REsquared – Detecção e Recuperação de Intrusão com uso de Controle de Versão

Gabriel Dieterich Cavalcante, Paulo Lício de Geus

[1]Laboratório de Administração e Segurança – LAS
Instituto de Computação – IC
Universidade Estadual de Campinas – UNICAMP
Av. Albert Einstein, 1251 - CEP 13083-852 – Campinas – SP – Brasil

{gabriel,paulo}@las.ic.unicamp.br

Abstract. *Current computer systems have a huge number of configurations that are hard to manage. The combinations of system configurations can impact on performance and behavior. From the moment that a system stops working correctly it is remarkable that something has changed. That is in common in software development, where changes made by the programmer may result in some features no longer working or the project not compiling anymore. Revision control systems can recover a previous state of the source code through revision mechanisms. Integrity checking is used to catch file modifications, however this technique does nothing toward recovering those files. This study proposes and implements an integrated architecture that combines integrity checking and restoring mechanisms. Tests were executed in order to measure the load imposed by the solution. In addition, analysis of two case studies shows the efficiency of the adopted solution.*

Resumo. *Sistemas atuais possuem um elevado número de configurações que tornam a gerência dos sistemas uma tarefa árdua. As combinações entre diferentes configurações podem impactar no desempenho e comportamento do sistema. Quando um sistema para de funcionar corretamente é evidente que algo mudou. Este é um cenário comum no desenvolvimento de software, onde mudanças no código podem acarretar mal funcionamento do programa ou que mesmo deixe de compilar. Sistemas de Controle de Versão podem recuperar o código fonte para um estado anterior através de mecanismos de revisão. A Verificação de Integridade é usada para detectar alterações em arquivos, entretanto não oferece suporte para a recuperação destes arquivos. Este artigo descreve uma implementação de um sistema integrado, que combina a verificação de integridade e mecanismos de restauração. Testes executados demonstram a carga extra imposta pelo suporte de restauração em sistemas de verificação de integridade, dois estudos de caso demonstram a eficácia deste método.*

1. Introdução

Quando sistemas são comprometidos uma das mais árduas tarefas é a recuperação, pois tipicamente envolve a instalação de uma nova imagem de sistema e seus aplicativos, instalação de correções para as vulnerabilidades conhecidas e recuperação de dados importantes corrompidos. Cada passo é manual e tedioso, tomando considerável quantidade de

tempo dos administradores. Mesmo após todos estes passos, ainda é necessário certificar-se de que não existe nenhum código estranho embutido em algum arquivo executável do sistema, pois esse código pode comprometer o sistema a longo prazo ou ainda o sistema recém instalado.

Muitos problemas são causados pelas configurações presentes em cada sistema. A perda de algumas configurações ou incoerência das mesmas pode acarretar seu mal funcionamento, ou até sua total queda. A principal função das configurações é prover uma vasta quantidade de recursos, naturalmente partilhados por várias aplicações, dentre os quais: fornecer meios para que aplicações instaladas em momentos diferentes possam interagir e se integrar umas com as outras; conter mecanismos para que aplicações sejam registradas em serviços base do sistema para utilizarem suas funcionalidades; permitir que pequenos componentes possam se acoplar a algum aplicativo para estender suas funcionalidades.

A trajetória de solução de problemas de sistemas frequentemente é acompanhada da afirmação: "funcionava ontem e não está funcionando hoje". Dito isso, é de se esperar que algo tenha mudado no sistema para ele deixar de funcionar. Encontrar o que foi alterado é de fato, o princípio da solução. Se de alguma forma, o sistema fosse preparado para que pudesse retornar a um estado anterior, seria economizada uma quantidade considerável de tempo.

Este tipo de histórico é comum em ferramentas de controle de versão. Ferramentas desse tipo são usadas no processo de desenvolvimento de software, elas são responsáveis pelo acesso paralelo dos programadores ao código e guardam eficientemente a evolução do mesmo. Existe uma funcionalidade presente nestes versionadores que pode solucionar o caso da resiliência para configurações de sistemas operacionais, a revisão de código. A revisão de código permite que seja extraída uma versão do código baseando-se em uma data anterior, revertendo-o para um outro estado.

Ferramentas de Verificação de integridade buscam diferenças nos arquivos presentes em um sistema de arquivos, com o intuito de checar a integridade dos mesmos. Esta é uma maneira útil de buscar alterações que podem ter sido feitas por um ataque bem sucedido. Estes ataques normalmente visam implantar uma brecha permanente no sistema para viabilizar ataques posteriores.

Entretanto, as ferramentas de Verificação de integridade não possuem mecanismos de recuperação eficientes. Fazendo com que os administradores sejam reféns de conhecidos e bem entendidos sistemas de *backup*. Este artigo apresenta uma solução de recuperação para sistemas de verificação de Integridade, tendo com principal benefício a pequena quantidade de dados usados para prover a recuperação de sistemas operacionais.

O restante do artigo está organizado da seguinte maneira: na Seção 2 está contextualizado o ambiente de Controle de Versão; a Seção 3 apresenta alguns sistemas de Detecção de Intrusão relacionados com este trabalho; na Seção 4 é mostrada a arquitetura de *REsquared*; a Seção 5 comenta a respeito dos resultados obtidos e finalmente a Seção 6 apresenta uma pequena conclusão sobre tudo que foi desenvolvido ao longo deste trabalho.

2. Controle de Versão

Sistemas de Controle de Versão—*Version Control Systems*(VCSs)—são ferramentas utilizadas para automatizar o Controle de Versão sob documentos, códigos fonte e quaisquer arquivos utilizados dentro de um determinado ambiente. Com um VCS é possível gerenciar o histórico de um arquivo e obter simultaneamente duas ou mais versões do mesmo[Gift and Shand 2009].

O funcionamento de um VCS consiste basicamente em manter um repositório com os arquivos que devem ser controlados, no repositório também são armazenados os arquivos de controle e *logs*. Para otimizar o espaço em disco ocupado pelas diferentes versões dos arquivos, a maior parte dos VCSs utiliza o método de compressão *Delta*[Berliner and Prisma 1990][Rochkind 1975][Tichy 1985], que consiste em armazenar apenas a diferença entre versões sucessivas de um arquivo. O princípio geral da compressão *Delta* é aplicar sucessivamente as diferenças armazenadas sobre a versão inicial do arquivo, deste modo, partindo da versão inicial pode-se obter a mais recente ou qualquer outra intermediária[Suel and Memon 2002].

A maioria dos VCSs trabalha com o modelo *check-in/check-out*[Feiler 1991] que tem base em um Controle de Versão individual para cada arquivo. Os arquivos versionados ficam armazenados dentro de um repositório e o usuário não pode interagir diretamente com eles. Primeiramente o usuário deve fazer um *check-out*—fazer a retirada—dos arquivos que ele deseja editar, ou seja, copiar os arquivos para um diretório de trabalho. Futuramente os arquivos modificados terão novas versões a partir do momento em que o usuário efetuar seu *check-in*—dar entrada—no repositório.

A principal função do VCS é guardar todo o histórico de desenvolvimento do documento, desde o primeiro *check-in* até sua última versão. Isso permite que seja possível recuperar uma determinada versão de qualquer data mais antiga. Existem diversos VCSs disponíveis gratuitamente e soluções comerciais. Dentre os livres podemos citar como mais famosos: GIT[Wiegley 2008], Bazaar[Wheeler 2008], SubVersion[Collin-sussman et al. 2007] e Mercurial[O'Sullivan 2007].

3. Detecção de Intrusão

Os computadores atuais estão sujeitos a vulnerabilidades de variados tipos. Aplicações—inclusive aquelas que são responsáveis pelas políticas de segurança—e sistemas operacionais possuem falhas em muitas de suas camadas, o que impossibilita a implementação de políticas de segurança perfeitas. Melhorias no nível de segurança são possíveis com o uso de sistemas de detecção de intrusão—*Intrusion Detection Systems*(IDSs). Esta abordagem baseia-se na hipótese de que um sistema pode não ser completamente seguro, porém violações de segurança podem ser captadas pelo monitoramento e análise de seu comportamento.

Há diferentes maneiras de monitorar o comportamento de um sistema. Os dois primeiros tipos de IDS foram: o baseado em *host*—*Host-Based Intrusion Detection System (HIDS)*—e o baseado em rede—*Network-Based Intrusion Detection System (NIDS)*—a evolução fez com que surgissem novas alternativas como: o IDS híbrido—*Hybrid* IDS—que aproveita as melhores características das duas abordagens; finalmente, como complemento dos HIDS e NIDS surgiu o *storage-based* IDS, que utiliza dados armazenados

separadamente do sistema para realizar verificações mesmo que o *host* esteja comprometido.

3.1. *Host-Based Intrusion Detection System* – HIDS

Um sistema de detecção de intrusão baseado em *host* monitora qualquer comportamento ou estado dinâmico de uma máquina. Para isto, utiliza em sua análise arquivos de *log* ou a saída de agentes de auditoria, podendo assim avaliar alterações no sistema de arquivos, modificações no controle de acesso de usuários, comportamento de processos do sistema, uso de recursos entre outros aspectos[Shah 2001].

Uma outra característica presente em muitos *HIDS* é o *hash* do sistema de arquivos. Através de uma lista de arquivos importantes, o *HIDS* cria uma base de dados inicial com valores *hash* dos arquivos desejados, posteriormente recalcula os valores de hash e compara com os dados armazenados. Se alguma comparação entre os *hashes* não for bem sucedida alguma alteração ocorreu. Esta estratégia é bem eficiente para detectar instalação de *rootkits*, adição de novos usuários ou alteração de configurações.

3.1.1. Verificação de Integridade

A integridade é uma das chaves principais para confiabilidade, ou seja, se um arquivo é íntegro, podemos confiar no seu conteúdo. A partir daí, pode-se concluir que um sistema é íntegro se todos os seus arquivos também são. O sistema de arquivos está presente na maioria dos sistemas operacionais; nele estão contidos dados do usuário, arquivos executáveis, arquivos de configuração e informações de controle de acesso. Usualmente, também está contida a base de executáveis do próprio sistema operacional.

Uma classe de ataques denominada ataque de integridade, que tem o intuito de alterar arquivos sem detecção. O atacante se aproveita de um sistema vulnerável—muitas vezes temporariamente vulnerável—alterando arquivos para garantir sua reentrada ou para simplesmente comprometer o sistema[Kim and Spafford 1994].

A medida que a complexidade dos sistemas operacionais foi crescendo, os administradores de sistema foram deixando de conseguir gerenciar mudanças indesejadas no sistemas de arquivos, devido a quantidade de arquivos e ao número elevado de dependências entre aplicações e bibliotecas. Isto aumentou a demanda de ferramentas com objetivo de assegurar a integridade dos arquivos de um sistema, e assim buscar ameaças instaladas no sistema ou simplesmente guardar uma configuração padrão[Kim and Spafford 1994]. Surgiram então ferramentas utilizando diferentes mecanismos para flagrar mudanças indesejadas no sistema de arquivos. Em geral são baseadas no *hash* de arquivos e existem algumas práticas de assinatura de arquivos que tem funcionamento semelhante.

Tripwire[Kim and Spafford 1994] baseia-se em um arquivo de configuração que define de quais arquivos ele irá manter informações, além de algumas políticas de análise para cada arquivo. Por exemplo, guardar informações de metadados para *logs*, já que são arquivos que estão em constante crescimento. Para este caso somente são analisados o tamanho do arquivo—ele deve crescer, nunca diminuir—e dados do controle de acesso ao arquivo. Em 2002 os criadores desta ferramenta abriram uma empresa e pararam de desenvolver ela abertamente, tornando-a proprietária. Aide[Reed 2003] é considerada a

continuação de Tripwire na comunidade de software livre. Aide adicionou *daemons* de monitoramento com uso de *threads*, porém as características gerais de uso são as mesmas. Afick é uma ferramenta de securança com *design* muito próximo ao de Tripwire, um dos seus diferenciais é a possível instalação sistemas *Microsoft Windows*[Afick 2010].

Um dos problemas críticos desta abordagem é a frequência das análises. Certamente alterações somente serão detectadas no momento que um arquivo alterado for examinado. O problema está na intermitência das verificações, ou seja, se um arquivo for infectado e este for utilizado antes da próxima análise. Dependendo da periodicidade das análise pode haver um grande tempo livre para ataques, que podem causar sérios danos aos dados do usuário.

Além disso, todas as ferramentas desta categoria realizam massivo uso do disco durante suas operações, o que pode causar profundo impacto no *host*. Isto ocorre devido a necessidade de se realizar *hashes* dos arquivos analisados, o que requer leitura de todo ele. Geralmente, as verificações nos servidores, são agendadas em horários onde o número de usuários dependentes do serviço é menor.

Samhain[Samhain 2010] é uma ferramenta aberta multiplataforma com intuito de centralizar a verificação de integridade. Foi projetada para monitorar centralizadamente múltiplos *hosts* com diferentes sistemas operacionais instalados. Também pode ser usada como aplicação simples em uma só maquina. Como diferencial, Samhain possui bases de dados e comunicação com os clientes criptografada, além de um modo "invisível" de operação, como contra-medida a possíveis ataques sobre seus arquivos e operação. Osiris[Osiris] é um sistema de monitoramento de integridade que realiza análises periódicas de um ou mais *hosts*. Ele mantém *logs* das mudanças no sistema de arquivos, usuários, grupos e módulos de *kernel* instalados. Osiris pode ser configurado para enviar esses *logs* por *e-mail* e, se desejado, ele os mantém de modo a suportar análises forenses sobre eles. Por sua vez, Radmind[Craig and McNeal 2003] consiste em um conjunto de comandos *UNIX* e um gerenciador projetado para disparar verificações em várias outras máquinas cliente.

3.2. Storage-based IDS

Essa classe de IDS consiste em monitorar características das modificações no sistema de arquivos, ou seja, monitorar os pedidos de escrita de dados enviados pelo sistema operacional ao sistema de armazenamento de dados. Isso permite que o IDS possa identificar várias ações comuns de um atacante como instalar *backdoors*, inserir cavalos-de-tróia no sistema, adulterar ou excluir arquivos de *log*. Todo sistema de armazenamento consegue enxergar as operações que serão realizadas em dados persistentes, permitindo transparente análise de mudanças indesejáveis e gerar alertas para cada cliente do sistema de armazenamento—alguns sistemas de armazenamento via rede possuem vários clientes como os de [Banikazemi et al. 2005] e [Strunk et al. 2002].

4. REsquared – REsilient REcovery

Esta seção descreve a implementação de um sistema integrado, que combina a Verificação de Integridade e mecanismos de restauração presentes em Sistemas de Controle de Versão. Para isto foi utilizada a detecção de intrusão baseada em integridade de arquivos e ferramentas de Controle de Versão de arquivos e diretórios para armazenar os diferentes

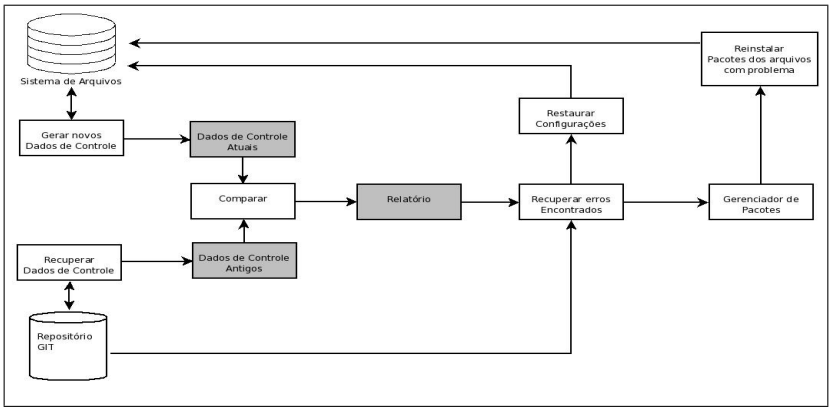

Figura 1. Diagrama de Fluxo Geral de Operação.

estados de configuração do sistema, com objetivo final de criar um sistema operacional *GNU/Linux* resiliente. *REsquared* foi projetado de maneira inteiramente modular, o que garante alta portabilidade.

A Figura 1 exibe o diagrama de fluxo geral de operação. A figura mostra duas entradas, o sistema de arquivos e um repositório. Inicialmente o repositório é gerado com informações coletadas dos sistema de arquivos quando este é considerado íntegro, no repositório também são guardados os arquivos de configuração encontrados—arquivos de texto. Cada arquivo sob observação—binários e texto—de *REsquared* possui um descritor de integridade dentro do repositório, este descritor contêm todas as informações sobre seus metadados e *hashes*.

Após esta inicialização, a cada verificação são calculados novos dados de controle do sistema de arquivos, esse dados são comparados com os armazenados no repositório. Dessa forma, podem ser encontradas evidências de alteração. O relatório possui todas essas informações de forma clara para o usuário, e serve de base para o recuperador tomar decisões sobre a correção do sistema de arquivos.

No repositório ficam armazenadas somente configurações, que é a parte "customizável" dos sistemas operacionais. Todos os executáveis—programas—podem ser obtidos em fontes públicas através de um gerenciador de pacotes, que é peça fundamental em toda distribuição *Linux* atual.

O gerenciador de pacotes é o responsável por reinstalar os arquivos binários possivelmente comprometidos. Após a reinstalação são restauradas as configurações para que sistema volte a sua normalidade. Além disso, o gerenciador de pacotes também permite uma rápida maneira de criação de arquivos de controle para um determinado pacote. Desta maneira o usuário pode manter sob controle um aplicativo apenas sabendo qual pacote ele pertence, garantindo sua integridade completa.

A portabilidade referente ao uso de diferentes controladores de versão envolve apenas um módulo, que é o grande responsável por todos os acessos a arquivos e pela comunicação externa. A manipulação dos descritores de integridade também é feita por este módulo, pois os mesmos serão salvos no repositório. Este módulo é responsável pelo fornecimento dos descritores a todos os outros módulos.

A arquitetura apresentada na Figura 2 define um módulo responsável pela comu-

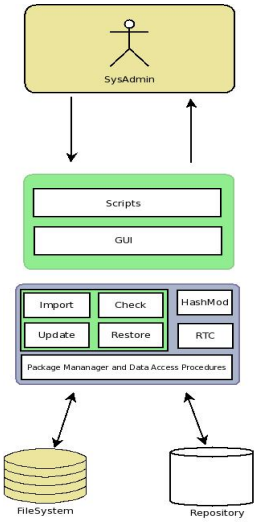

Figura 2. Arquitetura de *REsquared*.

nicação com o sistema de arquivos, com o repositório e com o gerenciador de pacotes. Logo, portar a ferramenta para várias distribuições envolve apenas codificação neste módulo. Na segunda camada da arquitetura encontram-se duas possibilidades distintas de comunicação com os módulos implementados. Os *scripts* seriam ferramentas de linha de comando, úteis em ambientes de servidor onde não há recursividade gráfica. Outra implementação voltada ao usuário comum seria uma interface gráfica, almejando maior nível de usabilidade.

O módulo RTC— Runtime Tree Calculator— é responsável por criar a árvore de dependências de execução—*runtime dependency*—de arquivos binários. A integridade das ações de um aplicativo só é garantida com uso bibliotecas e arquivos de configuração íntegros. Um executável pode possuir diferentes tipos de dependência: arquivos de configuração, arquivos temporários, arquivos de *log*, executáveis auxiliares, bibliotecas e módulos carregáveis do *kernel*. RTC inclui todas as dependências de um aplicativo sob controle de *REsquared*, almejando garantir a integridade de suas ações, e não somente a integridade do arquivo.Este módulo também é responsável por classificar os arquivos para o módulo *import* tomar decisões para cada arquivo, armazenar conteúdo e metadados caso texto, somente metadados de binários.

O descritor de integridade contém todas as informações possíveis a respeito de um arquivo ou diretório. Posteriormente informações retiradas do sistemas de arquivos pelo módulo *Check* serão comparadas com as armazenadas nos descritores dentro do repositório. A verificação possui três níveis de profundidade, o primeiro checa apenas a existência do arquivo ou diretório, o segundo compara os metadados e finalmente o terceiro compara os *hashes* dos arquivos e as informações específicas de diretórios. Para efeitos de análise forense, um quarto nível de verificação pode ser considerado: a auditoria de arquivos de configuração. Com base nos arquivos texto armazenados no repositório podem ser identificadas as modificações que levaram o sistema a ter sua integridade afetada. A ordenação dos *timestamps*—criação, modificação e acesso—permite a recriação da sequência de ações do atacante.

A detecção de intrusão e seu diagnóstico fazem parte de uma boa estratégia de

segurança; a recuperação, entretanto é uma necessidade. Manter o sistema atualizado diminui a ocorrência de intrusões, mas elas inevitavelmente podem acorrer, portanto é crítico que a recuperação seja eficiente e completa. A restauração—feita pelo módulo *restore*—conterá dois passos principais, a reinstalação de pacotes e a restauração das configurações, que serão executados dependendo do problema que for encontrado no sistema de arquivos. Se um arquivo binário for comprometido, é necessária a reinstalação do pacote do qual ele faz parte. Após isso devem ser restauradas todas as configurações previamente armazenadas no repositório. Quando o problema é encontrado em um arquivo texto a recuperação torna-se trivial, apenas restaurar o arquivo que foi alterado. Salvo se esta configuração fizer parte de um serviço esteja rodando, neste caso cabe ao administrador parar o serviço para descarregar da memória possíveis configurações comprometidas que foram carregadas no arranque do serviço.

O módulo de atualização—*Update*—é acionado toda vez que o administrador decidir guardar um novo estado dos pacotes instalados no sistema. Esta atualização pode ser realizada por vários motivos: atualização de aplicativos; instalação de novos aplicativos; mudança nas configurações; inclusão de novos usuários ou grupos de acesso etc. Este módulo irá verificar primeiramente os *timestamps* dos arquivos para criar uma lista dos arquivos que necessitam ser atualizados. Após isso são adicionadas as novas versões dos arquivos texto e dos descritores de integridade no repositório.

4.1. Protótipo

Um protótipo foi construído visando validar a arquitetura proposta, o objetivo final é amadurecer o projeto para uma solução completa de resiliência para sistemas *Linux*. Em busca de portabilidade, foram criados módulos independentes com mecanismos de comunicação bem definidos, facilitando a implementação futura de uma interface gráfica destinada a usuários comuns ou de um aplicativo de linha de comando para administradores de grandes servidores.

O protótipo criado consiste em um utilitário de linha de comando *Linux*—como os tradicionais `ls`, `cd`, `grep` etc. Em geral, aplicações do modo texto são mais rápidas do que as que utilizam ambientes gráficos, pois estes por sua vez requerem maiores recursos de processamento . Por esta mesma razão, as aplicações de modo texto usam mais eficientemente os recursos de memória. Em contrapartida, ferramentas de modo texto apresentam interfaces menos amigáveis, principalmente para usuários iniciantes. Desta forma, a metodologia adotada é ideal para o uso em servidores, onde exige-se ótimo aproveitamento de recursos e administradores experientes gerenciam o ambiente.

A segurança proposta pela ferramenta depende diretamente da integridade de seus componentes e dados de controle, portanto deve-se garantir que apenas operações genuínas alterem os dados de controle do repositório. Para o uso da ferramenta é necessário a configuração de uma estrutura de segurança básica. Todo o repositório e o código dos módulos e da interface principal estarão em uma partição separada, cifrada e montada em modo leitura. Para realizar as operações de *import* e *update* a partição precisa ser montada em modo escrita, o que requer senha e intervenção do administrador. Logo após a operação a partição deve ser novamente montada em modo leitura, para evitar que alterações não permitidas sejam feitas na ferramenta e em seus dados de controle. As operações de *check* e *restore* podem ser executadas normalmente sem intervenção de montagem, pois não utilizam recursos de escrita nos dados da ferramenta.

5. Resultados e Discussão

Esta seção é composta por duas partes. Primeiramente foi realizada uma simples, porém efetiva análise da sobrecarga imposta pelo método de Controle de Versão em conjunto com a Verificação de Integridade. A sobrecarga foi calculada pela diferença do tempo médio de operação de alguns verificadores de integridade pelo tempo médio de operação da ferramenta proposta com todos os módulos ativados. Para a segunda parte de validação foram criados pequenos cenários de intrusão que tipicamente ocorrem em ambientes de produção. A ferramenta proposta foi colocada em prática em cada um deles para ser testado seu poder de detecção e recuperação.

5.1. Sobrecarga do Modelo Proposto

Algumas ferramentas de Verificação de Integridade e *REsquared* foram testadas sobre a mesma base de dados, a fim de realizar uma efetiva medição da sobrecarga imposta pelo controlador de versão acoplado ao verificador de integridade. Para isto uma outra versão de *REsquared* foi criada, com os módulos de mapeamento da árvore de dependência e de Controle de Versão desativados, ou seja, a ferramenta teve duas funcionalidades desativadas para ser limitada a operar como um verificador de integridade.

As ferramentas foram avaliadas levando em consideração dois conjuntos de arquivos diferentes, a primeira base de dados possuía em sua maioria arquivos de texto; a segunda, por sua vez possuía mais da metade composta de arquivos binários. Cada teste foi executado 30 vezes, sendo descartados os limites inferior e superior e antes de cada teste foram excluídos os *caches* de disco . Por conveniência, nesta seção será referenciada por *Resquared* a ferramenta com capacidades limitadas e por *Resquared+GIT* a ferramenta com todas as funcionalidades ativadas.

5.1.1. Base de dados composta de arquivos texto

Para esta avaliação foram considerados os arquivos do diretório `/etc`, contendo 3428 arquivos, sendo 623 binários, 904 diretórios e 1901 arquivos texto, com um tamanho total de 104 Mb. Foram realizadas operações de inicialização e verificação em todas as ferramentas, a cada segundo foram capturadas informações dos processos, que são representadas nos gráficos abaixo.

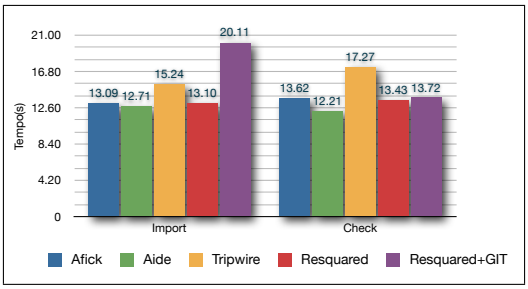

Figura 3. Tempo de execução sobre o diretório `/etc`

Na Figura 3 podemos verificar que o tempo de execução de *Resquared* é bem similar ao dos outros verificadores de integridade, tanto para operação de inicialização—*import*—quanto para a operação de verificação—*check*. A sobrecarga imposta pelas fun-

cionalidades de mapeamento de dependências e recuperação em *Resquared+GIT*, impactaram somente na execução de *import*, o que era esperado, pois junto a essa operação é criado o repositório e são copiados os arquivos texto. Na verificação apenas são considerados os dados de controle de *Resquared+GIT*, portanto a operação é bem semelhante a de *REsquared*.

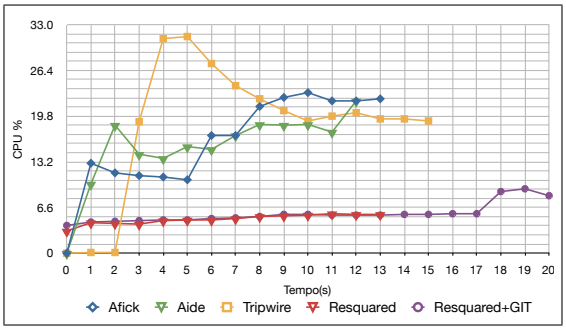

Figura 4. Porcentagem de CPU na operação de inicialização com diretório `/etc`.

A análise do consumo de CPU—Figura 4—revelou uma grande vantagem tanto de *REsquared* quanto de *Resquared+GIT* sobre as outras ferramentas, girando em torno de 6%, foi cerca de 3 vezes menor do que a média das outras ferramentas. Pode ser notado um pequeno pico no final da execução de *Resquared+GIT* que representa o momento de adição de versionamento nos arquivos texto importados.

Durante operação de verificação constatou-se novamente que o consumo de CPU de *REsquared* e de *Resquared+GIT* foi bem semelhante ao da operação de inicialização girando em torno de 6%. Todas as outras apresentaram aumento no consumo, liderado por *Tripwire*. Na verificação o consumo de CPU por parte de *REsquared* e *Resquared+GIT* foi em média 4 vezes menor do que a média das outras ferramentas.

5.1.2. Base de dados composta de arquivos binários

A base de dados foi montada com os arquivos do diretório `/usr/bin`, contendo um total de 2890 arquivos, sendo 2203 binários, 687 arquivos texto, com um tamanho total de 214 Mb. Foram realizadas operações de inicialização e verificação em todas as ferramentas. Conforme foi feito na análise anterior a cada segundo foram coletadas informações do processo.

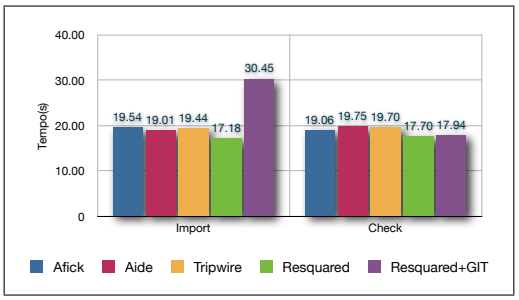

Figura 5. Tempo de execução sobre o diretório `/usr/bin`

O tempo de inicialização de *Resquared+GIT* foi aproximadamente 62% maior do que a média das outras ferramentas, somado a sobrecarga atribuída à cópia dos arquivos texto, *Resquared+GIT* também sofre este aumento devido ao cálculo da compressão delta e dos arquivos de controle do GIT. O tempo de verificação de *Resquared+GIT* ficou próximo da média, apenas o carregamento dos descritores de arquivos é feita durante a verificação, portanto nenhuma operação específica do repositório é necessária.

Tanto *REsquared* quanto *Resquared+GIT* mostraram-se novamente mais eficientes no uso de CPU do que todas as outras ferramentas, tendo consumo médio em torno de 5,2%. A Figura 6 revela consumo médio em volta de 6 vezes maior de *Afick* e *Tripwire*. *Aide* também teve consumo mais elevado, em média um consumo 4 vezes maior. A *performance* de *REsquared* e *Resquared+GIT* na verificação seguiu o padrão da inicialização, girando em torno de 5%.

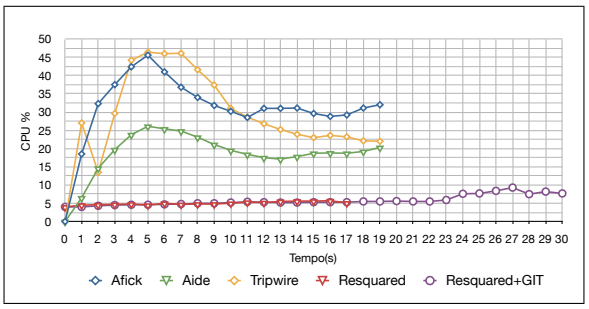

Figura 6. Porcentagem de CPU na operação de inicialização com diretório `/usr/bin`.

5.2. Efetividade de Recuperação

Nesta seção serão apresentados dois estudos de caso que demonstram a detecção de intrusão e suas respectivas recuperações por *REsquared*. Para cada cenário foi criada uma máquina virtual *Gentoo Linux* 32 bits simulando serviços normais providos por estações de laboratório. A maioria dos arquivos modificados por *rootkits* são utilitários simples do sistema Linux. Eles são modificados para ocultar evidências e atividades de intrusão[Pennington et al. 2003].

Para cada cenário estavam sendo monitorados os arquivos recomendados por [Pennington et al. 2003][Goel et al. 2005] e arquivos relacionados na configuração base de *Tripwire*, que é considerado o primeiro verificador de integridade existente.

5.2.1. Uso ilegal de disco

- **Cenário**: Um usuário logado no *host* roda um *exploit* para escalar privilégios—*proto_ops exploit*[1]—e consegue acesso de super-usuário. O atacante cria uma nova conta de super-usuário denominada *root100*, para conseguir isso faz escrita direta em dois arquivos, `/etc/passwd` e `/etc/shadow`. Logo após, cria um novo diretório e copia 100 arquivos protegidos por leis de *copyright*. Com a intenção de esconder seu armazenamento ilegal instalou uma variante do *t0rn rootkit*.

[1]Mais em: http://lwn.net/Articles/347006/

Este *rootkit* substitui arquivos binários do sistema responsáveis por mostrar ao usuário informações do sistema de arquivos e dos processos do sistema. Desta forma o atacante conseguiria esconder seus arquivos do usuário e esconder os processos do usuário *root100*.

- **Ações corretas de recuperação**: Remover todos os arquivos ilegais, os binários falsos e o *homedir* do usuário root100 criado pelo atacante. Além disso é necessário recuperar versão legítima dos arquivos `/etc/passwd` e `/etc/shadow`.
- **Ponto de detecção**: Verificação dos arquivos importados na ferramenta: arquivos alterados em `/etc` e substituição de binários.
- **Resultados**: A ferramenta proposta acusou alteração em 6 binários—`/bin/netstat`,`/bin/ls`, `/bin/ps`, `/usr/bin/top`, `/usr/bin/pstree`, `/sbin/ifconfig`—e nos três diretórios onde eles se encontram. *REsquared* também reportou corretamente as alterações nos arquivos `/etc/passwd` e `/etc/shadow`. Por intermédio da funcionalidade que mostra a diferença entre os arquivos texto infectados e os que estão sobre Controle de Versão—Seção 4, pode ser constatado o super- usuário criado pelo atacante. O que indicou uma posterior busca por arquivos "ocultos" criados por este usuário. Após serem identificados os arquivos alterados, o módulo de comunicação com o gerenciador de pacotes permitiu a identificação de quatro pacotes que necessitariam de reinstalação.

5.3. Infecção de *daemon* de conexão remota

- **Cenário**: Um usuário local consegue acesso privilegiado através de um *exploit* no módulo de *kernel* PulseAudio[2], após isso instala um *backdoor* sobre o OpenSSH[3] para garantir sua reentrada.
- **Ações corretas de recuperação**: Reinstalação do OpenSSH e restauração das configurações deste serviço na máquina vítima.
- **Ponto de detecção**: Verificação de rotina na máquina.
- **Resultados**: Flea é um *rootkit* para todas as distribuições que contém um utilitário de instalação escrito em `C`. Flea modifica configurações e aplicativos do OpenSSH deixando uma porta aberta para a reentrada do atacante ao sistema infectado, também sobrescreve alguns executáveis comuns da linha de comando com o objetivo de esconder seus processos que estão realizando escuta da rede para sua reentrada ou para obedecer comandos remotos. *REsquared* foi capaz de detectar alterações em 5 binários—/bin/ps; /bin/netstat; /usr/bin/pstree; /usr/bin/du; /usr/bin/slocate—e a alteração do *daemon* do OpenSSH e em suas configurações. A restauração envolveu a reinstalação de 6 pacotes e a restauração de 4 arquivos de configuração referentes ao OpenSSH. Foi necessária a intervenção do administrador apenas para reiniciar o *daemon* sem alterações.

6. Conclusão

Novas técnicas de intrusão surgem a cada dia, administradores gerenciam centenas de máquinas, usuários não se dão conta de que existem ameaças em *links* acessados e *e-*

[2]Servidor de som em rede instalado por padrão em várias distribuições. Mais detalhes do *exploit* em: http://www.securityfocus.com/bid/35721/

[3]Coleção de aplicativos que permitem sessões de comunicações cifradas em uma rede de computadores usando o protocolo SSH. http://www.openssh.com/

mails lidos. Essa é a realidade atual dos ambientes de produção, que além de tudo isso, está suscetível a problemas que exigem recuperação imediata. Uma vasta quantidade de metodologias é aplicada na detecção de intrusão e ou alteração de sistemas, podendo até haver uma combinação delas trabalhando com o mesmo ideal. Verificadores de integridade são usados para capturar mudanças indesejadas no sistema de arquivos em busca de alterações maliciosas.

Foi descrito e implementado neste artigo um versátil Verificador de Integridade de sistema de arquivos chamado *REsquared*, que, traz como principal diferencial, a possibilidade de recuperação do sistema sem necessidade de *backup* comum dos aplicativos. Através das funcionalidades presentes em controladores de versão, é possível que sejam restaurados estados anteriores das configurações sistema. A recuperação dos aplicativos é confiada através de comunicação com o gerenciador de pacotes, item comum a todas as distribuições atuais. *REsquared* prevê portabilidade em diferentes distribuições através do módulo de comunicação com o gerenciador de pacotes, que pode ser desenvolvido de acordo com a distribuição onde a ferramenta será utilizada. Deve-se ressaltar que a ferramenta não possui habilidades para recuperação de dados do usuário, este tipo de informação deverá ser confiado a sistemas de *backup* comuns.

Por meio de estudos comparativos constatou-se que *REsquared* teve sobrecarga de aproximadamente 52% na operação de inicialização dos dados de controle—que serão usados em futuras verificações do sistema. O tempo médio da operação de verificação e de consumo de memória foi semelhante entre todas as ferramentas, porém *REsquared* teve consumo médio de CPU consideravelmente menor, o que leva à conclusão que *REsquared* tem maiores benefícios de uso em servidores, que dependem de ferramentas de baixo consumo para não afetarem sua disponibilidade. Dois estudos de caso avaliaram o poder de detecção, diagnóstico e recuperação automatizada da ferramenta, que mostrou-se capaz de auxiliar administradores de sistema em tarefas que normalmente seriam exaustivas.

Referências

[Afick 2010] Afick (2010). Afick: Another file integrity checker. http://afick.sourceforge.net/.

[Banikazemi et al. 2005] Banikazemi, M., Poff, D., and Abali, B. (2005). Storage-based file system integrity checker. In *StorageSS '05: Proceedings of the 2005 ACM workshop on Storage security and survivability*, pages 57–63, New York, NY, USA. ACM.

[Berliner and Prisma 1990] Berliner, B. and Prisma, I. (1990). CVS II: Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, volume 341, page 352.

[Collin-sussman et al. 2007] Collin-sussman, B., Fitzpatrick, B. W., and Pilato, C. M. (2007). Version control with subversion. E-book http://svnbook.red-bean.com/.

[Craig and McNeal 2003] Craig, W. D. and McNeal, P. M. (2003). Radmind: The integration of filesystem integrity checking with filesystem management. In *LISA '03: Proceedings of the 17th USENIX conference on System administration*, pages 1–6, Berkeley, CA, USA. USENIX Association.

[Feiler 1991] Feiler, P. (1991). Configuration management models in commercial environments. Technical Report CMU/SEI-91-TR-7 ESD-9-TR-7, Software Engineering Institute. Carnegie Mellon, University. Pittsburgh.

[Gift and Shand 2009] Gift, N. and Shand, A. (2009). Introduction to distributed version control systems. IBM Technical library https://www.ibm.com/developerworks/aix/library/au-dist_ver_control/.

[Goel et al. 2005] Goel, A., Po, K., Farhadi, K., Li, Z., and de Lara, E. (2005). The taser intrusion recovery system. *SIGOPS Oper. Syst. Rev.*, 39(5):163–176.

[Kim and Spafford 1994] Kim, G. H. and Spafford, E. H. (1994). The design and implementation of tripwire: a file system integrity checker. In *CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security*, pages 18–29, New York, NY, USA. ACM.

[Osiris] Osiris. Osiris: Host integrity management tool. In *www.osiris.com*.

[O'Sullivan 2007] O'Sullivan, B. (2007). Distributed revision control with Mercurial. *Mercurial project*.

[Pennington et al. 2003] Pennington, A. G., Strunk, J. D., Griffin, J. L., Soules, C. A. N., Goodson, G. R., and Ganger, G. R. (2003). Storage-based intrusion detection: watching storage activity for suspicious behavior. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 10–10, Berkeley, CA, USA. USENIX Association.

[Reed 2003] Reed, J. (2003). File integrity with aide. www.iforkr.org/bri/presentations/aide.

[Rochkind 1975] Rochkind, M. J. (1975). The source code control system. *IEEE Trans. Software Eng.*, 1(4):364–370.

[Samhain 2010] Samhain, L. (2010). Samhain: File system integrity checker. http://samhain.sourceforge.net.

[Shah 2001] Shah, B. (2001). How to choose introduction detection solution. Whitepaper http://www.sans.org/reading_room/whitepapers/detection/choose-intrusion-detection-solution_334, SANS Institute.

[Strunk et al. 2002] Strunk, J. D., Goodson, G. R., Pennington, A. G., Craig, S. A. N., and Ganger, G. R. (2002). Intrusion detection, diagnosis, and recovery with self-securing storage. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh. CMU-CS-02-140.

[Suel and Memon 2002] Suel, T. and Memon, N. (2002). Algorithms for delta compression and remote file synchronization. *Lossless Compression Handbook, Academic Press, 2002*.

[Tichy 1985] Tichy, W. F. (1985). Rcs - a system for version control. *Software, Practices and Experience*, 15(7):637–654.

[Wheeler 2008] Wheeler, D. A. (2008). Free software (oss/fs) software configuration management (scm) / revision-control systems. Comments on Open Source Software.

[Wiegley 2008] Wiegley, J. (2008). Git from from the bottom up. E-book http://ftp.newartisans.com/pub/git.from.bottom.up.pdf.