O vigia dos vigias: um serviço RADIUS resiliente

Oleksandr Malichevskyy, Diego Kreutz, Marcelo Pasin e Alysson Bessani

Universidade de Lisboa, Faculdade de Ciências
{olexmal,kreutz}@lasige.di.fc.ul.pt
{pasin,bessani}@di.fc.ul.pt

Resumo RADIUS é um protocolo amplamente utilizado para autenticação, autorização e contabilização nas infra-estruturas e serviços de rede. Um servidor RADIUS pode falhar ou apresentar comportamento arbitrário também chamadas faltas bizantinas, decorrente de erros no programa, falhas no ambiente, ou ainda, uma intrusão. Este artigo apresenta uma solução para aumentar a resiliência do serviço RADIUS. A ideia central passa pela utilização de protocolos de replicação activa, utilizando máquinas de estado, de forma a prover tolerância a faltas e intrusões. Como caso de uso e avaliação é utilizado um ambiente de acesso a rede sem fio, similar ao que ocorre na rede *eduroam*. É apresentada e discutida uma arquitetura de operação do serviço, desde o cliente a ser autenticado até o serviço de autenticação, e como cada componente da arquitetura pode tolerar faltas. Os resultados iniciais demonstram a aplicabilidade da solução proposta.

Abstract. RADIUS is a widely used protocol for authentication, authorization and accounting for the infrastructures and network services management. One of RADIUS problems continue to be the server resilience, which may fail or become Byzantine due to a bug, problem / failure of the environment or because of an intrusion. This article presents a solution to increase the resilience of a RADIUS service. The central idea is the use of state machine replication active protocols, in order to provide fault and intrusion tolerance. We present an architecture for service operation, from the client to the RADIUS back-end, and how each component of the architecture can tolerate faults. As experimental evaluation of a wireless network access scenario, similar to what occurs on the network *eduroam*, is presented and discussed. Initial results demonstrate the applicability of the proposed solution.

1 Introdução

Os protocolos AAA (Authentication, Authorization and Accounting) [1], como o RADIUS [9], são amplamente difundidos e utilizados nos mais diversos contextos. Operadoras de telecom, por exemplo, utilizam o RADIUS para autenticar e controlar o acesso dos clientes dial-up, ADSL, entre outros serviços. Diversas organizações e universidades utilizam o protocolo para controlar acesso às suas

redes e outros recursos informáticos. Um dos exemplos mais bem-sucedidos é o da rede *eduroam*, uma federação que utiliza uma arquitetura de servidores RADIUS que permite autenticação de utilizadores para acesso a recursos de rede e mobilidade. Um utilizador registado em uma instituição qualquer da federação pode ter acesso a rede de qualquer outra instituição da federação, servindo-se das suas credenciais atribuídas em seu registo.

Apesar do RADIUS ser um dos protocolos AAA mais utilizados, pouca atenção tem sido dada aos potenciais riscos atrelados ao comprometimento de um servidor AAA dentro ou fora de uma federação. Com o aumento dos riscos e incidentes de segurança nos últimos anos, passa a ser cada vez mais crítica a existência de mecanismos e formas de proteção contra acções ou comportamentos maliciosos em serviços e infraestruturas. Neste âmbito, o objetivo deste trabalho é atacar um dos problemas de servidores RADIUS, que consiste na falta de resiliência. Essencialmente, os servidores RADIUS existentes não são projetados para resistir a comportamentos bizantinos, como aqueles causados por bugs, problemas na infra-estrutura ou intrusões.

O caso de uso em evidência neste artigo é o controlo de acesso a redes sem fio, que é um dos usos mais frequentes do RADIUS. Este trabalho propõe modificações no serviço RADIUS de forma a tolerar faltas e intrusões, utiliza mecanismos de autenticação fortes, e demonstra como a arquitetura pode tornar o serviço mais resiliente. Mesmo fazendo uso de replicação em seus componentes internos, a solução proposta permite utilizar componentes de software padrão na extremidade mais próxima ao cliente. Todo material criptográfico é protegido por um componente seguro, de forma a impossibilitar que um intruso no servidor possa ter acesso ao material criptográfico dos clientes e do servidor. Testes mostraram que a arquitetura funciona bem, e apesar de servidor de autenticação ser replicado o sistema consegue ter bons resultados de desempenho.

Este texto está estruturado da seguinte forma. A secção 2 apresenta a motivação do trabalho. Uma visão geral do contexto e problema é apresentada na secção 3. Na secção 4 são detalhados os modelos de sistema e adversário. Informações sobre a implementação e avaliação são providos nas secções 5 e 6. Por fim, os trabalhos relacionados e as conclusões são apresentados nas secções 7 e 8.

2 Motivação

Os padrões de acesso as redes sem fios, como o 802.1x, descrevem modelos de acesso centralizado que se integram com o padrão AAA [1]. Este padrão disponibiliza um framework que permite configurar routers e servidores de autenticação de forma a disponibilizar os três componentes deste padrão.

O padrão AAA permite um controlo abrangente sobre o que se passa na rede. A autenticação verifica a identidade digital do utilizador. A autorização verifica o que utilizador pode ou não aceder e que permissões tem. A contabilização permite, por exemplo, cobrar pelo uso da rede ou também pode ser utilizada para o controlo de acesso, verificação de erros e detecção de ataques, pois esta guarda informações sobre o uso dos recursos da rede por cada utilizador.

Um dos protocolos AAA mais conhecidos é o RADIUS [9]. Este protocolo é largamente utilizado para providenciar autenticação em diferentes redes e sistemas, tais como dial-up, VPN e redes sem fio. O RADIUS baseia-se no modelo cliente/servidor, onde o cliente, denominado de NAS, é responsável por proceder a autenticação e autorização, junto do RADIUS, para então permitir o acesso a recursos da rede.

O servidor RADIUS pode suportar uma variedade de métodos de autenticação. Quando o utilizador procede a uma autenticação usando nome de utilizador e uma palavra chave, o protocolo pode suportar PPP PAP [5], CHAP [8,11] e login de UNIX. A autenticação também pode ser realizada através de certificados (e.g. EAP-TLS).

A arquitetura de base do RADIUS é apresentada na figura 1. Os componentes essenciais são o suplicante, o NAS e o servidor AAA. O utilizador (suplicante) realiza uma requisição de acesso a rede ao NAS, que encaminha o pedido de autenticação e autorização ao servidor AAA. Este, por sua vez, pode utilizar um back-end, como SQL, Kerberos, LDAP ou Ative Diretory, para validar as credenciais do utilizador. Uma vez autenticado, o servidor AAA envia a confirmação para o NAS, que permite o acesso do solicitante a rede.

Fig. 1. Arquitetura do RADIUS.

Na arquitetura do RADIUS componentes como o NAS e o próprio servidor AAA podem ser replicados. Em cada NAS é configurada uma lista de servidores RADIUS que podem ser contatados para realizar a autenticação e autorização do solicitante. Entretanto, um servidor AAA comprometido ainda contínua a ser um ponto de falha da arquitetura, visto que um atacante pode utilizar o servidor para capturar dados de utilizadores ou mesmo gerar novas autorizações, para utilizadores anteriormente inexistentes.

O objetivo da autenticação utilizando o protocolo RADIUS é gerar uma chave simétrica entre o suplicante e o NAS, para cifrar a comunicação, visto que o meio de comunicação é aberto. Assim, se o servidor de RADIUS for comprometido, o atacante consegue aceder aos dados não cifrados que são trocados entre o suplicante e o NAS. Portanto, além da replicação para disponibilidade do serviço, são necessários métodos mais robustos e confiáveis de autenticação.

Os protocolos de autenticação, como PPP PAP e CHAP, vulgarmente utilizados no RADIUS são fracos, e podem sofrer ataques que permitam ter acesso as credenciais deste, (e.g. no caso de PPP PAP, a senha é enviada em formato de texto não cifrado). Para resolver esse problema podem ser utilizados métodos de autenticação adicionais, como o EAP. Este protocolo permite que o utilizador se autentique perante o servidor de autenticação sem ter um método de autenticação previamente combinado. Para suportar este método de autenticação o protocolo RADIUS funciona apenas como transporte de pacotes EAP. Desta

forma, basta alterar o suplicante e o servidor de autenticação, sem haver necessidade de modificar o NAS. Um dos aspetos mais positivos em relação ao uso do EAP é a possibilidade de uma autenticação fim-a-fim, sem que os pacotes sejam alterados nos nós intermédios da rede.

O EAP suporta uma grande variedade de métodos de autenticação. Entre eles MD5, TLS, TTLS, PEAP, FAST e LEAP. Neste trabalho será utilizado e discutido o EAP-TLS, pois este protocolo permite a utilização de certificados, autenticação mútua, altos níveis de segurança [6] e geração segura de chaves simétricas através de *handshake* do protocolo TLS 3.1. O EAP-TLS sobre o RADIUS possibilita autenticação fim-a-fim, permitindo construir um túnel para o protocolo TLS desde de o suplicante até ao servidor de autenticação. Este método de autenticação permite-nos efetuar uma autenticação bilateral, o que faz com que o suplicante, antes de se enviar suas credenciais para o servidor AAA, consiga verificar a identidade do mesmo.

Os problemas apresentados são a motivação principal do desenvolvimento deste trabalho. O objetivo é prover uma arquitetura resiliente para autenticação e autorização em redes sem fios, utilizando o protocolo RADIUS. A próxima secção apresenta uma visão geral da arquitetura e mecanismos propostos para resolver o problema de resiliência do servidor AAA e confidencialidade dos dados dos utilizadores.

3 Visão geral

O protocolo RADIUS é constituído, na componente de autenticação, por quatro tipo de pacotes, "Access-Request", "Access-Challenge", "Access-Accept" e "Access-Reject". O pacote "Access-Request" é enviado pelo NAS ao servidor de autenticação solicitando acesso a rede para o suplicante. O "Access-Challenge" é usado para enviar um desafio ao NAS. Este, por sua vez, pode responder ao desafio ou envia-lo ao suplicante. Para responder ao desafio o NAS envia de novo o pacote "Access-Request" com a resposta. Os pacotes "Access-Accept" e "Access-Reject" servem para indicar ao NAS se este pode ou não liberar o acesso a rede ao suplicante. O pacote "Access-Accept" pode conter outros dados necessários para o NAS dar acesso ao suplicante, como parâmetros de QoS ou tabela de permissões.

O NAS não interpreta o conteúdo dos pacotes EAP, limitando-se a passar estes ao servidor RADIUS como atributo do pacote. Como alguns dos métodos EAP necessitam trocar muitas mensagens, esta troca é realizada através de pacotes "Access-Request" e "Access-Challenge". A comunicação prossegue assim até o servidor de autenticação autorizar ou não o acesso ao suplicante.

Mesmo com métodos seguros de autenticação a arquitetura do RADIUS apresenta um ponto de falha que é o servidor de autenticação. Para resolver este problema é necessário replicar o servidor de autenticação. O novo desafio que surge com a replicação é como garantir que a autenticação fim-a-fim, através de EAP-TLS, ocorra em um serviço replicado. Os detalhes técnicos da solução para

este problema são apresentados na secção 5. Basicamente, a solução passa pela utilização de geradores replicados de números pseudo-aleatórios determinísticos.

A figura 2 ilustra a arquitetura proposta. Foi adicionado um novo componente, denominado de "Untrusted Gateway", cuja função é receber as requisições EAP-TLS, encapsuladas pelo NAS num pacote RADIUS, e repassá-las ao serviço de autenticação replicado. O *gateway* é um componente sem estado que simula o comportamento de um servidor RADIUS. A sua principal função é tornar o serviço replicado transparente para os demais componentes, ou seja, NAS e suplicante. O NAS contínua a actuar normalmente, contendo uma lista de servidores AAA (*gateway*s na nova arquitetura). O *gateway*, ao receber a requisição do NAS, contata as réplicas do serviço de autenticação e aguarda por um quórum mínimo de respostas idênticas. Ao receber a confirmação de um número mínimo de réplicas, encaminha a resposta ao NAS. Este não consegue violar a confidencialidade e autenticidade da comunicação visto que a autenticação EAP-TLS é estabelecida fim-a-fim entre o suplicante e as réplicas, acrescido das garantias de integridade dadas pelo protocolo RADIUS, ver secção 5.

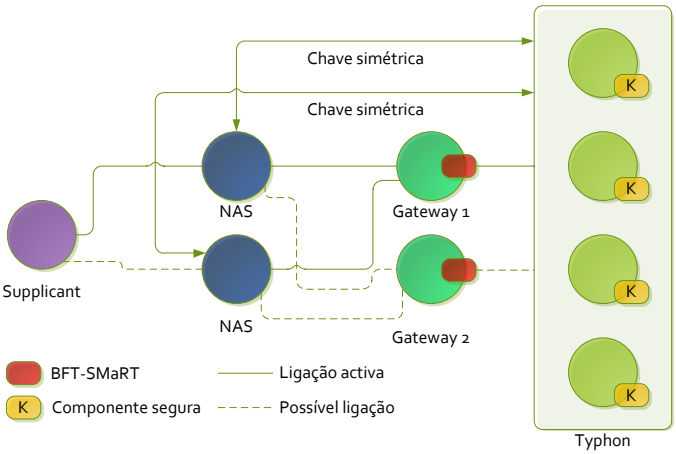

Fig. 2. A arquitetura proposta.

O serviço de replicação e o componente seguro do Typhon [7] foram utilizados como mecanismos básicos para realizar a replicação do servidor RADIUS. Este por sua vez faz uso de uma uma biblioteca de replicação, BFT-SMaRt [12][1]. O Typhon é um serviço de autenticação e autorização originalmente implementado para prover a versão 5 do Kerberos tolerante a faltas bizantinas e a intrusões. Ele utiliza a técnica de replicação de máquina de estados e dos componentes seguros. A replicação de máquina de estados permite-nos tolerar faltas por paragem e faltas bizantinas enquanto a componente segura permite-nos que as chaves dos clientes e todo material criptográfico seja mantido em segredo, mesmo na presença de um intrusão no servidor. Estes dois recursos formam os pilares essenci-

[1] `http://code.google.com/p/bft-smart/`

ais do servidor RADIUS replicado implementado, como será descrito nas próximas secções.

4 Modelo de Sistema e Adversário

A seguir são descritas os principais modelos com relação a sincronia, rede e faltas. Diferentes componentes da arquitetura proposta podem ter características diferenciadas, como modelo de faltas bizantino e protocolo de comunicação 802.1x.

Modelo de sincronia. O sistema tem sincronia parcial [4], ou seja, há sincronia o suficiente para resolver os algoritmos de consenso utilizados nos mecanismos de replicação com máquina de estado.

Modelo de rede e comunicação. Os componentes da arquitetura possuem meios de comunicar-se uns com os outros de acordo com a necessidade. Por exemplo, os *gateway*s possuem comunicação, TCP/IP, com todas as réplicas do serviço de autenticação. O NAS, da mesma forma usando UDP, consegue aceder os *gateway*s. Já os suplicantes utilizam 802.1x para comunicarem-se com os NAS.

Modelos de faltas. O serviço de autenticação proposto tolera faltas bizantinas e intrusões. Neste caso, existem $3f + 1$ réplicas, suportando até f faltas sem prejudicar o funcionamento correto do sistema. Por outro lado, no caso dos componentes NAS e *gateway* são assumidas faltas por paragem e algumas faltas bizantinas detetáveis pelos protocolos, como a violação da integridade dos pacotes RADIUS, falta de respostas (pacotes descartados) e reply de mensagens. Estes componentes podem ser replicadas para aumentar a disponibilidade, levando a uma arquitetura com $f_N + 1$ NASs e $f_G + 1$ *gateway*s. Caso a autenticação esteja a falhar, cabe ao suplicante tentar conetar-se a outros NAS. Assim como o NAS, da mesma forma, fará as requisições a diferentes *gateway*s uma vez que o primário (em utilização) falhar.

Na arquitetura proposta, que pode ser vista na figura 2, cada componente de sistema esta a ser executada numa máquina separada. Estas máquinas podem estar localizadas em diferentes locais ou sub-redes. O único requisito é que sejam respeitadas as necessidades de comunicação entre os respetivos pares de componentes, como é o caso do NAS e do *gateway*. Isso faz com que o adversário tenha uma maior dificuldade de atacar e comprometer o sistema como um todo. Adicionalmente, componentes como os extitgateways e as réplicas do serviço de autenticação podem, inclusive, estarem alocados em diferentes domínios administrativos, dificultando ainda mais o trabalho dos potenciais atacantes.

O sistema foi projetado para tolerar quaisquer faltas bizantinas no serviço de autenticação e algumas faltas bizantinas em componentes como o NAS e o *gateway*. Estes componentes podem ser substituídos a qualquer hora, uma vez que não contem estado. As falhas por paragem são facilmente transpostas pelos respetivos protocolos, como é o caso do RADIUS e do 802.1x. Contudo, comportamentos bizantinos podem, eventualmente, levar o suplicante a exaustão, ou

seja, número de tentativas consecutivas sem sucesso a um mesmo NAS comprometido. Ou seja, se o NAS tiver comportamento bizantino pode negar o acesso ao suplicante, sem que este se aperceba de comportamento anormal do NAS. Alguma replica pode se tornar bizantina, mas isso não constitui um problema maior desde que o quórum seja mantido. Caso isso não seja possível o *gateway* pode ligar-se a outro conjunto de réplicas do RADIUS. A seguir são apresentados alguns exemplos de falhas detetadas e contornadas pelos componentes da arquitetura.

Se um *gateway* falhar por paragem ou se tornar bizantino, começar a enviar pacotes errados, o NAS deteta e tenta retransmitir x_T vezes. Se mesmo assim o comportamento se mantiver ou o *gateway* deixar de responder, durante um tempo predefinido t_R, o NAS automaticamente procede a troca do *gateway*. A deteção de pacotes corrompidos ou forjados é possível porque o NAS e o servidor de autenticação RADIUS partilham uma chave secreta que nunca é enviada pela rede. Com este segredo conseguem verificar a integridade dos pacotes.

Um ataque ao *gateway* não compromete o sistema uma vez que este limita-se a passar os pacotes de um lado para outro e não tem qualquer conhecimento de chaves partilhadas entre os NAS o os servidores de autenticação. Além disso, a autenticação entre o suplicante e o servidor RADIUS é mútua, fim-a-fim através de certificados TLS, logo é assumido a partida que o sistema pode ser atacado nas componentes intermédias. As réplicas do Typhon são tolerantes as faltas bizantinas. Mesmo se uma delas se tornar bizantina o sistema continua a funcionar de forma correta. O serviço de autenticação pode, inclusive, sofrer uma intrusão. Contudo, como as chaves estão na componente segura, o adversário não consegue obter qualquer tipo de informação sobre elas.

5 Implementação

A concretização da arquitetura apresentada na figura 2 consistiu na implementação de um *gateway* e do serviço de autenticação RADIUS replicado. A codificação foi realizada em Java, com o apoio das bibliotecas Bouncy Castle, BFT-SMaRt e do serviço de autenticação Typhon. Este último foi adaptado para suportar autenticação mutua EAP-TLS entre o suplicante e o back-end do servidor RADIUS, o Typhon.

Os pacotes EAP são transportados do suplicante até o NAS, através de um túnel sobre o protocolo 802.1x. Depois de chegarem ao NAS, o pacote EAP é encapsulado no pacote RADIUS como um atributo deste. A resposta do servidor de autenticação segue um caminho inverso. Primeiro o servidor encapsula o pacote EAP dentro de um atributo do pacote RADIUS e o envia ao NAS. Este encapsula o pacote EAP dentro do 802.1x e envia-o ao suplicante.

A autenticação mútua EAP-TLS prossegue da seguinte forma. Primeiro o suplicante envia um pacote com a identidade do utilizador, esta pode ser usada para determinar que método EAP o servidor deve usar para autenticá-lo. A resposta do servidor indica o início do protocolo de autenticação. Depois de receber esta mensagem o suplicante envia uma mensagem com os métodos de cifragem

que ele conhece, e um número aleatório. O único método implementado neste trabalho (único que é obrigatório) é o TLS_RSA_WITH_3DES_EDE_CBC_SHA.

Na segunda fase é realizada a autenticação do servidor. O servidor envia uma mensagem onde ele define qual método de cifragem será usado e um número aleatório do servidor a ser usado para gerar a chave de sessão. A seguir, o servidor envia o seu certificado assinado por uma CA em que o suplicante confia e consiga validar a assinatura. Depois de validar o certificado do servidor o suplicante envia o seu certificado, uma notificação de que o certificado do servidor foi validado e um número aleatório chamado de pré-master cifrado com a chave pública do servidor. Este número aleatório serve para o servidor gerar a chave master de sessão usando-o em conjunto com o número aleatório dele e o do cliente.

Integridade dos pacotes RADIUS. O NAS é capaz de verificar a integridade de um pacote RADIUS através do campo `Authenticator`. No caso de um pedido do NAS, o Authenticator é um número aleatório denominado de Request Authenticator. O valor tem de ser imprevisível e único durante a duração de segredo partilhado entre cliente e o servidor. Quando o servidor responde ao pedido, aceitando-o, rejeitando-o, ou oferecendo um desafio, este campo é denominado como Response Authenticator, contendo um resumo critpográfico (MD5 [10]) calculado a partir do conjunto de octetos formado pelo pacote RADIUS, começando com campo `Code`, incluindo `Identifier`, `Length`, `Request Authenticator` e atributos (`Attributes`) e, por fim, a chave secreta partilhada entre o NAS e o servidor RADIUS. Em resumo, temos ResponseAuth = MD5(`Code` + `Identifier` + `Length` + `Request Authenticator` + `Attributes` + `Secret`), onde '+' denota a concatenação. Desta forma a resposta de servidor é autenticada e o pacote é protegido de ataques de homem-no-meio uma vez que o atacante não conhece a chave partilhada entre o NAS e servidor de autenticação.

Como no primeiro pacote enviado pelo NAS o autenticador é um numero aleatório, e na verdade não autentica o pacote, é necessário ter uma autenticador que seja igualmente forte e que permita verificar a integridade de pacote desde de inicio da comunicação. Com uso de EAP o protocolo RADIUS define a obrigatoriedade de uso de um autenticador chamado Message-Authenticator [2]. Este autenticador é passado como atributo de pacote RADIUS e é através de computação HMAC-MD5 (`Code` + `Identifier` + `Length` + `Request Authenticator` + `Attributes`).

Tendo em conta a existência de dois autenticadores, `Autenticator` e `Message-Authenticator`, podemos garantir a integridade dos pacotes RADIUS. A combinação dos dois autenticadores impossibilita o *gateway* de forjar ou corromper dados sem ser detetado, pois este desconhece o segredo partilhado entre o NAS e o serviço de autenticação. A única ação possível do *gateway* seria o descarte de pacotes, o que eventualmente levará o NAS a realizar tentativas de autenticação junto a outros *gateway*s.

Adaptações para o método EAP-TLS replicado. O processo de autenticação via o método EAP-TLS engloba uma sequência de mensagens para completar a autenticação. O primeiro problema que surge é a incapacidade de o servidor de autenticação de identificar o suplicante, uma vez que as requisições partem todas

do mesmo NAS. Uma alternativa para resolver o problema é o atributo `State` do RADIUS. Este atributo é um número aleatório de 16 octetos que é gerado pelo servidor depois de receber a primeira mensagem do suplicante (identidade do utilizador), e é mantido até ao fim da comunicação.

Na execução do *handshake* de TLS é necessário gerar alguns números aleatórios. Este representa um segundo desafio da implementação, uma vez que as réplicas do serviço precisariam gerar números aleatórios de uma forma determinista. Como ainda não existem trabalhos de geração de números aleatórios de uma forma determinista num sistema replicado, a solução passa pela utilização de geradores de números pseudo-aleatórios com características que garantam o determinismo necessário ao sistema.

A solução que foi encontrada é usar uma função de geração de números pseudo-aleatórios que é usada cada vez que a replica necessitar de gerar um número aleatório. Todas as replicas começam com uma semente s, uma string aleatória l e um segredo partilhado com o NAS k_S. Todas as replicas têm o mesmo segredo. Depois de a função PRF (Pseudo-random-funcion) ser chamada, o segredo k_S é dividido em dois, k_S1 e k_S2, s é concatenado com l, e, por fim, é gerado um HMAC-MD5$(k_S1, s) \otimes$ HMAC-SHA-1 (k_S2, s). Na sequência é executada a seguinte operação de atribuição $s =$ HMAC-MD5 (k_S, s). Desta forma, cada vez que a função é chamada a semente muda e todas as réplicas passam a gerar o mesmo número aleatório. A semente faz parte de estado de servidor, ou seja, quando a função PRF é chamada todas as réplicas evoluem da mesma forma. Consequentemente, todas as réplicas novas ou recuperadas irão possuir a mesma semente uma vez que este faz parte do estado do sistema e é atualizado/sincronizado a cada transferência de estado entre réplicas.

6 Avaliação

O objetivo desta secção é apresentar os primeiros resultados de avaliação da arquitetura proposta. A solução implementada, baseada no Typhon, é comparada com a implementação base do FreeRADIUS, uma das mais utilizadas em ambientes de produção. O Typhon foi escrito em linguagem Java e o FreeRADIUS em linguagem C.

O ambiente de testes é constituído de 7 máquinas físicas, devido a replicação, no caso de Typhon e 3 máquinas físicas no caso de FreeRADIUS. As máquinas de teste são Intel Xeon E5520, 32 GB (8x4GB) de memoria RAM e Broadcom NetXtreme II BCM5716 Gigabit Ethernet. O ambiente de testes do Typhon foi composto por um cliente, dois *gateway*s e quatro réplicas do Typhon de modo a tolerar uma falha. O segundo ambiente foi constituído por um cliente e duas instalações de FreeRADIUS.

O objetivo foi realizar duas avaliações, uma medindo a latência e outra o débito (desempenho). Os dados da latência e do débito foram obtidos a partir da média de X autenticações, executadas e mensuradas em lotes de 500 autenticações no caso de teste de latência e 1000 no caso dos testes de débito.

Para realizar o teste de débito foi escrito um programa em C que, mede o débito do sistema. Este programa permite medir número de pacotes que o `tcpdump` apanha. O débito é calculado dividindo o número de mensagens por tempo decorrido.

Latência. Os resultados de teste latência estão exibidos na figura 3, onde, a cada teste, foram realizadas 10000 autenticações EAP-TLS completas. Para certificar-se que o compilador JIT (Just-In-Time) não causa efeitos no teste, foram realizadas 20000 autenticações, descartando-se as primeiras 10000, de modo a eliminar a fase onde o compilador poderia ter afetado. Os dois testes do Typhon mostraram uma latência parecida.

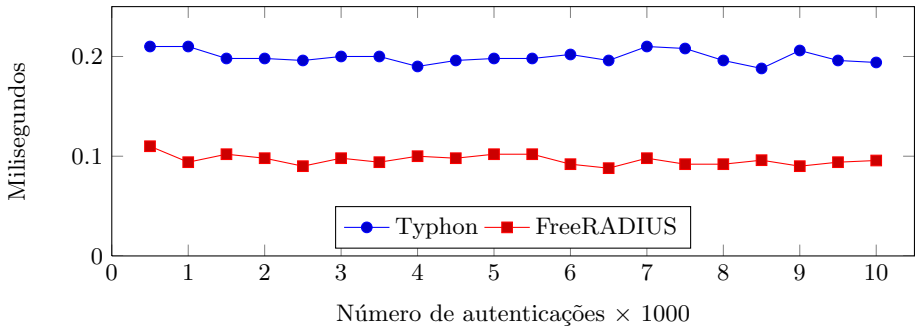

Fig. 3. Latência de uma autenticação, médias calculadas a cada 500 autenticações

Como pode ser observado na figura 3, existem algumas diferenças em termos de desempenho de Typhon em relação ao FreeRADIUS. Os testes mostraram que o Typhon é mais lento do que o FreeRADIUS. Isso deve-se, em especial, pelo facto do primeiro ser um sistema replicado e necessitar uma maior troca de mensagens devido aos protocolos BFT e replicação de máquina de estados. Apesar do Typhon ser um sistema replicado, consegue mostrar uma latência razoável, aceitável em ambientes de redes locais e corporativas.

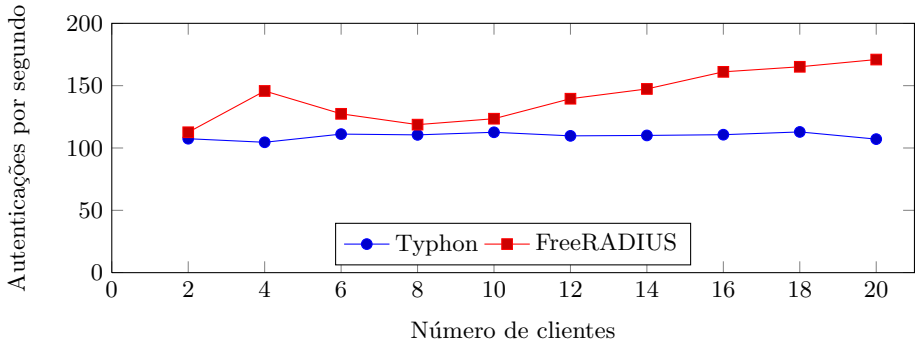

Fig. 4. Débito (autenticações por segundo) em função do número de clientes

Debito. É possível observar no grafico 4 que o débito do FreeRADIUS varia porque ele possui uma piscina de threads de servidores, o que permite aumentar dinamicamente a capacidade de vazão do sistema de acordo com o número de requisições recebidas. É possível verificar no gráfico 4 que a partir de 4 clientes o débito começa a diminuir. No entanto, a partir de 8 clientes este voltou a aumentar. Isso deve-se ao gestor de servidores do FreeRADIUS. O limite inferior de servidores foi configurado para 3 e limite superior para 30. No Typhon, o débito manteve-se sempre constante, apesar do aumento no número de clientes. Uma solução parecida à do FreeRADIUS (thread pool) poderia ser tentada desde que fosse possível manter o determinismo na ordem das chamadas ao Typhon, o que ainda não foi solucionado.

7 Trabalhos Relacionados

Uma maneira de atingir mais segurança no servidor RADIUS é através de um back-end composto por micro controladores, tais como servidores de autenticação TEAPM [15]. O servidor de autenticação pode ser implementado dentro de um Smart Card [3]. Entretanto, o principal ponto negativo é o desempenho, uma vez que esses dispositivos possuem reduzidas capacidades de armazenamento e processamento.

O servidor de autenticação que corre como uma máquina física normal pode autenticar muitos utilizadores em simultâneo sem grande esforço. Mesmo utilizando grids de Smart Cards [13] não é possível chegar-se próximo do desempenho de um microcomputador convencional. Foi demonstrado que um processo de autenticação, para um conjunto reduzido de utilizador, utilizando grids de Smart Cards, leva mais de 2 segundos para finalizar [14]. Esse tempo é inaceitável em um ambiente de produção e o suficiente para processar alguns milhares de autenticações num sistema tipo PC.

Neste sentido, o objetivo do trabalho foi propor, implementar e avaliar uma arquitetura RADIUS mais resiliente que as convencionais e com padrões de desempenho necessários a ambientes reais. Isso foi atingido através das técnicas de tolerância a faltas e intrusões utilizadas no trabalho e validadas na prática.

8 Conclusão

O RADIUS é um protocolo amplamente utilizado em empresas e universidades. Como ele é utilizado também nas redes abertas, foram sendo adicionados mais e novos mecanismos de proteção, como EAP, que permitem aumentar a segurança do protocolo. Mesmo assim ainda persistem alguns pontos de falha que podem fazer com que o adversário consiga induzir falhas no sistema.

Neste artigo foi apresentada a arquitetura, implementação e avaliação de uma solução que visa aumentar a resiliência de serviços RADIUS. Ela adiciona um novo componente, denominado *gateway*, e utiliza técnicas de tolerância a faltas e intrusões, como replicação de máquina de estados para componentes

bizantinos, no serviço de autenticação. Os resultados apresentados demonstram a aplicabilidade e viabilidade da solução proposta.

Agradecimentos. Gostaríamos de agradecer a João Sousa pela ajuda no uso da biblioteca BFT-SMaRt e do sistema Typhon. Este trabalho é suportado pela comissão europeia através do projecto SecFuNet (STREP 288349, FP7-ICT-2011-EU-Brazil) e pela FCT através dos programas multianual (LaSIGE) e CMU-PORTUGAL e do projecto CloudFIT (PTDC/EIA-CCO/108299/2008).

Referências

1. B Aboba and J Wood. RFC 3539: Authentication, authorization and accounting (AAA) transport profile. *Request for Comments*, 2003.
2. P. Calhoun B. Aboba, Microsoft and Airespace. RFC 3579: Radius (remote authentication dial in user service) support for extensible authentication protocol (eap). *Request for Comments*, 2003.
3. Serge Chaumette, Pascal Grange, Achraf Karray, Damien Sauveron, and Pierre Vignéras. Secure distributed computing on a java card; grid.
4. Cyntia Dwork, Nancy A. Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–322, 1988.
5. Lianfen Huang, Ying Huang, Zhibin Gao, Jianan Lin, and Xueyuan Jiang. Performance of authentication protocols in lte environments. *2009 Intl. Conf. on Computational Intelligence and Security*, 2:293–297, 2009.
6. Intel®. 802.1x overview and EAP types. `http://www.intel.com/support/wireless/wlan/sb/cs-008413.htm`, March 2012.
7. João Sousa, Alysson Bessani and Paulo Sousa. Typhon: Um serviço de autenticação e autorização tolerante a intrusões. *Inforum 2010*, 2010.
8. David Q. Liu and Mark Coslow. Extensible authentication protocols for ieee standards 802.11 and 802.16. In *Proceedings of the International Conference on Mobile Technology, Applications, and Systems*, Mobility '08, pages 47:1–47:9, New York, NY, USA, 2008. ACM.
9. C Rigney, S Willens, A Rubens, and W Simpson. RFC 2865: Remote authentication dial in user service (RADIUS). *Request for Comments*, 2007, 2000.
10. R. Rivest. RFC 1321: The MD5 message-digest algorithm. *Request for Comments*, 1992.
11. William Allen Simpson. Ppp challenge handshake authentication protocol (chap). Internet RFC 1994, August 1996.
12. Joao Sousa and Alysson Bessani. From Byzantine consensus to bft state machine replication: A latency-optimal transformation. In *Dependable Computing Conference (EDCC), 2012 Ninth European*, pages 37 –48, may 2012.
13. P. Urien and M. Dandjinou. Introducing smartcard enabled radius server. In *Collaborative Technologies and Systems, 2006. CTS 2006. International Symposium on*, pages 74 – 80, may 2006.
14. Pascal Urien, Estelle Marie, and Christophe Kiennert. An innovative solution for cloud computing authentication: Grids of EAP-TLS smart cards. *2010 Fifth Intl. Conf. on Digital Telecommunications*, pages 22–27, 2010.
15. Pascal Urien and Guy Pujolle. TEAPM, trusted EAP modules. In *e-Smart2006*, September 2005.