Detecção de Vazamentos de Dados em Aplicativos Javascript em Dispositivos Móveis

Thiago de Souza Rocha[1], Eduardo Souto[1], Diego Azulay[1], Brandell Cássio[1], Alex Monteiro[1], Pedro Minatel[2], Breno Silva[2], Felipe Boeira[2]
[1]Instituto de Computação Universidade Federal do Amazonas (UFAM) Manaus, AM – Brasil
[2] Instituto de Pesquisa da Samsung, SP – Brasil

{thiago.rocha,esouto,dsa,bccf,alex.monteiro}@icomp.ufam.edu.br

{p.minatel,breno.p,f.boeira}@samsung.com

Abstract. *The development of applications with HTML5 and JavaScript that can be executed in multiple mobile devices like smartphones, tablets and others provided the appearance of multi-platform operating systems and increased the use of these languages. However most of these devices store sensitive information about the users and have become potential targets of attacks. One of the biggest concerns from companies that develop applications to these devices is the leakage or exposure of sensitive data. In this work we are addressing this problem by modifying the Tizen Web Runtime to add dynamic taint tracking, with that we can track sensitive information that is being leaked, even if the information is obfuscated, and warn the users. From our knowledge this is the first prototype that adds this kind of technique to Tizen and is the first prototype that tracks web applications in mobile devices. The results show that Dynamic Taint Tracking is a promising approach that can be improved and used to detect data leakage in mobile devices.*

Resumo. O desenvolvimento, utilizando as linguagens JavaScript e HTML5, de aplicativos que podem ser executados em diversos dispositivos móveis como celulares, tablets, entre outros proporcionou o surgimento de sistemas operacionais multiplataforma e aumentou a utilização dessas linguagens. Entretanto a maioria desses dispositivos armazenam informações confidenciais, também chamadas de informações sensíveis, sobre os usuários e tem se tornado um alvo em potencial de ataques. Uma das grandes preocupações das empresas que desenvolvem aplicações para esses dispositivos é justamente o vazamento dessas informações sensíveis. Este trabalho enfrenta esse problema através de uma modificação na Web Runtime do Tizen para adicionar uma análise dinâmica de informações sensíveis, com isso é possível acompanhar para onde a informação está indo e informar os usuários, mesmo que a informação esteja codificada. Pelo nosso conhecimento esse é o primeiro protótipo que adiciona esse tipo de técnica no Tizen e também é o primeiro protótipo que realiza análise dinâmica de informação em aplicações web em dispositivos móveis. Os resultados mostraram que a

abordagem é promissora e que pode ser usada para detectar vazamento de dados em dispositivos móveis.

1. Introdução

A contínua evolução das tecnologias web tem possibilitado o fácil acesso e interação com uma ampla variedade de dispositivos em várias plataformas, incluindo dispositivos móveis (e.g. celulares, notebooks e sensores de saúde), eletrodomésticos (e.g. smart TVs e Smart Geladeira) e redes de sensores veiculares.

Devido a essa variedade de dispositivos, aplicações móveis têm sido projetadas para serem executadas em várias plataformas e tal fato proporcionou o surgimento de novos sistemas operacionais multiplataformas como o Tizen [Tizen 2015], webOS [WebOS 2015] e Firefox OS [Firefox 2015] e aumentou a utilização de linguagens como HTML5 [HTML 2016] e JavaScript [ECMA-262 2016].

Graças ao HTML5 e aos navegadores web embarcados nos novos sistemas operacionais, as aplicações web desenvolvidas em JavaScript estão disponíveis em qualquer lugar [Bae et al. 2014]. A natureza multiplataforma da linguagem JavaScript possibilita que uma mesma aplicação possa ser executada em diferentes dispositivos. Alguns aplicativos Tizen (desenvolvidos em Javascript), por exemplo, executam em notebooks, smartphones, smart TVs e smartwatchs.

Por outro lado, esses dispositivos, na maioria das vezes, lidam com informações sensíveis do usuário. Portanto, garantir a segurança das aplicações que são executadas nesses dispositivos é uma tarefa prioritária e indispensável. Existem diversas abordagens que são utilizadas para analisar as aplicações e garantir a segurança, como testes manuais, testes automatizados usando alguma biblioteca como Jasmine[Jasmine 2016], análise estática[Kashyap et al. 2014] ou dinâmica de código[Seth et al. 2011], entre outras.

Este artigo descreve uma extensão do sistema operacional Tizen, denominada de TTizen (acrônimo de *Taint Tizen*), que realiza a análise dinâmica do fluxo de informações (do inglês, *dynamic taint analysis*) sensíveis em aplicações JavaScript. A meta primária é detectar quando uma informação sigilosa deixa o dispositivo por meio de aplicações de terceiros, caracterizando vazamento de dados. A abordagem proposta automaticamente marca um dado (*taint*) de uma fonte de informação considerada sensível e transitivamente rastreia esse dado à medida que ele se propaga por meio de variáveis e arquivos do programa. O TTizen registra os dados marcados e uma notificação é gerada para o usuário sempre que um dado marcado é transmitido pela rede.

Para alcançar esse objetivo, o TTizen apresenta uma modificação do motor (*engine*) JavaScript para realizar a análise *taint* dinâmica. Mais especificamente foram realizadas modificações no interpretador bytecode do WebKit [WebKit 2016], motor JavaScript usado no Safari e em outros navegadores livres.

A partir do nosso conhecimento, esse é o primeiro trabalho que aplica análise dinâmica de informação em JavaScript dentro de um ambiente móvel. De acordo com nosso levantamento, todas as pesquisas existentes nessa área realizam seus testes somente em navegadores web como mostrado na Seção 3.

O restante desse trabalho está organizado da seguinte forma. Seção 2 apresenta alguns conceitos relacionados à plataforma Tizen e análise de informação. Seção 3 detalha alguns trabalhos relacionados. Seção 4 apresenta o TTizen. Seção 5 mostra os experimentos realizados. Finalmente, a Seção 6 conclui o artigo e expõe alguns trabalhos futuros.

2. Conceitos Básicos

2.1. Plataforma Tizen

O Tizen é um sistema operacional móvel livre promovido principalmente pela Samsung e Intel para desenvolver uma distribuição Linux para vários tipos de dispositivos como automóveis, celulares, TVs inteligentes, relógios inteligentes, câmeras e geladeiras [Rick 2016]. Para acomodar essa grande variedade de dispositivos, o Tizen disponibiliza um conjunto de APIs (*Application Programming Interfaces*) e bibliotecas que possibilitam o desenvolvimento e execução de aplicações Web (escritas em JavaScript) e aplicações nativas (escritas em linguagens C e C++). Uma vantagem das aplicações desenvolvidas em linguagem JavaScript é a portabilidade entre diferentes plataformas.

A Figura 1 mostra a plataforma do Tizen composta pelos frameworks para aplicações web e nativas e algumas APIs (denominada de Core) no topo de um Kernel Linux. As aplicações web e nativa possuem seus próprios méritos como a fácil portabilidade e alto desempenho, respectivamente. Como citado anteriormente, este trabalho foca na detecção de vazamento de dados em aplicações web.

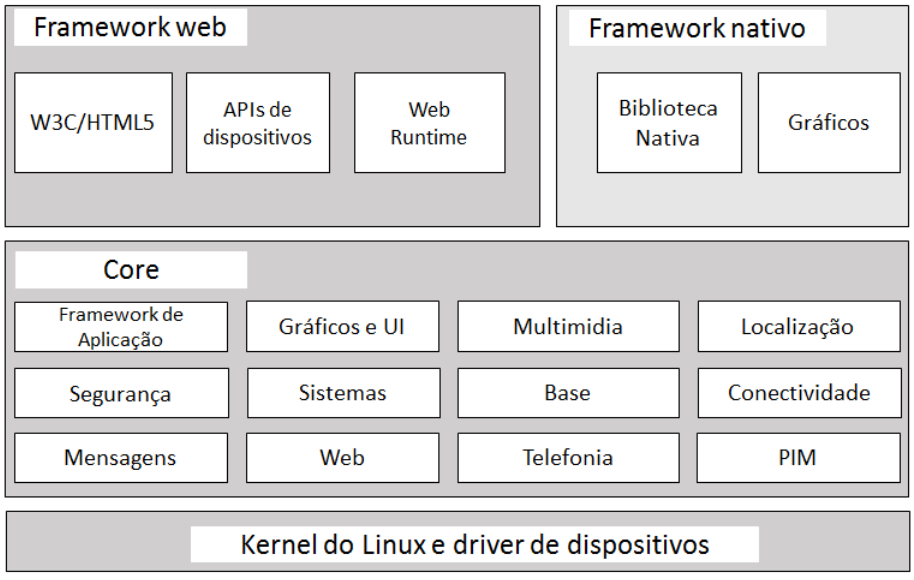

Figura 1. Arquitetura da plataforma Tizen.

2.2. Análise dinâmica do fluxo de Informação

Análise de informação é o processo de acompanhar como os dados marcados se propagam enquanto um programa é executado [Tonin 2016]. Os trabalhos de análise de

informação podem ser descritos por quatro elementos: marcação, fonte, propagação e saída, explicados abaixo:

- Marcação (*taint*): ato de adicionar uma marcação em cada objeto que represente informação sensível de um usuário.
- Fonte (*taint source*): local onde uma aplicação pode obter informações sensíveis sobre um usuário. Por exemplo, a API de localização do Tizen fornece uma interface que permite que uma aplicação obtenha informações sobre GPS.
- Propagação (*taint tracking*): processo de acompanhar (rastrear) um dado marcado enquanto o programa é executado.
- Saída (*taint sink*): interfaces de saída da aplicação, usadas para enviar informações sensíveis para fora da aplicação, por exemplo, uma interface HTTP.

O fluxo da informação que está sendo acompanhado pode ocorrer de duas formas: explícita e implícita [Seth et al. 2011]. No fluxo explícito a informação é passada diretamente entre uma variável marcada que usa o valor e a que recebe. O fluxo implícito é quando o valor de uma variável marcada afeta o valor de outra variável de forma indireta, por exemplo, através de uma estrutura condicional. Inicialmente esse trabalho foca apenas no fluxo explícito.

A análise de informação possui duas categorias, estática e dinâmica. A análise estática realiza a análise do código sem executar o programa, geralmente através da criação de um grafo de fluxo de programa (CFG), porém como JavaScript é uma linguagem dinâmica (permite execução de código através de funções como o *eval*) a análise estática do código se torna difícil [Bichhawat et al. 2014].

Por esse motivo foi escolhida a análise dinâmica que é o processo de executar o rastreio da informação enquanto o programa é executado. O Algoritmo 1 mostra um código fonte que exemplifica a análise dinâmica. Quando um valor de uma fonte de informação sensível (representada no código pelo método *source()*) é atribuído a variável *c*, essa variável recebe uma marcação (*taint*) sinalizando que seu conteúdo é sensível. Posteriormente o valor da variável c é passado para outra variável (*a* no caso do exemplo) e juntamente com o valor, a marcação de c também é passada para a, caracterizando uma propagação. Essa marca (*taint*) é rastreada dinamicamente até que seu conteúdo chegue a uma interface de saída, que no exemplo é o método *send()*.

Algoritmo 1

```
1:  c = source();
2:  a = b+c;
3:  send(a);
4: ...
```

3. Trabalhos Relacionados

Esta seção mostra trabalhos recentes relacionados à análise de informação dinâmica. Através das nossas pesquisas foi notado que todos os trabalhos em dispositivos móveis como [Enck et al. 2014], [Zheng e David 2014], [McClurg et al. 2016], [Zhao e Osorio 2012], [Yang e Yang 2012] e [Egele et al. 2011] realizam a análise das informações apenas nas aplicações nativas. Como o foco desse trabalho é realizar a análise em aplicações web a nossa pesquisa foi direcionada a trabalhos que realizam o rastreio em aplicações JavaScript. Entretanto, todos esses trabalhos fazem testes apenas em navegadores web.

[Tran et al. 2012] criaram uma técnica chamada principal-based tracking onde eles realizam a identificação de bibliotecas que possuem comportamento suspeito. A Uniform Resource Locator (URL) do código JavaScript é definida como principal. Para saber a qual principal cada parte de código JavaScript pertence, é adicionada uma marcação especifica em cada parte de código JavaScript que vai ser compilada em tempo de execução. Por fim, seu sistema monitora o comportamento dos scripts com diferentes principals e identifica os comportamentos suspeitos que acessam informações sensíveis.

Para verificar se algum script suspeito enviou informações sensíveis para servidores de terceiros, os autores desenvolveram um analisador de informação dinâmico em nível de variável apenas nos scripts suspeitos. Uma ferramenta chamada *LeakTracker* foi desenvolvida para testar essa técnica e um *benchmark* chamado SunSpider [SunSpider 2016] foi utilizado no Mozzila Firefox 3.6.13. Cada experimento foi repetido 10 vezes e apresentou um overhead em tempo de execução em torno de 2.2 vezes.

[Jang et al. 2015] propõem um *framework* que insere e propaga marcações enquanto uma aplicação é executada para reforçar a confidencialidade e políticas de integridade. Para realizar esse trabalho, os autores utilizam o código fonte da aplicação e adicionam as suas estruturas de dados e marcações. Os testes foram realizados no navegador Chrome usando benchmarks disponibilizados pela empresa Alexa [Alexa 2016]. Para checar o desempenho foram utilizados 10 benchmarks web onde foram coletados dados de execução do JavaScript e do carregamento das páginas obtendo um overhead em tempo de execução em torno de 2.4 vezes.

[Sayed et al. 2014] apresentam um modelo de controle de fluxo de informação que acompanha dinamicamente todas as informações e previne que uma informação sensível seja enviada para servidores de terceiros. Os autores mostram formalmente a sua abordagem e as modificações que devem ser feitas na semântica Javascript para que a abordagem funcione. Entretanto, nenhuma informação é disponibilizada sobre como a abordagem poderia ser implementada.

[Bichhawat et al. 2014] realizaram uma instrumentação no motor JavaScript para acompanhar as informações dinamicamente. Especificamente, o trabalho propõe a instrumentação do *bytecode* da WebKit e uma modificação do compilador para que ficasse compatível com a instrumentação. A abordagem proposta foi avaliada usando o benchmark SunSpider [SunSpider 2016] no navegador safari e os resultados mostraram um overhead em tempo de execução entre 1.1 e 0.5 vezes.

[Hedin e Sabelfeld 2012] propuseram um controle de fluxo de informação dinâmico para a linguagem que modela as principais características do JavaScript porém foram ignoradas algumas instruções como o *break* e o *continue* e precisam testar a sua abordagem em um navegador real.

A partir dos trabalhos relacionados é possível verificar que toda pesquisa que aplica análise de informação dinâmica em aplicações JavaScript realiza seus testes em um navegador web e possui um overhead considerável. Nossa proposta visa realizar a análise de aplicações JavaScript em dispositivos móveis pois é uma lacuna que existe nesse tipo de pesquisa e também para deixar as aplicações JavaScript nos dispositivos móveis mais seguras.

4. TTizen

4.1. Visão Geral

A abordagem proposta foi concebida para permitir que os usuários possam monitorar como as aplicações embarcadas em dispositivos móveis lidam com os seus dados privados em tempo real. Especificamente, os aplicativos espalhados nos diferentes dispositivos Tizen (e.g. smart TVs, smart geladeiras, smartwatchs, e smartphones) são desenvolvidos em Javascript, que por ser uma linguagem dinâmica torna difícil uma análise estática do código [Bichhawat et al. 2014].

Para tratar esse problema, nós incorporamos ao sistema operacional Tizen um sistema de marcação de dados em tempo real que envolve os seguintes passos básicos:

- Identificar as fontes de informações sensíveis no Tizen (*taint source*) como lista de contatos, e-mail, informações de localização (e.g. dados do GPS), fotos, etc.
- Realizar a marcação das fontes (*taint marking*) de informações sensíveis.
- Definir uma política para rastreamento dos dados marcados (*taint analysis*).
- Identificar as saídas utilizadas pelas aplicações (*taint sink*), usualmente uma interface de rede.

A Figura 2 mostra um diagrama da Web Runtime (WRT) do framework Tizen versão 2.2, onde o sistema de marcação de dados foi implementado. A WRT é responsável por todo ciclo de vida de uma aplicação Tizen desenvolvida em JavaScript, desde a sua instalação até a sua execução dentro da WebKit. Além disso, a WRT fornece um conjunto de APIs que facilitam o desenvolvimento de aplicações provendo informações sobre os dispositivos móveis.

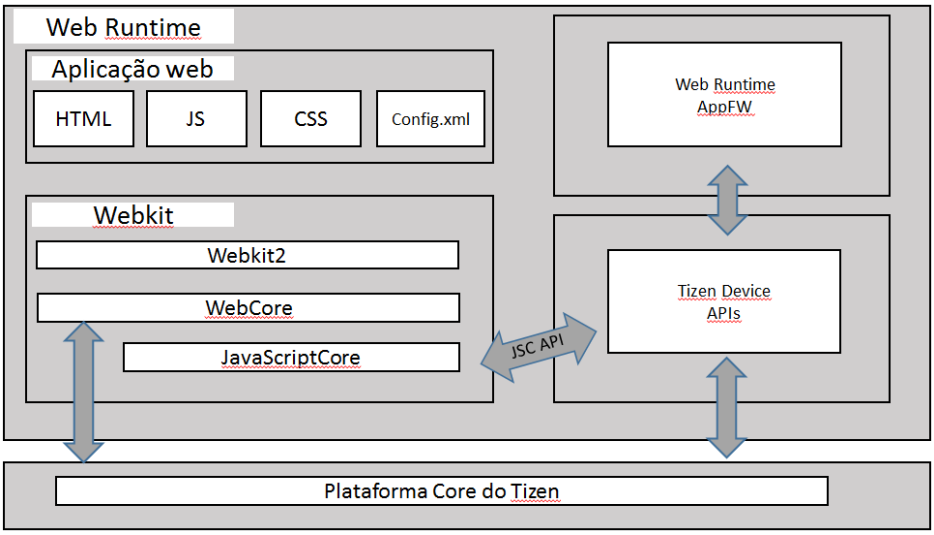

Figura 2. Web Runtime do Tizen.

Primeiramente, foi realizado um estudo em nível de instrução para identificar e marcar as fontes de informações sensíveis. Analisando as APIs de dispositivos móveis do Tizen, nós identificamos as seguintes fontes: código do telefone, lista de contato, número de telefone, informações de rede, endereço de correio eletrônico, fotos e registros de chamadas.

Uma vez que os dados considerados sensíveis foram rotulados (*tainted*), modificações foram realizadas nos componentes da Webkit, especificamente no *engine* do JavaScript (JavaScriptCore), para acompanhar o dado marcado a medida que se propaga durante a execução da aplicação.

A instrumentação do componente de saída foi realizada no componente WebCore da Webkit que é responsável por verificar se uma informação sensível está sendo enviada e por escrever as informações necessárias em um arquivo de log. Atualmente, apenas o envio de informações através de uma conexão HTTP está sendo tratado como saída.

Por fim, para informar o usuário que alguma informação sensível está sendo enviada foi criada uma aplicação Tizen nativa que monitora um arquivo de log. Toda vez que esse arquivo de log é modificado a aplicação lê essas alterações e informa ao usuário qual informação esta sendo vazada e o horário do vazamento.

4.2. Detalhes de Implementação

Esta Seção fornece detalhes das modificações realizadas no componente Web Runtime do Tizen versão 2.2 para realizar a análise de informação dinâmica. Para facilitar a identificação do fluxo, esta seção será dividida em 3 partes : Identificação de fontes, Propagação e Identificação das saídas.

4.2.1 Identificação das fontes

As APIs Tizen se encontram dentro do diretório framework/wrt-plugins-tizen, tendo como diretório raiz todo o projeto do tizen. Entre elas se encontra uma que possui acesso a informações do sistema (*Systeminfo*) e contém alguns métodos para acessar propriedades tanto do sistema quanto do dispositivo, tais como número do IMEI, estado da rede, nível de bateria, etc. Para ter acesso a essas propriedades é preciso fazer uma chamada para o método *getPropertyValue()*.

Para utilizar este método, é necessário seguir a seguinte sintaxe: tizen.systeminfo.getPropertyValue (<Informação>, <Função de Sucesso>, <Função de Falha>), onde:

- <Informação> é a propriedade que se quer do sistema, como o modelo do dispositivo.
- <Função de sucesso> em caso de nenhuma falha na obtenção da informação sensível, essa função é chamada.
- <Função de falha> é chamada caso algo aconteça de errado no na execução do método, sendo opcional.

A implementação do source consiste em marcar o retorno das chamadas que estão na API *Systeminfo,* com a adição de uma *tag* (neste caso, foi utilizada a tag "\1") no final da string que representa a informação sensível.

4.2.2 Propagação

A propagação é feita dentro da JavaScriptCore, núcleo de interpretação e execução do código JavaScript que fica na WebKit. Os arquivos que foram modificados estão na pasta /framework/web/webkit-efl/Source/JavaScriptCore/llint. Nesse diretório há implementações das operações do bytecode gerado pela JavaScriptCore.

Atualmente está sendo tratado apenas o fluxo explícito que pode ser divido em duas categorias: As operações de atribuição, como no caso de operações que passam um valor diretamente, e as que retornam algum valor como no caso de concatenação para strings. Estas operações são geradas pelo bytecode da JavaSriptCore e são compostas por várias operações em nivel de assembly. As categorias serão detalhadas a seguir.

4.2.2.1 Operações de Atribuição

Estas operações atribuem a um espaço de memória, um determinado valor. No código assembly gerado pela JavaScriptCore existem três definições de operações que são responsáveis pela atribuição: mov, put_global_var e put_by_id que serão explicadas a seguir.

4.2.2.1.1 Operação mov

Essa operação é implementada em uma linguagem própria da JavaScriptCore sendo definida como: mov <Reg>, <Valor>. Em que <Reg> é um registrador definido pela linguagem de bytecode gerado pela JavaScriptCore, e <Valor> é o valor que será atribuido para o mesmo. Esta operação é realizada quando se atribui um valor a uma variável dentro de uma função (definida ou anônima), como mostra o Algoritmo 2.

Algoritmo 2

```
1:  function def() {
2:     var a  = 5;
3:     .........
4:  }
```

A operação "var a = 5;", transcrita para bytecode ficaria como "mov r1, @k1", em que @k1 representa uma constante dentro do contexto da função. Como no contexto dessa função existe apenas uma constante, então @k1 representa 5.

4.2.2.1.2 Operação put_global_var

Essa operação também é implementada em uma linguagem própria da JavaScriptCore e atribui valores para uma variável quando esta se encontra num escopo global, fora de uma função. Ela é definida da forma put_global_var <Reg>, <Valor>. Seguindo o exemplo do Algoritmo 3 a operação da linha 1 "var a = 5;" seria transcrita para "put_global_var r1, @k1".:

Algoritmo 3

```
1:  var a = 5;
2:  function def() {
3:     .........
4:  }
```

4.2.2.1.3 Operação put_by_id

A operação put_by_id é implementada na linguagem C e é utilizada quando não é definida uma variável com a palavra reservada "var", como no caso de "a = 5". Essa operação é mais lenta pelo fato de que a JavaScriptCore necessita mapear explicitamente essa variável.

4.2.2.2 Operações de Retorno

As operações de retorno são as operações aritiméticas no caso de inteiros e para strings são operações como concatenação, tamanho da string, entre outras. O algoritmo 4 mostra um exemplo de adição.

Algoritmo 4

```
1:  var a = 5;
2:  var b =6;
3:  var c = a + b;
```

A operação da linha 3 "var c = a + b", é transformada em bytecode para "add r1, r2,r3" em que r2 e r3 representam respectivamente o valor de a e de b, e r1 representa o valor onde deve ser guardado o resultado da soma, que será reservado para a variável c.

Para verificar a propagação foi feita uma checagem em todas essas operações para avaliar se algum dos registradores envolvidos está marcado. Caso esteja marcado, a marcação é adicionada no resultado da operação. Um pseudocódigo é apresentado no Algoritmo 5.

Algoritmo 5

```
1:  propagacao (Objeto v_1, v_2, ...., v_n)
2:  Inicio
3:     resultado = operacao (v_1, v_2, ...., v_n);
4:        Se estaMarcado( v_1, v_2, ...., v_n) então
5:           resultado.adicionaMarca()
6:        Fim se
7:     retorna resultado;
8:  Fim
```

4.2.3 Identificação de saídas

Para verificar a saída foi modificada a classe WebSocket que se encontra no WebCore, nela é verificado quando uma conexão HTTP é criada. Foi adicionado um teste no método *send*() dessa classe para verificar se a informação que está saindo encontra-se marcada (*tainted*). Caso isso aconteça, um vazamento de dados é detectado e o método de log é chamado para registrar a informação sensível e data e hora (*timestamp*) que esse evento ocorreu.

5. Experimentos e Resultados

Essa seção apresenta os resultados obtidos através da avaliação de desempenho da abordagem. Para realizar os experimentos foram utilizadas dois dispositivos ODROID U3 [ODROID 2016] da empresa Hardkernel. Estes dispositivos oferecem suporte a distribuições Linux como Xubuntu 13.10 e Android 4.X, além de oferecer suporte as distribuições Tizen 2.2 e 2.3. O hardware é composto por um processador Arm Quad-Core 1.7GHz,om 2GB de RAM, com suporte a cartão de memória de 64 GB, memória eMMC e saídas de audio e de comunicação serial. Tais configurações se assemelham a um Smartphone SAMSUNG Galaxy S3.

Para validar e comparar as alterações foram realizados experimentos quantitativos e qualitativos. Nos experimentos quantitativos foram medidos a porcentagem de uso de CPU e de memória dos processos relacionados à execução de uma aplicação de teste. No Tizen cada aplicativo web possui dois processos. O primeiro

é responsável pelo ciclo de vida da aplicação e o segundo é responsável por gerenciar e renderizar o conteúdo da aplicação, denominado de *Web Process* [Jaygarl et al. 2016].

No experimento qualitativo foi verificado se a abordagem conseguia identificar um vazamento de dados primeiramente com a informação sendo passada diretamente e posteriormente com a informação sendo passada de forma codificada. Devido à utilização de um dispositivo limitado para embarcar o Tizen (ODROID U3), que não possui propriedades como IMEI e SIM Card, foi utilizada como informação sensível o tipo de rede em que o aparelho está conectado (3G, 4G, WiFi ou NONE). No caso do ODROID U3 o Tizen sempre retorna NONE. Por ser um sistema operacional novo, o Tizen não possui nenhum benchmark público disponível. Portanto, para fazer a análise quantitativa, foi desenvolvida e utilizada uma aplicação Tizen que obtém informações sensíveis do usuário. Como foi utilizada uma plataforma que não possui todas as características de um dispositivo móvel não foi possível testar todas as fontes de informações sensíveis.

A aplicação desenvolvida obtém informações sensíveis do sistema, realiza a propagação e envia os dados para um servidor externo a cada dois segundos. O tempo de cada experimento foi de 10 minutos, esse processo foi repetido seis vezes, totalizando uma hora de experimento.

Durante esse tempo, foram obtidas informações sobre a porcentagem de uso de CPU e memória. Para auxiliar na obtenção das informações foi desenvolvido um *shell* script utilizando a ferramenta PS [PS 2016] do sistema operacional Linux que é utilizada para mostrar os processos que estão sendo executados no sistema operacional Linux.

A Figura 3 mostra a porcentagem de utilização de memória e CPU do aplicativo de teste que foi desenvolvido. Esse aplicativo recebeu o nome de APP e a informação sensível que é transmitida através de uma conexão HTTP são informações sobre a rede. O aplicativo foi executado em dois ODROIDs, um com o TTizen e outro com o Tizen original sem modificações.

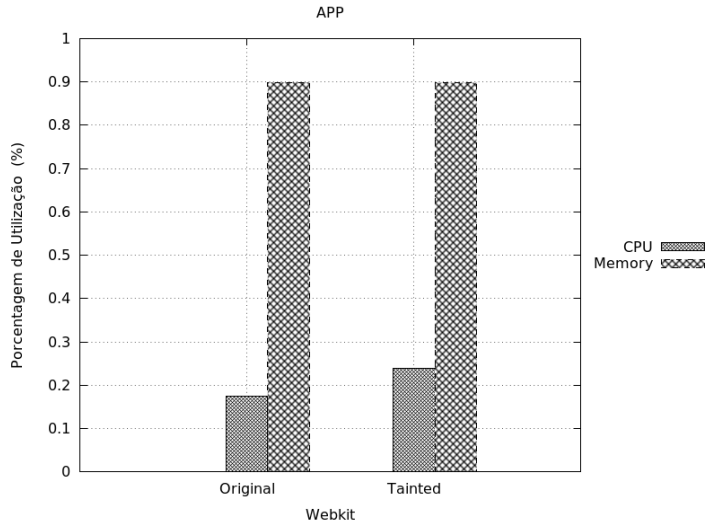

Figura 3. Porcentagem de utilização de memória e CPU do APP em um ODROID com TTizen e outro ODROID com Tizen original.

Os resultados mostram que a porcentagem de utilização de memória da aplicação usando um ODROID com o TTizen e outro ODROID com o Tizen original foi a mesma de 0,9% enquanto que a porcentagem de utilização de CPU foi de aproximadamente 0.19% no ODROID com a versão original e de 0.22% no ODROID com TTizen, obtendo um pequena diferença de 0.03%. A Figura 4 mostra a porcentagem de utilização de memória e CPU do *Web Process* da APP.

Os resultados mostram que a porcentagem de utilização de memória do *Web Process* usando um ODROID com o TTizen foi de aproximadamente 1,7% e a porcentagem de utilização de CPU no ODROID com a versão original foi de aproximadamente 0,9%, obtendo uma diferença de 0,8% enquanto que a porcentagem de utilização de CPU foi de aproximadamente 0.3% no ODROID com a versão original e de 1.5% no ODROID com TTizen, obtendo uma diferença de 1.2%.

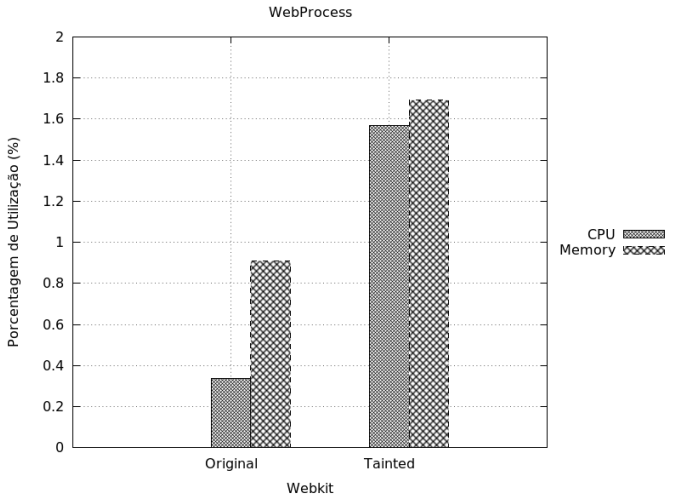

Figura 4. Porcentagem de utilização de memória e CPU do Web Process em um ODROID com TTizen e outro ODROID com Tizen original.

Para o experimento qualitativo foi desenvolvida uma aplicação, denominada *MyObserver*, para notificar o usuário caso ocorra vazamento de dados. Essa aplicação foi desenvolvida em C++ e funciona como um serviço que e em segundo plano para monitorar um arquivo de log.

Quando um vazamento de dados ocorre a informação e horário do vazamento são adicionadas a um arquivo de log, posteriormente esse arquivo é lido e o usuário notificado. A Figura 5 mostra o resultado do experimento qualitativo com a APP sendo executada e com a MyObserver exibindo uma notificação no topo da tela informando ao usuário que uma informação sensível estava sendo vazada.

No terceiro experimento foi utilizada a mesma aplicação que envia as informações sensíveis, porém a informação enviada foi ofuscada através da aplicação de uma codificação *Base64* [Base64 2016]. Como a análise dinâmica da informação realiza o rastreio da informação durante toda a execução da aplicação mesmo que seja aplicada alguma codificação a informação continuará marcada e acompanhada, portanto o experimento obteve os mesmos resultados do experimento anterior com a aplicação MyObserver enviando uma notificação para o usuário.

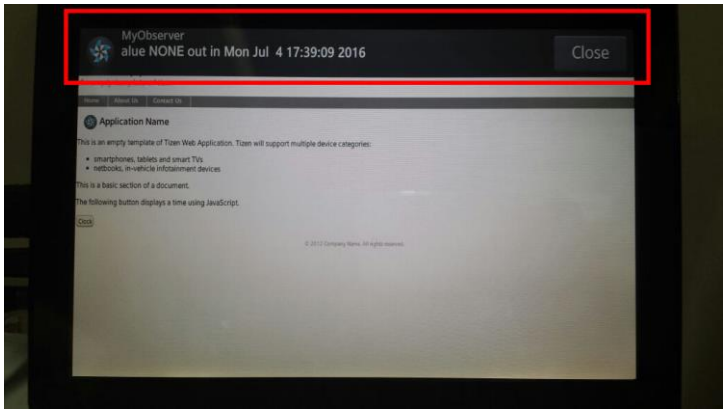

Figura 5. Usuário sendo informado sobre um vazamento de dados no topo da tela do ODROID U3.

6. Conclusões e Trabalhos Futuros

Este artigo apresentou uma versão modificada do sistema operacional Tizen que aplica análise de informação dinâmica em aplicações web. Para realizar a análise dinâmica das informações foi necessário um estudo sobre o sistema operacional Tizen e de como funciona a sua WebKit. Pelas nossas pesquisas esse é o primeiro protótipo que aplica esse tipo de técnica ao sistema operacional Tizen e também é o primeiro protótipo que realiza rastreio de aplicações web em um ambiente móvel.

Os resultados mostraram que no processo de execução do aplicativo no ODROID com TTizen e com o Tizen original a porcentagem de utilização de memória foi a mesma (0,9%) enquanto que a porcentagem de utilização de CPU obteve uma pequena diferença de 0.3%.

Já no processo responsável pela renderização do aplicativo (*Web Process*) a diferença da porcentagem de utilização de memória no ODROID com TTizen e no ODROID com Tizen original foi de 0,8% enquanto que a diferença da porcentagem de utilização de CPU foi de 1.2%. Por fim, no experimento qualitativo o TTizen foi efetivo e informou ao usuário que uma informação sensível estava sendo enviada para um servidor externo.

Para trabalhos futuros serão realizadas as modificações necessárias para cobrir tratamento de exceções, variáveis do tipo real e, principalmente, a propagação implícita, pois diferentes técnicas tem que ser empregadas para abordar esse tipo de propagação.

Outro aspecto pesquisado é que será estudada uma forma de descobrir se o vazamento de dados é com a permissão do usuário, pois existem vezes em que o usuário realmente precisa enviar a informação sensível para servidores de terceiros.

Agradecimentos. Partes dos resultados apresentados nesse artigo foram obtidas através de pesquisas em um projeto chamado "Um Sistema para Detectar e Prevenir Vazamento de Dados em um Ambiente Android", patrocinado pela Samsung Eletrônica da Amazônia Ltda. Dentro dos termos da lei federal brasileira No. 8.387/91. (SUFRAMA).

Referências

Abhishek Bichhawat, Vineet Rajani, Deepak Garg e Christian Hammer “Information Flow in WebKits JavaScript Bytecode” Disponível em: http://www.mpi-sws.org/~dg/papers/post2014.pdf, Janeiro 2016.

Alexa “Alexa – Top Sites by Category: Computers/Performance and Capacity/Benchmark” Disponível em: http://www.alexa.com/topsites/category/Computers/Performance_and_Capacity/

Benchmarking, Janeiro 2016.

Base64. “Base64 encoding and decoding – Web APIs | MDN”. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/WindowBase64/Base64_encoding_and_decoding. Janeiro 2016.

Bassam Sayed, Issa Traoré and Amany Abdelhalim (2014) “Detection and Mitigation of Malicious JavaScript Using Information Flow Control”. Proceedings of the Twelfth Annual International Conference on Privacy, Security and Trust, páginas 264-273.

Daniel Hedin and Andrei Sabelfeld (2012) “Information-flow security for a core of JavaScript”. Proceedings of 25th IEEE Computer Security Foundations Symposium, páginas 3–18.

Dongseok Jang, Ranjit Jhala, Sorin Lerner (2015). Disponível em: https://pdfs.semanticscholar.org/60b9/11615642efeabcb3485c093d7cade0fa77b8.pdf , Dezembro 2015.

ECMA-262 “Standard ECMA-262” Disponível em: http://www.ecma-international.org/publications/standards/Ecma-262.htm, Janeiro 2016.

Firefox OS “Firefox OS – Just what you need – Great smartphone features, apps and more - Mozzila” Disponível em: https://www.mozilla.org/en-US/firefox/os/, Janeiro 2016.

HTML5 “HTML5” Disponível em: https://www.w3.org/TR/html5/, Janeiro 2016.

JASMINE “Jasmine: Behavior-Driven Javascript” Disponível em: http://jasmine.github.io/, Janeiro 2016.

Jaygarl Hojun, Luo Cheng, Kim YooSoo, Choi Eunyoung, Bradwick Kevin, Lansdell Jon “Professional Tizen Application Development” Disponível em: https://goo.gl/uWIpvn, Janeiro 2016.

Jedidiah McClurg, Jonathan Friedman and William Ng “Android Privacy Leak Detection via Dynamic Taint Analysis”. Disponível em: http://www.jrmcclurg.com/papers/ internet_security_final_report.pdf Fevereiro 2016.

Manuel Egele, Christopher Kruegel, Engin Kirda and Giovanni Vigna (2011) “PiOS: Detecting Privacy Leaks in iOS Applications”. Proceedings of the International Secure Systems Lab.

Minh Tran, Xinshu Dong, Zhenkai Liang and Xuxian Jiang (2012) “Tracking the trackers: fast and scalable dynamic analysis of web content for privacy violations”. Proceedings of the 10th International Conference on Applied Cryptography and Network Security, páginas 418-435.

ODROID. “ODROID | Hardkernel”. Disponível em: http://www.hardkernel.com/main/main.php. Janeiro 2016.

PS. “PS man page”. Disponível em: http://www.petefreitag.com/tools/man-pages/ps.html. Janeiro 2016.

Rick Lehrbaum “Slides from Intels Tizen talk at IDF2013 Beijing” Disponível em: http://hackerboards.com/intel-tizen-talk-slides-idf2013/, Fevereiro 2016.

Seth Just, Alan Cleary, Brandon Shirley e Christian Hammer (2011) “Information flow analysis for javascript”. Proceedings of the 1st ACM SIGPLAN international workshop on Programming language and system technologies for internet clients, páginas 9-18.

SungGyeong Bae, Hyunghun Cho, Inho Lim e Sukyoung Ryu (2014) “SAFEWAPI: web API misuse detector for web applications”. Proceedings of the 22nd ACM SIGSOFT international symposium on Foundations of Software Engineering, páginas 507-517.

SunSpider “SunSpider 1.0.2 JavaScript Benchmark” Disponível em: https://webkit.org/perf/sunspider/sunspider.html, Janeiro 2016.

Tizen. “Tizen | An open source, standards-based software platform for multiple devices categories”. Disponível em: https://www.tizen.org. Janeiro 2016.

Tonin, G., “Tendências em computação móvel” Disponível em: http://grenoble.ime.usp.br/~gold/cursos/2012/movel/mono-1st/2305-1_Graziela.pdf, Janeiro 2016.

Vineeth Kashyap, Kyle Dewey, Ethan A. Kuefner, John Wagner, Kevin Gibbons, John Sarracino,Ben Wiedermann e Ben Hardekopf (2014) “JSAI: a static analysis platform for JavaScript”. Proceedings of the 22nd ACM SIGSOFT international symposium on Foundations of Software Engineering, páginas 121-132.

Wei Zheng, Lie David (2014) “LazyTainter: Memory-Efficient Taint Tracking in Manager Runtimes”, ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, páginas 27-38.

webOS “Open webOS” Disponível em: http://www.openwebosproject.org/, Janeiro 2016.

WebKit “WebKit” Disponível em: https://webkit.org/, Janeiro 2016.

William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel and Anmol N. Sheth (2014) “TaintDroid: An Information Flow Tracking System for Real-Time Privacy Monitoring on Smartphones”, ACM transaction on Computer Systems (TOCS), páginas 99-106.

Zhibo Zhao and Fernando C. Colon Osorio (2012) “TrustDroid™": Preventing the use of SmartPhones for information leaking in corporate networks through the use of static analysis taint tracking”, International Conference on Malicious and Unwanted Software, páginas 135-143.

Zhemin Yang and Min Yang (2012) “Leakminer: Detect Information Leakage on Android with static taint analysis”, World Congress on Software Engineering (WCSE), páginas 101-104.

