Dizang: Uma solução para coleta de evidências forenses de ataques de injeção na nuvem

Hamilton J. S. Fonte II, Marcos A. Simplicio Jr.

[1]Escola Politécnica – Universidade de São Paulo (USP)
Av. Prof. Luciano Gualberto 380 - Butantã – 05.508-010 – São Paulo – SP – Brasil

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

Resumo. *A adoção de arquiteturas em nuvem aumenta a cada dia, e com ela também o número de casos em que esse tipo de tecnologia é usada para fins ilícitos. Infelizmente, devido à natureza volátil da nuvem, a tarefa de coletar evidências para análise forense nesse ambiente tem esbarrado em desafios práticos e legais. Este trabalho analisa propostas na literatura voltadas a resolver os principais desafios existentes na coleta evidências na nuvem, discutindo suas limitações, e então propõe uma solução que cobre coleta, transporte e armazenamento da evidência visando suplantá-las. A solução aqui proposta provê uma forma de correlacionar evidências e sua origem virtual, permitindo transportar e armazenar tais dados sem afetar sua credibilidade.*

1. Introdução

Técnicas de virtualização, replicação de serviços e compartilhamento de recursos entre múltiplos usuários (multi-inquilinato) proveem a nuvens computacionais uma elevada escalabilidade [Morsy et al. 2010]. Ao mesmo tempo, tais mecanismos também criam uma elevada volatilidade dos recursos virtuais que executam aplicações em nuvem. Afinal, quando submetida a uma carga elevada, uma aplicação hospedada na nuvem pode criar clones das máquinas virtuais (*virtual machines* – VMs) que a hospedam e balancear a carga entre elas, de modo a atender à demanda sem prejuízos na qualidade do serviço oferecido. Após esse pico, as máquinas que foram clonadas são normalmente desativadas, seus recursos liberados e o sistema retorna à capacidade anterior, evitando-se custos desnecessários

Embora interessante do ponto de vista de eficiência e custos, do ponto de vista forense a volatilidade da nuvem traz problemas em caso de ataques. Por exemplo, caso uma das instâncias de processamento virtuais criadas temporariamente seja alvo de ameaças que atuam diretamente na sua memória, sem deixar rastros em discos (e.g., arquivos de *log*), as evidências desse evento podem ser completamente perdidas após elas serem desativadas e terem seus recursos liberados. Essa dificuldade é ainda agravada por aspectos como multi-inquilinato e multi-jurisdição típicas de soluções em nuvem [Keyun et al. 2011]. Especificamente, o aspecto multi-inquilino dificulta a obtenção do *hardware* que executa as aplicações de interesse, pois, como ele é compartilhado por vários usuários, removê-los para análise poderia levar a uma violação de privacidade

dos usuários não relacionados à investigação. Já a natureza distribuída da nuvem pode levar à alocação de informações relevantes à investigação em vários países, dificultando a obtenção das mesmas em especial quando não existem acordos de cooperação entre as entidades envolvidas [Dykstra and Sherman 2012]. Combinadas, tais características dificultam a coleta de evidências com a credibilidade necessária para que elas possam ser usadas em processos legais, o que exige o respeito à privacidade, à jurisdição e à cadeia de custódia, bem como a reprodutibilidade do processo de coleta [Rahman and Khan 2015].

Embora existam soluções na literatura que abordam a coleta de informações de nuvem com o propósito de análise forense, a grande maioria delas aborda a coleta, o transporte e o armazenamento de forma isolada. Por exemplo, trabalhos como [Reichert et al. 2015] tratam de fatores como multi-inquilinato e multi-jurisdição, discutindo formas de coleta e preservação da evidência fora da nuvem. Já estudos como [George et al. 2012] se concentram na análise forense para a coleta de evidência de máquinas virtuais enquanto elas estão em execução, enquanto trabalhos como [Sang 2013] abordam a questão de processos de garantia de cadeia de custódia em ambientes de nuvem para transporte da evidência. Por outro lado, não foram identificadas na literatura propostas que (1) descrevam como o dado é coletado e armazenado preservando sua integridade e confidencialidade, e (2) visem garantir que, mesmo que um recurso virtualizado (e.g., uma VM) seja desalocada, haja condições de se reproduzir o processo de coleta de evidências.

O presente trabalho visa suplantar tais limitações por meio de uma proposta que tem como focos (1) a reprodutibilidade do processo de coleta, (2) o estabelecimento de vínculo entre a evidência coletada e sua origem e (3) a preservação da jurisdição e da privacidade dos não envolvidos na investigação. Em suma, a solução descrita provê uma forma de correlacionar evidências e sua origem virtual, permitindo transportar e armazenar tais dados de modo a preservar sua credibilidade. Para isso, a proposta supõe que o sistema sendo monitorado é executado dentro de um contêiner em nuvem. O foco da solução em contêiner se justifica pelo crescimento da adoção de conteinerização nos últimos anos, bem como pela previsão de que este será o modelo de implementação mais usado em aplicações futuras [Piraghaj et al. 2016]. A solução tem como alvo específico ataques de injeção de código [Case et al. 2014], pois estes, quando usados contra uma arquitetura em nuvem, não deixam rastros quando recursos de processamento virtuais são desativados e sua memória é liberada [Case et al. 2014]. Em particular, têm especial interesse quatro tipos específicos dessa família de ameaças [Case et al. 2014]:

- **Injeção remota de bibliotecas**: Um processo malicioso força o processo alvo a carregar uma biblioteca em seu espaço de memória. Como resultado, o código da biblioteca carregada executa com os mesmos privilégios do executável em que ela foi injetada. Esta estratégia, comumente usada para instalar malwares, pode fazer com que uma biblioteca maliciosa armazenada no sistema seja distribuída por vários processos de uma mesma máquina, dificultando sua remoção [Miller and Turkulainen 2004].
- **Inline Hooking**: Um processo malicioso escreve código como uma sequência de bytes diretamente no espaço de memória de um processo alvo, e então força este último a executar o código injetado. O código pode ser, por exemplo, um *script* de *shell*.

- **Injeção reflexiva de biblioteca**: Um processo malicioso acessa diretamente a memória do processo alvo, inserindo nela o código de uma biblioteca na forma de uma sequência de bytes, e então força o processo a executar essa biblioteca. Nessa forma de ataque, a biblioteca maliciosa não existe fisicamente; isso torna tal estratégia de injeção de código potencialmente mais atrativa, pois o carregamento da biblioteca não é registrado no sistema operacional (SO), dificultando a detecção do ataque [Fewer 2008].
- **Injeção de processo vazio**: Um processo malicioso dispara uma instância de um processo legítimo no estado "suspenso"; a área do executável é então liberada e realocada com código malicioso.

O restante deste documento está organizado da seguinte forma. A Seção 2 discute brevemente soluções em nuvem. A Seção 3 analisa os trabalhos relacionados na área de forense de memória. A Seção 4 detalha a solução proposta e avalia como ela trata os desafios alvo deste projeto. Finalmente, a Seção 5 apresenta algumas considerações finais.

2. Adoção de arquiteturas em nuvem e contêineres

Uma nuvem computacional é um modelo de infraestrutura no qual recursos compartilhados em quantidade configurável, acessíveis via rede, são alocados e desalocados com esforço mínimo de gerenciamento por parte de um provedor de serviços. Há três modelos principais de comercialização de uso da nuvem: *software* como serviço (*Software as a Service* – SaaS), na qual se provê o *software* que será usado pelo cliente; plataforma como serviço (*Platform as a Service* – PaaS), na qual se provê o ambiente para que o cliente desenvolva, teste e execute seu *software*; e, o tipo mais pertinente para este trabalho, Infraestrutura como serviço (*Infrastructure as a Service* – IaaS), na qual são fornecidos recursos computacionais básicos, como processamento e memória, em geral de forma virtualizada. [Mell and Grance 2011]

A virtualização de recursos na nuvem, embora tradicionalmente feita por meio de máquinas virtuais, vêm sendo crescentemente feita também na forma de contêineres. De fato, segundo o "Container Market Adoption Survey 2016", realizado pelas empresas DevOps.com (https://devops.com/) e ClusterHQ (https://clusterhq.com) com 235 empresas que têm desenvolvimento de software como sua atividade fim ou como suporte à atividade fim, 76% dos respondentes utilizam contêineres para melhorar a eficiência do processo de desenvolvimento e em suas arquiteturas de micro-serviços em nuvem. Diferente de máquinas virtuais, que envolvem a criação de um *hardware* virtual e de um sistema operacional (SO) acima do sistema nativo e que opera independente deste, a virtualização com contêineres é feita no nível do SO nativo, tem uma implementação mais simples eliminado camadas entre o aplicativo executado e o *hardware* físico. Uma tecnologia bastante utilizada para esse propósito são Contêineres Linux (LXC) [Linuxcontainers.org 2015], que aproveitam-se de funcionalidades como cgroups e namespacing do kernel do Linux para auxiliar no gerenciamento e isolamento de recursos virtuais.

3. Trabalhos relacionados

Existem vários aspectos relativos à análise forense na nuvem. Para uma discussão mais estruturada dos trabalhos disponíveis na literatura sobre o tema, a seguir eles são apresentados com base nos diferentes aspectos que abordam.

3.1. Acessar e coletar as informações de memória das máquinas virtuais em nuvem

Diversos trabalhos de análise forense na nuvem se concentram na coleta de dados "após o fato", ou seja, após a intrusão ser detectada [Reichert et al. 2015, Poisel et al. 2013, Dykstra and Sherman 2013, George et al. 2012, Sang 2013]. Os processos de coleta descritos nesses trabalhos podem ser iniciados de forma manual ou automaticamente, via integração com um mecanismo de detecção de intrusão. No caso específico de memória volátil, tal forma de coleta não consegue descrever como era a memória antes da intrusão, pois o processo só é acionado depois da detecção do ataque. Tal limitação pode trazer prejuízos à investigação, dado que algumas análises dependem exatamente da capacidade de se comparar dois momentos da memória [Case et al. 2014]. Entre os trabalhos estudados, a única proposta encontrada que leva tal necessidade em consideração é [Dezfouli et al. 2012], que propõe que o dado seja armazenado no próprio equipamento sob análise. Infelizmente, entretanto, a aplicação de tal abordagem no cenário em nuvem é pouco viável, pois pode levar à perda de informações importantes caso a máquina virtual ou contêiner seja desativada, tendo seus recursos liberados.

Existem ainda trabalhos voltados à coleta de informações durante a execução do sistema, nos quais os dados são constantemente coletados sem distinção do que aconteceu antes ou depois do fato de interesse. Esse é o caso de trabalhos como [Poisel et al. 2013, Dykstra and Sherman 2013, Sang 2013], que adotam a estratégia de isolar e parar a máquina virtual para em seguida realizar o processo de coleta. Embora interessantes, as abordagens descritas nesses trabalhos podem levar a um elevado volume de dados coletados, além de também não tratarem o cenário em que é necessário coletar evidências quando os recursos virtuais contendo tais informações são liberados.

3.2. Capacidade de reproduzir o processo e obter os mesmos resultados

Se, durante uma análise forense, analistas diferentes obtêm resultados distintos ao executar o mesmo procedimento de coleta, a evidência gerada não tem credibilidade, inviabilizando seu uso em um processo legal. Por essa razão, a reprodutibilidade do processo de coleta é uma parte importante da geração de evidências para análise forense. Infelizmente, entretanto, nenhuma das propostas encontradas na literatura atualmente permite tal reprodutibilidade em cenários de nuvem em que máquinas virtuais ou contêineres são desativados e seus recursos físicos liberados: todas elas dependem da existência do recurso virtual para a repetição do processo de coleta.

3.3. Não violar privacidade ou jurisdição das partes não envolvidas na investigação

Em um ambiente de nuvem pública, remover o *hardware* para análise posterior pode levar à violação de privacidade de usuários, uma vez que o multi-inquilinato desse cenário faz com que uma mesma máquina física guarde informações de diversos clientes, alguns dos quais podem não estar envolvidos na investigação em curso. Diversos trabalhos na literatura tratam esse problema adequadamente, por meio das duas estratégias principais: a primeira, adotada em [Reichert et al. 2015, George et al. 2012, Poisel et al. 2013, Dykstra and Sherman 2013], consiste em coletar dados pertinentes à investigação e armazená-los fora da nuvem; a segunda, empregada em [Sang 2013] e que constitui um caso específico de [George et al. 2012], depende da cooperação do provedor de serviços de nuvem para conseguir as informações necessárias à investigação. Depender do provedor de serviços de nuvem é uma estratégia pouco recomendada, entretanto, pois (1) o volume de

dados de usuários pode forçar os provedores a limitar o tamanho dos *logs* armazenados, e (2) caso ocorra uma indisponibilidade causada por um ataque, o objetivo do provedor será o de restabelecer o serviço, não necessariamente o de preservar evidências[Clarke et al.].

4. Solução proposta: Dizang

A presente proposta tem como objetivo principal coletar memória de recursos computacionais virtuais de modo a conseguir: (1) identificar a fonte da evidência, mesmo se o recurso virtual não existir mais; (2) descrever o sistema antes e depois do incidente; (3) transportar e armazenar a memória coletada de uma forma que garanta sua integridade e confidencialidade; e (4) não violar a jurisdição e a privacidade de outros usuários que porventura tenham recursos alocados no mesmo servidor físico. A solução aqui apresentada, denominada Dizang, é descrita em detalhes a seguir.

4.1. Descrição

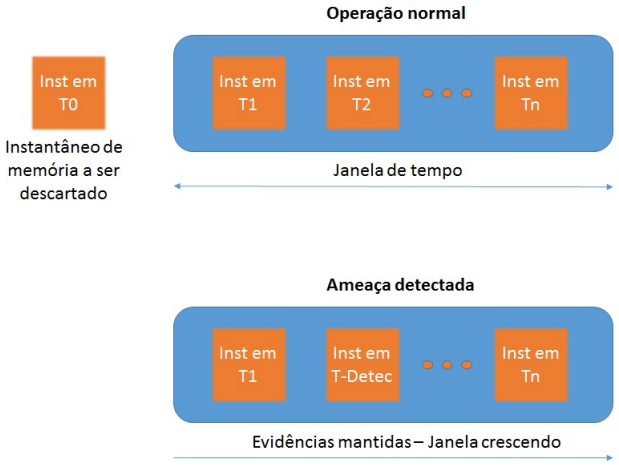
Em sistemas computacionais executados sobre uma infraestrutura física (i.e., não virtualizada), pode-se fazer uma associação direta entre um recurso qualquer, como uma informação da memória, imagem de disco ou pacotes trafegando na rede, e sua origem correspondente. Já em sistemas construídos sobre uma infraestrutura virtual, em especial quando esta é auto-escalável, os recursos computacionais são altamente voláteis e, portanto, podem ser desalocados a qualquer momento. Para conseguir correlacionar uma evidência a sua origem volátil, é necessário utilizar outro elemento em que persista a relação fonte-evidência, o que na presente proposta é feito por meio de contêineres linux (LXC). Embora um contêiner seja um *software* e, portanto, também volátil, cada imagem compilada e sua execução na forma de contêiner são normalmente atrelados a um *hash* que identifica univocamente essa relação. O contêiner também permite identificar com mais precisão a fonte de uma evidência, uma vez que é possível dividir as partes de um sistema em contêineres: por exemplo, um contêiner para o motor de páginas dinâmicas (e.g., Apache [Apache 2016b]), outro com a lógica de negócios (e.g., *golang* [Google]) e um terceiro para um banco de dados (e.g., *Cassandra* [Apache 2016a]).

A cópia de memória não é uma atividade atômica, pois ela é executada em conjunto com outros processos. Portanto, caso um desses processos seja um código malicioso apagando traços de sua existência da memória do contêiner, informações possivelmente importantes para a investigação podem acabar sendo perdidas. Com o objetivo de deixar o processo de cópia da memória mais atômico, Dizanginterrompe temporariamente a execução do contêiner, realiza a cópia de sua memória, e em seguida retoma sua execução. Essa técnica, que é semelhante àquela adotada em [Rafique and Khan 2013] para máquinas virtuais, produz um instantâneo da memória volátil do contêiner; isso permite sua análise em um estado de repouso, ou seja, sem a necessidade de ter o contêiner em execução. Ao realizar a coleta em intervalos de tempo adequados, é possível construir um histórico do estado da memória durante a execução no contêiner.

A maioria das técnicas forenses mais usadas atualmente são voltadas à obtenção da informação em sua totalidade, seja via cópia bit a bit, seja por meio da obtenção do *hardware* físico [Simou et al. 2014]. Embora tais técnicas possam parecer interessantes à primeira vista, elas muitas vezes acabam sendo responsáveis por um problema: o crescente volume de informações que os investigadores precisam analisar

[Quick and Choo 2014]. Para mitigar essa dificuldade, em Dizangsão adotadas duas estratégias: a primeira é a definição de um volume de dados que possa ser considerado *suficiente* para a realização de uma investigação; a segunda é a definição de uma *idade máxima* para a evidência enquanto o sistema trabalha em condições normais, isto é, quando não está sob ataque. Para detectar e analisar intrusões na memória de processos, é necessário ter uma cópia da memória antes e depois da intrusão [Case et al. 2014]. Assim, a solução proposta implementa uma janela de instantâneos de memória cobrindo um intervalo de tempo pré-definido, como ilustrado na Figura 1. Em condições normais de operação, as evidências são coletadas com certa periodicidade e coletas que atingem uma determinada idade são descartadas. Em contraste, após a detecção de um evento de ataque (e.g., por um sistema de detecção de intrusões), Dizangdeixa de descartar as coletas mais antigas do *log* de monitoramento, sendo possível conhecer o sistema antes e depois do ataque e, assim, avaliar sua evolução.

Figure 1. Janela deslizante de coleta de evidência

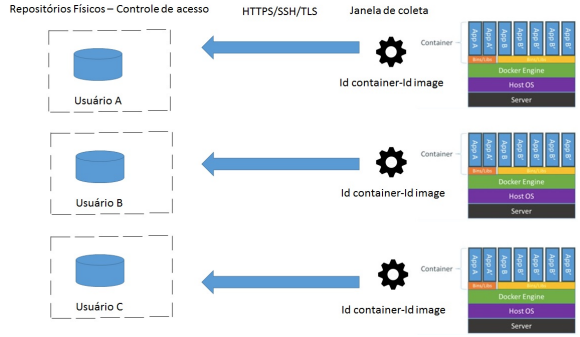


Para persistir a relação evidência-origem e garantir a integridade da mesma, a presente proposta calcula o hash do par [evidência, identificador da imagem do contêiner] e armazena a tripla [hash, identificador da imagem do contêiner, evidência]. Para evitar eventuais problemas com o armazenamento desses dados em países com jurisdições diferentes daquelas que devem ser aplicadas na investigação em questão, as evidências coletadas são armazenadas em um local físico fora da nuvem, após serem transportadas por meio de um canal seguro (e.g., usando o protocolo *Transport Layer Security* – TLS [Dierks T 2008]).

4.2. Implementação

Os métodos propostos foram implementados em uma plataforma de testes visando avaliar a eficácia de Dizangem coletar as informações de memória dos contêineres de forma reprodutível, garantindo sua confidencialidade e integridade sem violar jurisdições ou a privacidade de usuários. A solução, ilustrada na Figura 2, consistiu na criação de 1 máquina virtual usando o Oracle Virtual Box 5.0 [Oracle] em um notebook Intel i5 de 2.30Mhz e 4Gb de RAM com sistema operacional de 64 bits. Essas máquinas virtuais possuem 2 Gb de memória RAM e emulam apenas 1 processador, e em cada uma delas foi instalado o Docker Engine 1.10 [Docker Inc] e a API Docker 1.21 [Docker Inc],

Figure 2. Desenho completo da solução Dizang



com os quais foram criados 3 contêineres executando o nginx 1.0 [Igor] em diferentes portas. Usando uma aplicação Java que descobre o identificador de processo associado a cada contêiner, pode-se copiar conteúdo do *descritor de alocação de memória não uniforme* (**/proc/pid/numa_maps**), o qual contém a alocação das páginas de memória, os nós que estão associados a essas páginas, o que está alocado e suas respectivas políticas de acesso[Unix Man Pages]. A cópia e gravação desse arquivo acontece da seguinte forma: a cada minuto, a aplicação (1) pausa o contêiner em questão, (2) copia a diretório **numa_maps**, (3) concatena os dados obtidos com o *hash* de identificação da imagem do contêiner, (4) calcula o *hash* do conjunto e (5) salva o resultado em um arquivo cujo nome é o identificador da imagem do contêiner e a extensão é **.mem**. Após a conclusão do processo de cópia, a mesma aplicação verifica se existem arquivos **.mem** em disco mais antigos do que um certo intervalo de tempo "t", descartando-os.

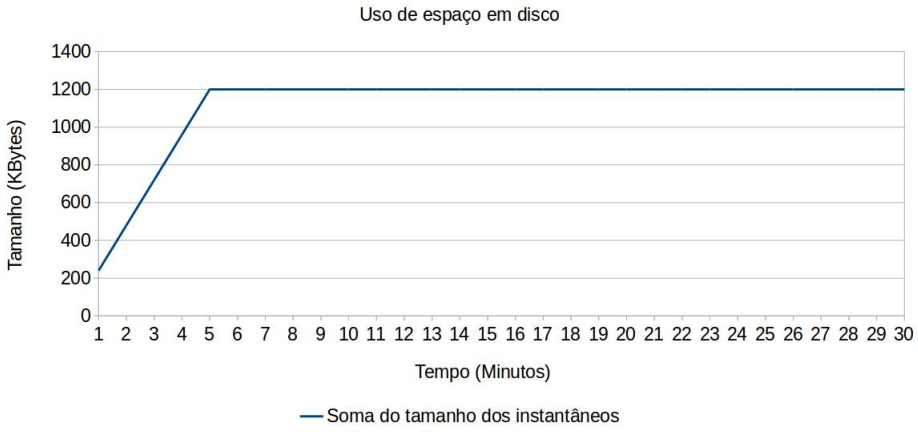
4.3. Resultados experimentais

Para avaliar a efetividade de Dizangna coleta de evidências, alguns experimentos foram realizados usando o ambiente implementado conforme descrito na Seção 4.2. Primeiramente, o sistema foi configurado para realizar coletas de memória em intervalos de 1 minuto, salvá-las em disco externo e apagar amostras coletadas há mais de 5 minutos. O sistema foi então executado por 30 minutos, tempo durante o qual foram coletadas como métricas (1) o uso de espaço em disco utilizado pelos instantâneos de memória salvos e (2) o tempo de pausa no contêiner necessário para a cópia das mesmas. A cada coleta, foi executado o comando `du -sh *.mem` do *Unix* no disco de armazenamento externo, para retornar a lista dos arquivos onde os instantâneos de memória foram armazenados e o espaço em disco ocupado pelos mesmos. Ao fim do experimento, os contêineres foram removidos.

A ocupação em disco devido aos instantâneos de memória capturados durante o experimento é mostrada no gráfico da Figura 3. O gráfico mostra que o aumento do uso do espaço em disco é linear e o crescimento se interrompe quando é atingido o limite de tempo configurado para a janela, pois as coletas com tempo de vida maior que tal limite são apagadas do disco. Pode-se concluir, portanto, que a solução mantém sob controle o espaço em disco ocupado pelas amostras coletadas.

A figura 4, por sua vez, mostra uma listagem de alguns dos instantâneos de memória salvos pela solução depois que os contêineres são removidos. Nela pode-se ver que as coletas continuaram no disco da máquina mesmo após a remoção dos contêineres.

Figure 3. Evolução do uso do espaço em disco com o Dizang.



Usando o identificador do contêiner e da imagem, consegue-se associar a evidência a sua origem (i.e., a imagem e o contêiner), conforme esperado para uma análise forense. Essa capacidade se mantém após a detecção de uma ameaça, pois nesse caso coletas mais antigas deixam de ser apagadas. Assim, é possível descrever o estado do sistema antes e depois do incidente [Case et al. 2014], permitindo-se, por exemplo, que ataques de injeção de código em memória sejam analisados.

Figure 4. Lista de instantâneos de memória.

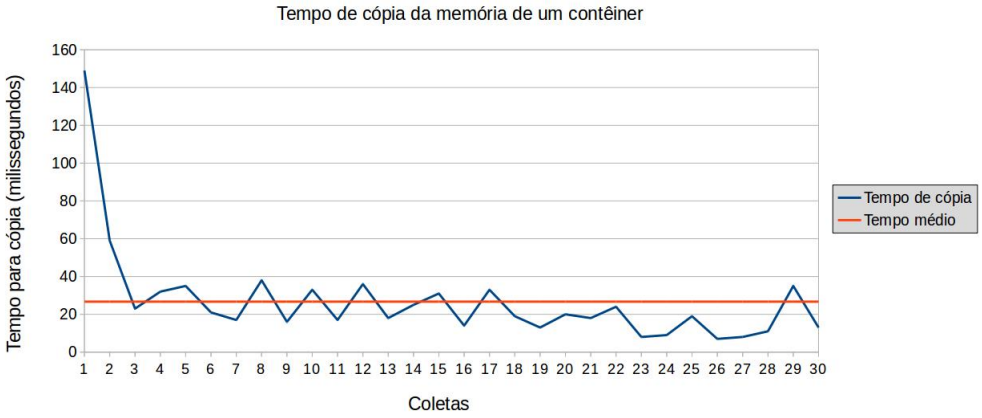
```
16K     4bd952884935d80421133400130290429778acc85df0ed7366e23a9d19425d1d-8fa80c6dba11002f45c835254343bce274fa27e1136708a0e4cb13ecf57d6b53-3412-30_07_2017_10_20.mem
16K     4bd952884935d80421133400130290429778acc85df0ed7366e23a9d19425d1d-8fa80c6dba11002f45c835254343bce274fa27e1136708a0e4cb13ecf57d6b53-3413-30_07_2017_10_19.mem
16K     4bd952884935d80421133400130290429778acc85df0ed7366e23a9d19425d1d-8fa80c6dba11002f45c835254343bce274fa27e1136708a0e4cb13ecf57d6b53-3413-30_07_2017_10_20.mem
16K     4bd952884935d80421133400130290429778acc85df0ed7366e23a9d19425d1d-8fa80c6dba11002f45c835254343bce274fa27e1136708a0e4cb13ecf57d6b53-3414-30_07_2017_10_19.mem
16K     4bd952884935d80421133400130290429778acc85df0ed7366e23a9d19425d1d-8fa80c6dba11002f45c835254343bce274fa27e1136708a0e4cb13ecf57d6b53-3414-30_07_2017_10_20.mem
16K     6f7e69cc438812334817a9211236b36c3b71b0e8dd606046631ee1c8625e142d-8fa80c6dba11002f45c835254343bce274fa27e1136708a0e4cb13ecf57d6b53-3318-30_07_2017_10_19.mem
16K     6f7e69cc438812334817a9211236b36c3b71b0e8dd606046631ee1c8625e142d-8fa80c6dba11002f45c835254343bce274fa27e1136708a0e4cb13ecf57d6b53-3318-30_07_2017_10_20.mem
16K     6f7e69cc438812334817a9211236b36c3b71b0e8dd606046631ee1c8625e142d-8fa80c6dba11002f45c835254343bce274fa27e1136708a0e4cb13ecf57d6b53-3339-30_07_2017_10_19.mem
16K     6f7e69cc438812334817a9211236b36c3b71b0e8dd606046631ee1c8625e142d-8fa80c6dba11002f45c835254343bce274fa27e1136708a0e4cb13ecf57d6b53-3339-30_07_2017_10_20.mem
16K     6f7e69cc438812334817a9211236b36c3b71b0e8dd606046631ee1c8625e142d-8fa80c6dba11002f45c835254343bce274fa27e1136708a0e4cb13ecf57d6b53-3340-30_07_2017_10_19.mem
```

Uma potencial limitação da solução proposta é que a pausa de um contêiner para coleta de dados poder, em princípio, causar perdas no desempenho da aplicação sendo executada. Para avaliar esse impacto, durante o experimento foram medidos os tempos de cópia da memória do contêiner. Os resultados são mostrados no gráfico da Figura 5. É possível notar que, após a inicialização da aplicação, o tempo para realizar a cópia é bastante reduzido, variando entre 20 e 40 milisegundos. Em especial, para contêineres executando um motor de páginas web dinâmicas, como é o caso do experimento em questão, essa latência deve ser pouco perceptível por usuários finais.

4.4. Limitações

Como a solução descrita tem como foco coletar informações de memória no espaço do usuário (*user space*), ela não consegue acessar o espaço de kernel (*kernel space*). Assim, Dizangem princípio não provê suporte a técnicas de investigação de malware que se baseiam em informações do *kernel space*, como, por exemplo, a comparação de informações do bloco do ambiente do processo (*Process Environment Block – PEB*), que ficam no *user space*, com informações do descritor de endereços de memória virtual (*Virtual Address Descriptor – VAD*), que fica no *kernel space*. Análise de ameaças que realizam

Figure 5. Lista de instantâneos de memória.



manipulação direta dos objetos do kernel (D.K.O.M.– *Direct Kernel Object Manipulation*) também não se beneficiam com a solução aqui proposta.

5. Considerações Finais

Ameaças digitais que atuam diretamente na memória de sistema não costumam deixar rastros em disco após terem os recursos correspondentes desalocados, dificultando análises forenses posteriores. Esse problema é especialmente notável em sistemas de computação em nuvem, nos quais a alocação e desalocação de recursos virtualizados (e.g., máquinas virtuais e contêineres) é frequente. Essa característica, aliada a aspectos como multi-inquilinato e multi-jurisdição de nuvens computacionais, dificulta a coleta de evidências para a investigação de incidentes.

Nesse cenário, a proposta apresentada visa relacionar o instantâneo de memória a sua origem, utilizando o *hash* calculado da imagem do contêiner como identificador da evidência armazenada. Para evitar uso excessivo de memória, a quantidade de dados armazenados usa uma janela de armazenamento, o que permite descrever a memória antes e depois de um ataque (e.g., de injeção de memória). Combinada com uma ferramenta para identificação de ameaças, essas características de Dizango transformam em uma solução poderosa para prover evidências e, assim, viabilizar análises forenses na nuvem.

References

Apache (2016a). Apache-Cassandra.

Apache (2016b). Apache-Tomcat.

Case, A., Ligh, M., Jamie, L., and Walters, A. (2014). *The Art of Memory Forensics: Detecting malware and threats in Windows, Linux and Mac memory*. Wiley, kindle edition.

Clarke, N. L., Reich, C., Saad, A., and Furnell, S. Cloud Forensics : A Review of Challenges , Solutions and Open Problems, year = 2015. (April):1–9.

Dezfouli, F. N., Dehghantanha, A., Mahmoud, R., Binti Mohd Sani, N. F., and Bin Shamsuddin, S. (2012). Volatile memory acquisition using backup for forensic investigation.

Proc. of Intnl. Conf. on Cyber Security, Cyber Warfare and Digital Forensic, pages 186–189.

Dierks T, R. E. (2008). The Transport Layer Security (TLS) Protocol.

Docker Inc. Docker.

Dykstra, J. and Sherman, A. T. (2012). Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. *Digital Investigation*, 9:S90–S98.

Dykstra, J. and Sherman, A. T. (2013). Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform. *Digital Investigation*, 10:S87–S95.

Fewer, B. S. (2008). Reflective DLL Injection. (October).

George, S., Venter, H., and Thomas, F. (2012). Digital Forensic Framework for a Cloud Environment. In Cunningham, P. and Cunningham, M., editors, *IST Africa 2012*, pages 1–8, Tanzania. Internation Information Management Corporation.

Google. Golang.

Igor, S. NginX.

Keyun, R., Joe, C., Tahar, K., and Mark, C. (2011). *Advances in Digital Forensics IV*, volume 1. Orlando, 7 edition.

Linuxcontainers.org (2015). Linux Containers (LXC).

Mell, P. and Grance, T. (2011). The NIST definition of cloud computing. *NIST Special Publication*, 145:7.

Miller, M. and Turkulainen, J. (2004). Remote Library Injection. pages 1–40.

Morsy, A. M., Grundy, J., and Muller, I. (2010). An Analysis of the Cloud Computing Security Problem. In *APSEC Cloud Workshop*, Sydney, Australia.

Oracle. Virtual Box 5.1.

Piraghaj, S. F., Dastjerdi, A. V., Calheiros, R. N., and Buyya, R. (2016). A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers. *Proc. - IEEE International Conference on Data Science and Data Intensive Systems*, pages 368–375.

Poisel, R., Malzer, E., and Tjoa, S. (2013). Evidence and cloud computing: The virtual machine introspection approach. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 4(1):135–152.

Quick, D. and Choo, K. K. R. (2014). Impacts of increasing volume of digital forensic data: A survey and future research challenges. *Digital Investigation*, 11(4):273–294.

Rafique, M. and Khan, M. N. A. (2013). Exploring Static and Live Digital Forensics: Methods, Practices and Tools. *International Journal of Scientific & Engineering Research*, 4(10):1048–1056.

Rahman, S. and Khan, M. N. A. (2015). Review of live forensic analysis techniques. *International Journal of Hybrid Information Technology*, 8(2):379–388.

Reichert, Z., Richards, K., and Yoshigoe, K. (2015). Automated forensic data acquisition in the cloud. *Proc. - 11th IEEE MASS*, pages 725–730.

Sang, T. (2013). A log-based approach to make digital forensics easier on cloud computing. *Proc of the 2013 3rd ISDEA*, pages 91–94.

Simou, S., Kalloniatis, C., Kavakli, E., and Gritzalis, S. (2014). Cloud forensics: Identifying the major issues and challenges. *Lecture Notes in Computer Science*, 8484 LNCS:271–284.

Unix Man Pages. Numa Maps - Non Uniform Memory Architecture.