

SU, X. et al. Secure routing in ad hoc and sensor networks. Wireless Network Security, Springer US, Part V, 2007, p. 381-402.

TITZER, B. L.; LEE, D. K.; PALSBERG, J. Avrora: scalable sensor network simulation with precise timing. In: INFORMATION PROCESSING IN SENSOR NETWORKS, 4., 2005, Los Angeles. **Proceedings**… 2005, p. 477-482.

VIANNA, Nilson Rocha. **EWIDS**: uma extensão para arquiteturas de sistemas de detecção de intrusos para redes sem fio metropolitanas. 2006. 163 f. Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2006.

VIEGA, J.; MESSIER, M. **Secure programming cookbook for C and C++**. Cambridge: O'Reilly, 2003.

WANG, YONG et al. CHI based instruction-words based software birthmark selection. In: MULTIMEDIA INFORMATION NETWORKING AND SECURITY, 4., 2012, Nanjing. **Proceedings**… 2012, p. 892-895.

WERNER-ALLEN, G. et al. Fidelity and yield in a volcano monitoring sensor network. In: OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 7., 2006, Seattle. **Proceedings**… 2006, p. 381-396.

XU, Guangxing; XIANG, Guangli. A method of software watermarking. In: INTERNATIONAL CONFERENCE ON SYSTEMS AND INFORMATICS, 2012, Yantai. **Proceedings**… 2012, p. 1791-1795.

XU, N. et al. A wireless sensor network for structural monitoring. In: CONFERENCE ON EMBEDDED NETWORKED SENSOR SYSTEMS, 2., 2004, Baltimore. **Proceedings**… 2004, p. 13-24.

YANG, Deli et al. Global software piracy: searching for further explanations. **Journal of Business Ethics**, v. 87, n. 2, p. 269-283, 2009.

ZAIDI, S. J. H.; WANG, H. On the analysis of software watermarking. In: INTERNATIONAL CONFERENCE ON SOFTWARE TECHNOLOGY AND ENGINEERING, 2., 2010, San Juan. **Proceedings**… 2010, p. V1-26-V1-30.

ZENG, Ying et al. Software watermarking through obfuscated interpretation: implementation and analysis. **Journal of Multimedia**, v. 6, n. 4, p. 329-340, 2011.

ZHU, Jianqi et al. H function based tamper-proofing software watermarking scheme. **Journal of Software**, v. 6, n. 1, p. 148-155, 2011.