TYPHON: Um Serviço de Autenticação e Autorização Tolerante a Intrusões

João Sousa, Alysson Bessani, and Paulo Sousa

Laboratório de Sistemas Informáticos de Grande Escala,
Faculdade de Ciências da Universidade de Lisboa
Lisboa, Portugal

Resumo A norma Kerberos v5 especifica como é que clientes e serviços de um sistema distribuído podem autenticar-se mutuamente usando um serviço de autenticação centralizado. Se este serviço falhar, por paragem ou de forma arbitrária (e.g., bug de software, problema de hardware, intrusão), os clientes e serviços que dependem dele deixam de poder autenticar-se. Este artigo apresenta um serviço de autenticação e autorização que respeita a especificação do Kerberos v5 tal como é descrita no RFC 4120, fazendo uso da técnica da replicação da máquina de estados e de componentes seguros para tornar o serviço mais resiliente. A técnica da replicação da máquina de estados utilizada oferece tolerância a faltas arbitrárias, enquanto os componentes seguros garantem que as chaves dos clientes e dos serviços são mantidas secretas mesmo na presença de intrusões. Os resultados de avaliação mostram que a latência e débito do serviço proposto são similares aos de uma concretização de Kerberos bem conhecida.

Abstract. The Kerberos v5 standard specifies how the clients and services of a distributed system may mutually authenticate through the use of a centralized authentication service. If this service fails, by crash or in an arbitrary way (e.g., software bug, hardware problem, intrusion), the clients and services that depend on it are not able to authenticate between themselves. This paper presents an authentication and authorization service that complies with RFC 4120, and that uses Byzantine-fault-tolerant state machine replication and secure components to make the service more resilient. These secure components guarantee that clients' and services' secret keys are kept private even in the presence of intrusions. The evaluation results show that the proposed service has similar latency and throughput values to the ones of a well known Kerberos implementation.

1 Introdução

Hoje em dia, os sistemas informáticos da maioria das organizações utilizam algum tipo de serviço de autenticação e autorização de forma a impôr politicas de controlo de acesso a diferentes tipos de dados e/ou serviços. A norma Kerberos v5 propõe uma especificação para um serviço de autenticação (Kerberos) com duas características interessantes: faz uso de autenticação mediada, sendo portanto escalável do ponto de vista do número de chaves que cada entidade do sistema tem de armazenar, e apenas utiliza criptografia simétrica que é, como se sabe, mais rápida do que criptografia assimétrica.

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 649–660

No entanto, o serviço de autenticação Kerberos é centralizado e concretizações comuns deste serviço tendem a residir apenas em um processo e em uma máquina. Basta que esse processo ou máquina falhe para que o serviço de autenticação se torne indisponível (falha por paragem) ou errático (falha arbitrária). Também basta a um intruso comprometer esse mesmo processo ou máquina para obter todas as chaves de todos os clientes e serviços do sistema. A falha do serviço de autenticação pode forçar todo o sistema informático que depende dele a parar, ou até mesmo permitir a utilização do sistema sem autenticação, dependendo de como o sistema foi concretizado e do tipo de falha do serviço de autenticação. Para além disto, a norma Kerberos v5 só especifica um serviço de autenticação, não fazendo qualquer referência à forma como o controlo de acesso/autorização é efectuado. Na prática, os serviços que concretizam esta norma (e.g., Apache DS, Microsoft Active Directory) combinam o serviço de autenticação Kerberos com um serviço de nomes e directorias (e.g., LDAP) onde é armazenada a política de controlo de acesso.

Este artigo apresenta o serviço de autenticação e autorização TYPHON[1], um serviço que obedece à especificação Kerberos v5, mas que é construído de forma a tolerar intrusões, fazendo uso da técnica de replicação da máquina de estados - para tolerar faltas arbitrárias - e de um componente seguro - para assegurar a confidencialidade das chaves no caso de acontecerem intrusões.

2 A Especificação Kerberos v5

O Kerberos v5 [8] é uma norma especificada no RFC 4120 [9]. Esta por sua vez é concretizada por várias aplicações, como o ApacheDS [1] e o Microsoft Active Directory[2].

Esta norma permite comunicações seguras e identificadas entre duas entidades por cima de uma rede insegura - como é o caso da Internet - e possibilita que duas entidades (tipicamente cliente e serviço) que à partida não têm nada que prove que podem confiar uma na outra, consigam autenticar-se mutuamente.

Para assegurar a confidencialidade dos dados e a autenticidade de clientes e serviços, é usada criptografia simétrica. Todos as entidades partilham uma chave secreta com o Kerberos e este faz de mediador entre as duas entidades que se pretendem autenticar.

A autenticação de uma entidade perante outra é conseguida por intermédio de estruturas de dados produzidas pelo Kerberos, denominadas de *tickets*. Os *tickets* provam que o Kerberos autenticou a entidade que pretende comunicar com outrem.

O Kerberos está dividido em dois componentes lógicos: o *Authentication Service* (AS) e o *Ticket Granting Service* (TGS). O primeiro permite a autenticação de uma entidade perante o Kerberos. O segundo destina-se a mediar a autenticação entre duas entidades após ambas estarem autenticadas perante o Kerberos (AS).

[1] Na mitologia grega, TYPHON é o pai de Cerberos (ou Kerberos).

[2] De notar que o ApacheDS e o Microsoft Active Directory são servidores LDAP que disponibilizam igualmente um serviço de autenticação Kerberos v5.

2.1 Interacções

A figura 1 ilustra[3] as interacções entre um cliente C, os componentes AS e TGS do Kerberos, e um serviço S.

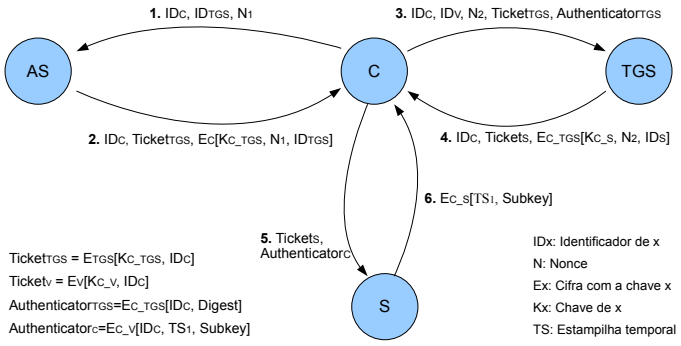


Figura 1. Ilustração simplificada das interacções das entidades presentes no Kerberos v5.

O objectivo das interacções ilustradas na figura 1 é autenticar o cliente C perante o serviço S. No passo 1 o cliente C requisita ao AS um *ticket granting ticket* (TGT). Esse TGT só poderá ser decifrado pelo Kerberos. No passo 2, o cliente obtém o TGT (Ticket_{TGS}) mais a chave de sessão contida nele (K_{C_TGS}) e produz um *authenticator* cifrado com essa mesma chave ($\text{Authenticator}_{TGS}$).

No passo 3, o cliente C envia Ticket_{TGS} e $\text{Authenticator}_{TGS}$ ao TGS, de forma a provar a sua identidade. Também indica o serviço com o qual se pretende autenticar, neste caso S. No passo 4, se o TGS conseguir decifrar $\text{Authenticator}_{TGS}$ com a chave contida em Ticket_{TGS} e encontrar S na sua base de dados, gera um *ticket* para S (Tickets) contendo uma chave de sessão a ser usada entre C e S (K_{C_S}). Por fim, no passo 5 e 6 o cliente C autentica-se perante o serviço S de forma semelhante à que se autenticou ao TGS.

3 Desafios na Concretização de um Kerberos Tolerante a Intrusões

A norma Kerberos foi criada com um pressuposto muito forte: o processo que concretiza a norma e a máquina que o executa nunca são comprometidos. O único perigo está na rede, que não assegura por si só a confidencialidade das mensagens. Mas tal assumpção não pode ser dada como garantida na prática. Isso dá origem a um ponto fraco nesta arquitectura: se o processo que executa o AS/TGS falhar por paragem, torna-se impossível realizar autenticação a partir do momento em que a falha ocorre. Os clientes que já tenham adquirido *tickets* para serviços ainda conseguem dialogar com os serviços para os quais esses *tickets* foram gerados[4], mas não é possível fazer novas autenticações.

[3] Esta ilustração é muito simplificada. Para facilitar a compreensão do protocolo, omitimos detalhes como o uso de *realms*, *flags* e o tempo de validade e renovação dos *tickets*.

[4] Mas não eternamente, porque os *tickets* têm um prazo de validade.

Também é possível que ocorram falhas arbitrárias, como por exemplo falhas de hardware, bugs e intrusões. No caso das intrusões, as chaves secretas podem ser obtidas, e quem se apoderar delas pode vir a personificar clientes e serviços.

O desafio neste trabalho é tornar mais robustas as quatro propriedades de segurança de um serviço Kerberos v5: autenticidade, integridade, confidencialidade e disponibilidade. A propriedade de autenticidade já é oferecida à partida pelo serviço Kerberos v5, uma vez que o seu objectivo é precisamente garantir a autenticidade dos clientes e serviços que o usam. Para garantir as três propriedades de segurança restantes, usamos a técnica de replicação de máquina de estados combinada com um componente seguro local em cada réplica.

4 Técnica de replicação de máquina de estados

Consideremos uma aplicação cliente/servidor, onde o servidor possui um estado, que evolui consoante os comandos enviados pelo cliente. A técnica da replicação da máquina de estados tem por objectivo replicar esse estado em vários servidores [11]. Esses servidores são considerados réplicas.

Os clientes enviam os seus comandos para essas réplicas, e o estado das mesmas deverá evoluir exactamente da mesma forma. Para isso cada réplica tem que começar a sua execução no mesmo estado, executar sobre ele apenas operações deterministas (e.g., não é possível gerar valores aleatórios, nem usar estampilhas temporais) e todos os comandos devem ser entregues na mesma ordem às réplicas - para este último requisito, é necessário uma primitiva de difusão atómica [6].

No entanto, é preciso notar que esta técnica de replicação, apesar de reforçar as propriedades de integridade e disponibilidade, não assegura confidencialidade e autenticidade. Pelo contrário, torna estas propriedades mais fracas. Isto porque ao tornarmos o sistema replicado, aumentamos a probabilidade de algum servidor ser corrompido. Como o estado está replicado em todos os servidores e, no caso do Kerberos o estado corresponde às chaves, basta uma intrusão para que estas sejam obtidas.

5 Protecção das Chaves

Para evitarmos o problema que acabámos de descrever, é necessário garantir que as chaves armazenadas por cada réplica não fiquem acessíveis mesmo quando a réplica for comprometida. No nosso sistema, a protecção das chaves é efectuada através da utilização de um componente seguro [5]. Um componente seguro consiste numa parte do sistema que se assume ser imune a intrusões, mesmo que o resto do sistema seja comprometido. Deverá ter uma especificação curta, de maneira a que a sua concretização seja simples, para ser exequível verificar as suas propriedades funcionais e não funcionais.

O componente seguro permite-nos obter a propriedade de confidencialidade, uma vez que armazena todos os dados confidenciais, i.e., as chaves. Também se delegam a este as operações que processam esses dados. A ideia é guardar os dados confidenciais no componente e executar operações sobre eles sem que estes sejam expostos ao

resto do sistema. Na secção 6.3 será explicado como é que este componente assegura a autenticidade.

6 O Serviço TYPHON

Nesta secção descrevemos em detalhe o serviço TYPHON, um serviço de autenticação e autorização tolerante a intrusões. Este serviço foi construído usando a primitiva de difusão atómica oferecida pela biblioteca BFT-SMaRt [2] e um conjunto de funcionalidades disponibilizadas por um componente seguro designado κ. De notar que conseguimos conceber este serviço sem subverter a especificação do Kerberos v5 tal como é apresentada no RFC 4120.

6.1 Modelo de Sistema

Assumimos um modelo e arquitectura de sistema híbridos em que o sistema é composto por duas partes, com propriedades e pressupostos distintos [12]. Estas duas partes designam-se tipicamente por *payload* e *wormhole*. Assume-se que a parte *payload* é composta por um conjunto de $n \geq 3f + 1$ réplicas e que um máximo de f réplicas podem falhar de forma arbitrária. Na parte *wormhole* executa um componente seguro com as funcionalidades que descreveremos mais à frente. Assume-se que esta parte do sistema é segura por construção, i. e., não pode ser comprometido por intrusos - desde que não exista acesso físico à máquina em que esse componente executa. assumesse que este componente pode falhar por paragem, mas se é só se a réplica associada ao componente estiver comprometida.

A biblioteca BFT-SMaRt executa na parte *payload*, assim como os componentes AS e TGS do serviço TYPHON. Assumimos que esta parte executa sob um modelo de sistema eventualmente síncrono semelhante ao definido em [4] uma vez que a primitiva de difusão atómica oferecida pela biblioteca BFT-SMaRt necessita deste pressuposto. Assumimos também sincronia local, i.e., o tempo máximo de processamento local e a taxa de desvio de cada relógio local são limitados e conhecidos. Estes pressupostos são requeridos pelo serviço de estampilhas temporais do BFT-SMaRt. Este serviço é usado pelo TYPHON tal como será explicado mais à frente.

6.2 Descrição Geral

O AS e o TGS apesar de serem componentes lógicos distintos são executados pelo mesmo processo. Por sua vez este processo é replicado em várias máquinas, e estas fazem uso da técnica da replicação da máquina de estados concretizada pelo BFT-SMaRt.

Se considerarmos que as chaves armazenadas pelo AS e TGS nunca são alteradas, o Kerberos v5 torna-se num sistema *stateless* (i.e., o estado nunca é modificado), e portanto poderia ser concretizado sem o uso da técnica de replicação de máquina de estados. Mas o TYPHON não tem esta característica e requer a manutenção de estado consistente nas suas réplicas por pelo menos três razões: (1) o serviço de autorização pode ser *stateful* na medida em que pode ter em conta autorizações prévias de um cliente

para decidir se este deve ou não aceder um novo serviço; (2) por razões de contabilização (*accountability*) e não-repudiação, o TYPHON armazena um histórico de todos os pedidos recebidos e respectivos resultados, e tal histórico precisa de ser consistente nas diversas réplicas; e (3) como foi mencionado na secção 4, a geração de estampilhas temporais não é uma operação determinista, e existe necessidade de estampilhas consistentes entre as réplicas. Para isso é necessário executar um acordo sobre o valor de cada estampilha que se queira gerar. A biblioteca BFT-SMaRt já disponibiliza essa funcionalidade.

Cada réplica tem acesso a um componente seguro. Esse componente guarda dentro de si as chaves dos clientes e dos serviços e oferece todas as operações essenciais para realizar operações com elas. Cada componente seguro possui ainda uma chave secreta que é usada para cifrar dados, gerar chaves de sessão e criar MACs para uma estrutura de dados especial que são os *ticket-approvals* (TAs).

TAs são estruturas de dados geradas pelo componente seguro de cada réplica para garantir que a geração, renovação e validação de um *ticket* para um serviço é permitida segundo a politica de autorização definida no TYPHON[5]. Quando os componentes TGS das várias réplicas do TYPHON recebem um pedido de *ticket* de serviço, cada réplica pede ao seu componente seguro a geração de um TA e envia esse TA para todas as outras réplicas. Posto isto, cada réplica espera a recepão de $2f + 1$ TAs, incluindo o TA dela própria. Se de entre $2f + 1$ TAs existirem pelo menos $f + 1$ válidos, significa que pelo menos uma réplica correcta está a autorizar a geração do *ticket* para esse serviço, logo o componente seguro de cada réplica pode também gerá-lo. Só são necessários $f + 1$ TAs válidos porque este passo de validação não é mais do que uma operação de leitura, logo, de entre esses $f + 1$ TAs válidos pelo menos um foi necessariamente gerado por uma réplica correcta, o que significa que a politica de autorização permite a geração do *ticket* de serviço.

Para assegurar a sua autenticidade, os TAs incluem um MAC gerado com a chave secreta dos componentes seguros - que é a mesma para todos. Também incluem dentro de si um resumo criptográfico dos parâmetros a serem passados ao componente seguro para a geração do *ticket* de serviço, de forma a assegurar que um conjunto de $f + 1$ TAs válidos só pode ser usado para gerar o *ticket* de serviço a que estão associados. Dentro de cada TA também está incluído uma estampilha temporal que evita ataques por repetição.

As chaves de sessão geradas por κ são aleatórias na medida em que são usadas as estampilhas temporais dos TAs e o valor da chave secreta de κ para criar essas chaves. As estampilhas garantem que o valor gerado é sempre diferente do anterior - isto é importante para não se gerar duas chaves com o mesmo valor. A chave de κ garante que esse valor não se consegue prever, pois a chave está dentro de κ e é secreta, logo, tornasse inexequível prever que valores vão ser gerados.

Todas as funcionalidades que não necessitem de realizar operações com/ou sobre as chaves são delegadas ao AS e TGS, como por exemplo definição de flags, verificação de realms, cálculo da validade dos *tickets* a serem emitidos, e outras operações semelhantes.

[5] Tal politica pode consistir simplesmente numa matriz de acesso que associa a cada cliente quais os serviços que este tem autorização para aceder.

Finalmente, existem três alternativas para integrar autorização num serviço kerberos, tal como especificado em [7]: no TGS, num serviço à parte, ou nos serviços destino. A terceira alternativa oferece mais flexibilidade, permitindo que cada serviço tenha a sua forma de controlo de acesso. Esta alternativa é de facto mais atraente num sistema super-distribuído com serviços de vários tipos e com diferentes níveis de criticidade. Mas no TYPHON estamos a assumir a primeira alternativa, que fará sentido num sistema distribuído homogéneo, e é esse tipo de sistema que estamos a assumir. Além disso, o controlo centralizado facilita a gestão da política de segurança do sistema.

6.3 Componente Seguro κ

Na concepção de um componente seguro é necessário ter em consideração o seguinte: (1) a quantidade de código no componente (é mais fácil verificar a sua correcção); (2) a interface do componente (é mais fácil usar e verificar que essa interface não pode ser usada para fazer acções erradas); e (3) a quantidade de vezes que o componente é acedido (porque como reside numa máquina separada, é tipicamente lento acedê-lo).

Tendo estas regras em consideração, concretizámos o componente seguro κ, que o AS e TGS devem utilizar sempre que precisarem de fazer operações que envolvam as chaves dos clientes e/ou serviços. Existe uma instância de κ por cada réplica do sistema. κ foi concebido para oferecer as operações mínimas necessárias a executar sobre *tickets* que envolvam as chaves. κ também tem uma chave secreta - guardada dentro de κ e só conhecida por ele - que usa para cifrar chaves de sessão quando são devolvidas para o resto do sistema. Desta maneira não é necessário guardá-las em κ para preservar a sua confidencialidade. Essa chave secreta também é usada para gerar os MACs dos TAs. Como tal chave está armazenada dentro de κ e este não pode ser comprometido, essa chave mantém-se segura. As operações de κ estão expostas na tabela 1.

Método	Argumentos	Retorno	Sumário
makeTicketApproval	*timestamp, client_name, server_name, hash_request*	*ticket_approval*	Gera *ticket-approvals*.
encryptParts	*enc_sessionKey, ticket_data, reply_data, ticket_approvals*	*ticket, reply_enc_part*	Gera o *ticket* e a parte cifrada da resposta do kerberos.
decryptParts	*ticket, authenticator, server_name*	*ticket_data, auth_data, enc_sessionKey*	Decifra o *ticket* e o *authenticator* enviados pelo cliente.
decryptPreAuth	*PrincipalName, pa_data*	*byte_array*	Decifra a pré-autenticaçao do cliente.

Tabela 1. Operações disponibilizadas por κ em cada réplica.

A operação *makeTicketApproval* gera um *ticket-approval* (TA) contendo a estampilha temporal, nome de cliente que requisita o *ticket*, nome de serviço ao qual o *ticket* se destina, e o resumo criptográfico passados à função. Estas estruturas de dados foram

explicadas na secção 6.2. Nesta função é usada a chave secreta de κ para produzir o MAC de cada TA emitido.

A operação *encryptParts* é usado na geração de *tickets* e na renovação/validação dos mesmos. Também precisa de receber *2f + 1* TAs gerados pelos componentes κ das outras réplicas, e pelo menos *f + 1* desses TAs precisam de ser válidos para se poder gerar o *ticket*.

A operação *decryptParts* é essencial para decifrar *tickets* e *authenticators*. A chave de sessão contida no *ticket* passado como argumento é devolvida cifrada com a chave secreta de κ, de forma a evitar que seja guardada dentro dele.

Finalmente, a operação *decryptPreAuth* serve para verificar a pré-autenticação do cliente.

6.4 Interacções

O algoritmo do TYPHON cumpre a especificação do Kerberos v5 e define um conjunto de interacções adicionais com um componente seguro. Ou seja, as interacções entre clientes e serviços, assim como as interacções entre clientes e AS/TGS continuam iguais ao Kerberos v5. No entanto, AS e TGS têm de contactar κ sempre que efectuem operações relacionadas com as chaves (confidenciais) de clientes e serviços. A figura 2 ilustra estas interacções.

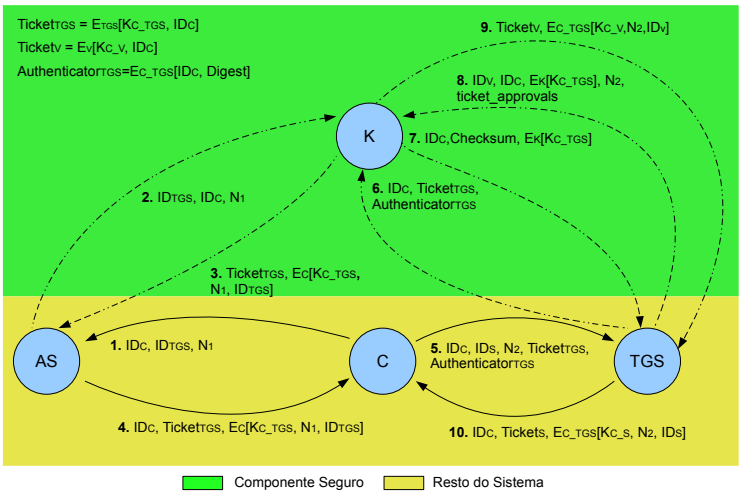


Figura 2. Ilustração simplificada das interacções das entidades presentes no TYPHON. A interacção com o serviço foi omitida desta figura porque é igual à da figura 1. A pré-autenticação também foi omitida por se tratar de um passo opcional segundo o RFC 4120.

No passo 1, C envia um pedido ao AS para obter um TGT. No passo 2, o AS reenvia os parâmetros desse mesmo pedido para κ, para que este crie um TGT e o tuplo cifrado a enviar a C.

No passo 3, após criar a chave de sessão (K_{C_TGS}), κ devolve ao AS o TGT ($Ticket_{TGS}$) e o tuplo cifrado ($E_C[K_{C\_TGS}, N_1, ID_{TGS}]$). No passo 4, o AS devolve a C o TGT e o tuplo.

No passo 5, C envia um pedido ao TGS para obter um *ticket* para S. No passo 6, o TGS re-envia para κ o ID de C (ID_C), o TGT ($Ticket_{TGS}$) e o *authenticator* ($Authenticator_{TGS}$). No passo 7, κ decifra estas duas estruturas e devolve o seu conteúdo ao TGS, com o cuidado de cifrar a chave de sessão contida no TGT com a sua própria chave secreta ($E_K[K_{C\_TGS}]$) - pois se o TGS for comprometido por um agente malicioso, este teria acesso a essa chave.

Após o TGS terminar as operações sobre os dados do TGT/*authenticator* e calcular novos dados de acordo com o RFC 4120, irá invocar κ para este criar um *ticket* para S. Esta interacção está representada nos passos 8 e é semelhante àquela que cria o TGT, com a diferença de que é especificado o identificador de S em vez do identificador do TGS, e também é fornecido $E_K[K_{C\_TGS}]$ de forma a que κ consiga criar $E_{C\_TGS}[K_{C\_V}, N_2, ID_V]$. Também é neste passo que todas as réplicas do sistema geram *ticket-approvals* e os enviam para as outras, para provar a κ que C tem autorização para interagir com S e por isso o *ticket* pode ser gerado.

No passo 9, κ devolve ao TGS o *ticket* para S (ID_S) e o tuplo cifrado que contém a chave de sessão entre C e S ($E_{C\_TGS}[K_{C\_S}, N_2, ID_S]$). No passo 10, estes são finalmente entregues pelo TGS a C. A partir daqui, a interacção entre C e S é igual à do Kerberos normal.

As interacções apresentadas aqui visam preservar a confidencialidade das chaves. Como κ é um componente seguro, as chaves nunca serão reveladas ao exterior, mesmo que no limite todas as réplicas sejam comprometidas. Isto porque κ não tem nenhuma operação que devolva as chaves. Os únicos dados confidenciais que o κ devolve são chaves de sessão, que vêm cifradas com a chave secreta deste. Se o tempo de utilidade dessas chaves (de sessão) for mais curto que o tempo que é necessário para quebrar a sua cifra através de um ataque de força bruta, também não é possível que intrusos consigam obter essas chaves em tempo útil. Também não se consegue gerar *tickets* para serviços arbitrariamente se o sistema for comprometido, devido ao uso de *ticket-approvals*, tal como explicado nas secções 6.2 e 6.3. Desta forma, conseguimos reforçar as propriedades de confidencialidade e autenticidade.

Note-se que os TAs não seriam necessários se estivéssemos a assumir que o TYPHON seria apenas um serviço de autenticação, deixando de lado a autorização. Os TAs servem para provar que existe pelo menos uma réplica correcta a fabricar esses *tickets*, o que significa que o cliente que requisitou esse *ticket* tem permissão para contactar o serviço que deseja. Mesmo que um intruso invadisse uma réplica e usasse κ para produzir *tickets* falsos em nome de outro cliente, este não conseguiria obter a chave de sessão que deve ser usada para fabricar o *authenticator* que deverá ser apresentado ao serviço para o qual se quer autenticar. Como não teria um *authenticator* para apresentar, não conseguiria fazer-se passar pelo cliente legítimo. Também não conseguiria usar ataques de repetição, pois a própria especificação do Kerberos está feita para se defender disso.

7 Avaliação

Nesta secção comparamos a latência e o débito de TYPHON e ApacheDS. O ApacheDS é um serviço de nomes e directorias, mantido pela Apache, que concretiza especificações como LDAP e Kerberos v5.

O TYPHON e o ApacheDS estão ambos escritos na linguagem Java. Como já foi referido, o TYPHON usa a bibliteca BFT-SMaRt, que também está escrita em Java. O TYPHON tem 2943 linhas de código (não contando as 7557 linhas de código da biblioteca BFT-SMaRt) e apenas 8,7% são executadas pelo componente seguro.

O nosso ambiente de testes consistiu em um conjunto de 5 máquinas interligadas por um switch gigabit. Para avaliar o TYPHON, replicá-mo-lo em quatro máquinas ($n = 4$) de forma a tolerar uma falta ($f = 1$) em alguma dessas 4 máquinas. A restante máquina executa todos os processos cliente, que estão constantemente a pedir um TGT seguido de um *ticket* de serviço. O ApacheDS foi avaliado de forma semelhante, com apenas numa máquina a executar o serviço e tendo um cliente noutra máquina a enviar-lhe pedidos da mesma forma que no caso do TYPHON.

Efectuámos dois tipos de experiências. Na primeira avaliámos a latência fim-a-fim da geração de um TGT seguido da geração de um *ticket* de serviço. Na segunda experiência medimos a quantidade de TGTs e *tickets* de serviço que se conseguem gerar por segundo. Estas experiências foram efectuadas em ambos os sistemas e os resultados são descritos na próxima secção.

7.1 Latência

Os resultados dos testes de latência são exibidos na tabela 2. Para cada sistema, foram efectuados 20.000 pedidos de TGTs e *tickets* de serviço, tendo-se calculado a média da latência dos últimos 10.000 pedidos. Os primeiros 10.000 pedidos foram utilizados para forçar a máquina virtual Java a fazer uso do compilador JIT (*Just-In-Time*) para transformar em código máquina as instruções do ApacheDS e TYPHON que tratam dos pedidos de TGTs e *tickets* de serviço.

TYPHON		ApacheDS	
TGT	*Ticket* de serviço	TGT	*Ticket* de serviço
4.6	5.7	26.3	3.0

Tabela 2. Resultados dos testes de latência para ambos os sistemas (em milisegundos).

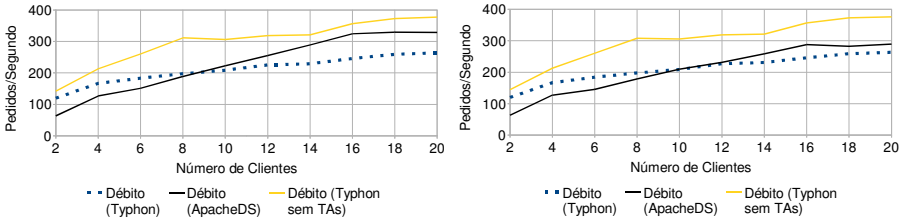
Os resultados da latência dos *tickets* de serviço mostram que o TYPHON é mais lento a realizar essa operação do que o ApacheDS. Por outro lado, podemos observar que o ApacheDS é bastante mais lento do que o TYPHON na geração de TGTs. Este resultado é inesperado, mas sabemos que enquanto o TYPHON guarda todos os seus dados em memória depois de ser inicializado, o ApacheDS vai buscá-los ao disco. Isto explica que a latência do ApacheDS seja maior que a do TYPHON[6].

[6] No entanto, após contactar a equipa de desenvolvimento do ApacheDS, ficámos a saber que este devia guardar essa informação numa *cache* em memória. Eles próprios fizeram os seus testes de latência e confirmaram os nossos resultados, mas ainda precisam de investigar o que está a acontecer que explique este fenómeno.

A diferença entre os valores de latência do TYPHON para TGTs e *tickets* de serviço é explicada pelo facto dos TGTs não requererem a utilização de TAs, enquanto que os *tickets* de serviço requerem. A latência média do envio e recepção de TAs corresponde portanto a 1.1 ms, a diferença entre a latência dos *tickets* de serviço (5.7 ms) e a latência dos TGTs (4.6 ms). No entanto é necessário ter em consideração que numa situação real o TYPHON teria que gerar mais *tickets* de serviços do que TGTs, e o ideal seria que fossem os *tickets* de serviço a ter uma latência menor.

7.2 Débito

Para computar o débito, calculámos o número de pedidos por segundo processados por ambos os sistemas em intervalos de 1000 pedidos. Como a escalabilidade pretendida é em termos de clientes, aumentámos progressivamente o número de clientes de forma a observar o ponto de saturação de cada sistema. A figura 3 reporta os resultados.



(a) Débito da geração de TGTs.

(b) Débito da geração de *tickets* de serviço.

Figura 3. Resultados dos testes de débito para os serviços ApacheDS e TYPHON (com e sem *ticket-approvals*).

Os gráficos mostram um comportamento semelhante tanto no caso de TGTs como de *tickets* de serviço, e sugerem que quando se trata de poucos clientes a enviar em simultâneo, o TYPHON mostra melhor comportamento que o ApacheDS. Mas a partir de 10 clientes, o TYPHON estagna, e o ApacheDS passa a apresentar melhores resultados. No entanto, a partir de 20 clientes também o ApacheDS estagna, apesar de se manter mais eficiente que o TYPHON.

Finalmente, fizemos um último teste retirando o uso de TAs da concretização do TYPHON, tornando-o somente num serviço de autenticação. Observámos que a remoção deste passo faz com que o sistema se torne mais eficiente a processar pedidos, chegando ao ponto de mostrar melhores desempenhos que o ApacheDS.

8 Trabalhos Relacionados

No passado foram propostos alguns serviços de segurança tolerantes a intrusões (e.g., autoridade de certificação COCA [13], serviço de gestão de chaves Ω [10], firewall aplicacional CIS [3]). No entanto, tanto quanto é do nosso conhecimento, este trabalho é o primeiro a apresentar um serviço de autenticação e autorização tolerante a intrusões que cumpre a norma Kerberos v5.

9 Conclusão

O Kerberos v5 é uma norma que especifica como clientes e serviços se devem autenticar mutuamente por intermédio de uma entidade centralizada que guarda as chaves de todos os participantes. O problema reside na possibilidade dessa entidade centralizada falhar, quer seja por paragem, arbitrariamente ou até por intrusão. Se isso acontecer não é possível fazer mais autenticações de clientes ou serviços.

Neste artigo apresentámos o serviço TYPHON, um serviço de autenticação e autorização que segue a especificação do Kerberos v5 ao mesmo tempo que introduz na sua concretização mecanismos para tolerar intrusões. Por um lado, usa a técnica de replicação da máquina de estados para oferecer tolerância a faltas arbitrárias. Por outro lado, faz uso de componentes seguros para guardar as chaves dos clientes e dos serviços de forma a assegurar que estes não são expostos na eventualidade de intrusões.

Finalmente, os resultados da avaliação do serviço TYPHON mostram que o seu desempenho é similar ao do ApacheDS, um serviço de autenticação e autorização não replicado que concretiza a norma Kerberos v5.

Agradecimentos. Este trabalho é suportado pela FCT através de seu programa multianual (LaSIGE) e através dos projectos PTDC/EIA-EIA/100581/2008 (REGENESYS) e CMU-Portugal (CMU-PT/0002/2007).

Referências

1. ApacheDS - An embeddable directory server entirely written in Java. `http://directory.apache.org/`.
2. BFT-SMART - High-performance Byzantine fault-tolerant State Machine Replication. `http://code.google.com/p/bft-smart/`.
3. A. Bessani, P. Sousa, M. Correia, N. F. Neves, and P. Verissimo. The CRUTIAL way of critical infrastructure protection. *IEEE Security & Privacy*, 6(6):44–51, Nov-Dec 2008.
4. M. Castro and B. Liskov. Practical Byzantine fault-tolerance and proactive recovery. *ACM Transactions Computer Systems*, 20(4):398–461, Nov. 2002.
5. M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Proc. of the 4th European Dependable Computing Conference*, Oct. 2002.
6. V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical report, Cornell University, May 1994.
7. S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Section E.2.1 kerberos authentication and authorization system, Apr. 13 1989.
8. B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, Sept. 1994.
9. C. Neuman, S. Hartman, and K. Raeburn. The kerberos network authentication service (v5). Internet Engineering Task Force RFC 4120, July 2005.
10. M. K. Reiter, M. K. Franklin, J. B. Lacy, and R. N. Wright. The Ω key management service. In *Proc. of the 3rd ACM Conf. on Computer and Communications Security*, 1996.
11. F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22, Dec. 1990.
12. P. Verissimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1):66–81, 2006.
13. L. Zhou, F. Schneider, and R. Van Rennesse. COCA: A secure distributed online certification authority. *ACM Transactions Computer Systems*, 20(4):329–368, Nov. 2002.