Software Defined Perimeter: improvements in the security of Single Packet Authorization and user authentication

Everson L. Rosa Lucion
Pós-Graduação em Ciência da Computação (PGCC)
Universidade Federal de Santa Maria - RS, Brasil
everson@cpd.ufsm.br

Raul Ceretta Nunes
Pós-Graduação em Ciência da Computação (PGCC)
Universidade Federal de Santa Maria - RS, Brasil
ceretta@inf.ufsm.br

***Abstract*—Traditional perimeter defense is typically being performed through dedicated firewall-based devices. However, it becomes necessary to decrease the surface and exposure to cyber attacks by hiding the infrastructure, applications and access controls, as well as increasing security levels. Software Defined Perimeter (SDP) brings new perimeter functionality and Single Packet Authorization (SPA) is the first step. Through the analysis of the SDP protocol there were security issues that need to be improved or addressed. This work proposes adaptations in the SDP architecture and definition of a new pattern of creation and sending of the SPA. It was designed under modular aspects that are incorporated into the model. A secure way to establish mutual TLS for initial user authentication has also been developed. The results demonstrate that building security solutions in modules greatly increases the degree of difficulty in detecting, replicating or reading data. Through the experiments it was demonstrated that the increase of the processing time of the SPA does not compromise the proposed solution and is justified by the gains in the levels of protection. The definition of a new SPA submission architecture and establishment of mutual TLS provided the concealment, scalability and redundancy desired. The proposed solutions thus contribute to increasing the levels of resilience and protection of the SDP reference standard.**

***Index Terms*—Cloud Security, Software Defined Perimeter, Single Packet Authorization, Authentication.**

I. Introdução

Normalmente, a defesa de perímetro é realizada através de um dispositivo dedicado que controla o acesso à rede privada ao permitir ou negar fluxos de tráfego com base nas diretivas de segurança de uma organização [1]. O modelo de perímetro baseado em *firewall* é comparado a um castelo medieval, cercado com paredes grossas e um único ponto de entrada. Tudo que estiver fora é considerado perigoso e tudo que estiver dentro é confiável. Mas a segurança que as barreiras proporcionam no mundo físico não se traduz no ciberespaço. Com a disseminação da computação móvel, o uso de diferentes dispositivos e crescente utilização de serviços baseados em nuvem, surgem vetores de ataques adicionais [2].

Procurando oferecer aos proprietários de aplicativos a capacidade de implantar funcionalidades perimetrais, a *Cloud Security Alliance*[1] (CSA) lançou a Especificação 1.0 [3] do protocolo que descreve o Perímetro Definido por Software (do inglês *Software Defined Perimeter* - SDP) e a autorização por um único pacote (do inglês *Single Packet Authorization* - SPA) é o primeiro passo para isso. A sua concepção inclui a ocultação da infraestrutura, de serviços e controles de acesso, bem como a confidencialidade e integridade das comunicações. O SDP permite a redução de riscos ao diminuir a exposição a ataques cibernéticos. O protocolo surgiu em resposta à urgência de atualizar o estado atual da segurança cibernética, dando início a um novo conceito de arquitetura de segurança [3].

Através da análise da Especificação 1.0 [3] verificaram-se questões de segurança do SPA que precisam ser melhoradas ou abordadas durante os processos de criação e envio: O SPA deve atender ao princípio de pacote único; os campos do *payload* são transportados sem criptografia; o uso da função HOTP [4] para formação da senha única a torna válida por tempo indeterminado; questões relacionadas ao uso do IPv6 não foram tratadas; a senha secreta do usuário usada não oferece proteção contra ataques de força bruta ou dicionário; a verificação de integridade dos campos do *payload* não foi implementada; a autenticidade de dados deveria ser aplicada a todo *payload;* ações que atendam a confidencialidade de tráfego não foram previstas. E por fim, o problema da conexão TCP subsequente é um problema que ainda persiste em técnicas de autenticação via SPA.

Este trabalho propõe adequações na arquitetura SDP descrita pela Especificação 1.0 [3] e uma nova abordagem projetada sob aspectos modulares de criação e envio do SPA. Os módulos de originalidade e autenticidade foram fortalecidos e os de pontualidade, integridade e confidencialidade de dados, criptografia da senha do usuário, compatibilidade com o protocolo IPv6 e confidencialidade de tráfego foram abordados. Desenvolveu-se, também, uma forma segura de estabelecer a conexão TCP subsequente e o TLS mútuo para autenticação inicial de usuários, tornando-a mais robusta.

[1]https://cloudsecurityalliance.org/

Os resultados experimentais mostram que o aumento do tempo de processamento de um SPA se justifica pelos ganhos relevantes dos níveis de proteção. A análise de segurança demonstra o fortalecimento e aumento significativo da proteção contra ameaças, vulnerabilidades ou ataques. Nesse sentido, a adição dos módulos propostos são viáveis. O transporte do campo *timestamp* e *counter* dentro do *payload* SPA reduz até 90 vezes o tempo de validação de senhas. A derivação de chaves criptográficas e a junção dos protocolos HOTP e TOTP fortalecem exponencialmente a segurança das senhas. A definição de uma nova arquitetura para envio do SPA e estabelecimento do TLS mútuo propiciam a ocultação, escalabilidade e redundância desejadas.

Este artigo está organizado da seguinte forma. Os trabalhos relacionados são discutidos na seção 2. Na sequência, a seção 3 apresenta os conceitos necessários para a compreensão do trabalho. Posteriormente, a seção 4 descreve os detalhes da arquitetura proposta. A seção 5 relata os experimentos e resultados do sistema proposto. Por fim, a seção 6 evidencia a discussão e a seção 7 traz as conclusões e trabalhos futuros.

II. Trabalhos Relacionados

Esta seção retrata o estado atual das pesquisas em defesa de perímetro e SPA.

Em [5] os autores relatam que a falta de associação entre o processo de autenticação e a conexão TCP subsequente é um problema que ainda persiste em ambas as técnicas de autorização passiva (SPA e *port knocking*[2]). Esse problema permite que um invasor se conecte a um servidor protegido em nome de um cliente válido, após o cliente ter autenticado com sucesso no *firewall*, mas antes de estabelecer uma conexão TCP com o servidor. Os autores criaram uma solução que compartilha um *nonce*[3] criptografado entre o cliente e o *firewall* durante o envio do SPA. O *firewall* somente aceitaria tentativas de conexões, caso o primeiro TCP SYN tivesse o *nonce*. Os campos IP *timestamp* e TCP *echo* foram usados para codificar o *nonce* e enviá-lo ao *firewall*. O *buffer* do *timestamp* deve ser marcado como *full* de modo que os roteadores intermediários não manipulem o seu valor.

Em [6] os autores relatam que até a realização de seu trabalho, somente duas implementações usam *One Time Password* (OTP). A primeira, o *Cerberus*[4] usa como OTP o *timestamp* (até precisão de minuto). A segunda, *COK*[5] aplica *hash* na senha do usuário iterativamente para criar uma cadeia de saídas. Os autores, também, notaram que não há garantia que o cliente que se conecta à porta aberta seja o mesmo cliente autenticado na porta, pois não existe ligação entre eles. Propuseram um *framework* que envia OTP via SMS ao usuário usando *Pseudo-Random Number Generator* (PRNG) com o *timestamp* e uma porta randômica usada no acesso. A derivação de chaves também foi usada. A senha recebida é usada para adicionar confidencialidade e calcular MAC[6] do SPA. Também é criado o túnel IPSec VPN com os parâmetros passados via SPA. É relatado que pode haver limitações pontuais no projeto devido a possíveis modificações no protocolo IPSec.

No trabalho de [7] o cliente envia um pacote TCP com as *flags* SYN/FIN/RST contendo os dados da validação criptografados. Usa o *timestamp* e um número randômico para prevenir *replay attacks*. Relata que seu trabalho resolve dois problemas: 1. ataque de reserva de recursos (através de atributo do SPA indicando porta do próximo SPA); 2. falta de associação entre o processo de autenticação e o processo de estabelecimento de seção (por meio de porta informada pelo SPA).

Em [8] é proposto um mecanismo nomeado de QUICKKNOCK, melhorando as potencialidades de tecnologias como *port knocking* e SPA usando *firewall* e criptografia. Com *steganografia*, usa os campos de número de sequência e *timestamp* do *header* TCP SYN para embutir o código *hash* resultante da autenticação. Se a verificação for bem-sucedida, o servidor permitirá que a conexão continue, caso contrário, o pacote será descartado. Há uma limitação de 32 *bits* e, isso, limita o envio do *hash* completo, mantido às claras. O cliente e o servidor compartilham uma chave, bem como um contador (usado como *nonce*) que é incrementado para cada conexão do cliente. O SPA e a conexão TCP subsequente são tratados em um único pacote enviado.

Os estudos [5, 6, 7, 8] propuseram soluções para o estabelecimento da seção TCP subsequente (após o envio do SPA). O trabalho de [8] sugeriu um SPA único com TCP SYN e *hash* encapsulado ao *header* TCP. Já [6] propôs um canal *out-of-band* para receber dados para envio do SPA e criação da VPN subsequente. O estudo de [5] envia um *nonce* via SPA, onde o servidor somente aceitará o TCP SYN caso contenha o *nonce.* Em [7] o cliente envia dentro do SPA a porta em que irá conectar via TCP.

A pontualidade foi abordada em [6] e a originalidade em [5, 6, 7, 8]. A integridade de dados foi fortemente usada em [7] e parcialmente [5, 6, 8]. A autenticidade foi bem abordada em [5, 6, 7, 8]. A derivação de chaves foi somente usada em [6]. Não houve considerações sobre a compatibilidade com o protocolo IPv6 e a confidencialidade de tráfego nos trabalhos.

Analisando os trabalhos [5, 6, 7, 8], observou-se que os autores não abordaram todos os aspectos de segurança de forma equânime. Tal observação foi fundamental para adoção dos objetivos e metodologia nesse trabalho. Abordou-se o SPA com funcionalidades sólidas de pontualidade, originalidade, integridade e autenticidade de dados, derivação de chaves, compatibilidade com o protocolo IPv6 e confidencialidade de dados e de tráfego. Criou-se uma arquitetura capaz de ocultar a conexão TCP subsequente.

[2]Tentativa de conexão em uma sequência de portas predeterminada.

[3]Palavra de uso único (n = *Number* e *once* = Uma vez, em inglês), o N pode usar letras também. Disponível: https://pt.wikipedia.org/wiki/Nonce.

[4]http://silverstr.ufies.org/blog/Cerberus.ppt

[5]http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-worth-up.pdf

[6]Autenticador de mensagem (em inglês, *message authentication code*).

III. Referencial Teórico

A. O Perímetro Definido por Software

Atualmente, o perímetro já não é apenas a localização física da empresa; e o que está dentro já não é um lugar protegido e seguro para hospedar dispositivos e aplicativos pessoais e corporativos. O Perímetro Definido por Software é uma arquitetura de segurança desenvolvida por membros da *Cloud Security Alliance*. O documento inclui a ocultação da infraestrutura e aplicativos, dos controles de acesso bem como a confidencialidade e integridade das comunicações [3]. No SDP, o **plano de controle** é separado do **plano de dados** para permitir um sistema completamente escalável, conforme ilustra a Figura 1.

Na arquitetura SDP, o esquema de controle é definido pelo cliente ou *host* de iniciação (Iniatiating Host - IH) e pelo *host* de aceitação (Accepting Host - HA). Nesse plano, ambos comunicam-se com o *Controller*. Já o plano de dados descreve a forma de comunicação entre o IH e o HA.

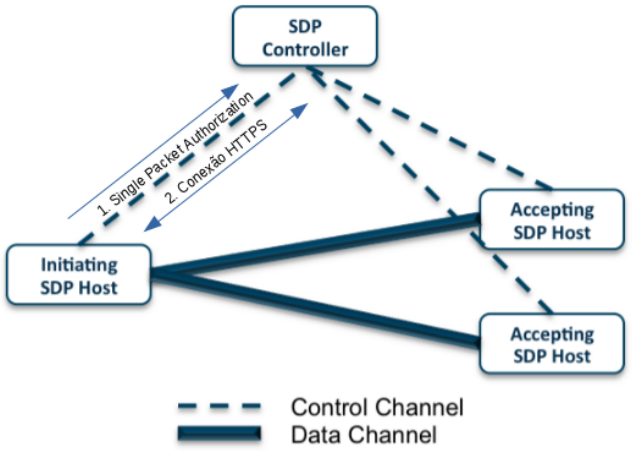

Fig. 1. Arquitetura SDP: 1. envio de um SPA do cliente ao *Controller*. 2. estabelecimento da conexão TLS subsequente. Adaptado de [3].

B. Single Packet Authorization

Desde dezembro de 2005, o SPA vem sendo usado para proteger as comunicações baseadas em IP. O conceito foi amplamente estendido e aprimorado desde então por [9]. O SPA requer apenas um único pacote criptografado para comunicar várias informações, incluindo o acesso desejado através de uma política de *firewall* e/ou enviar comandos para executar no sistema de destino.

Geralmente em um mecanismo SPA, são usados os protocolos UDP, TCP ou ICMP. O pacote único é enviado a uma porta destino específica. O servidor detecta o pacote, valida os dados incluídos dentro do *payload* e solicita ao serviço de *firewall* incluir o IP e porta informados em suas regras de permissão. A regra IP/porta estará ativa por um período pré-definido entre as partes, geralmente de 30 segundos. Logo após, o dispositivo cliente pode conectar-se ao servidor [7].

C. Função de Derivação de Chaves

Uma Função de Derivação de Chaves (do inglês *Key Derivation Funciton* - KDF) é usada para gerar uma ou mais chaves criptográficas a partir de uma chave mestra (senha secreta). A derivação de chaves criptográficas é usada para garantir a segurança da informação e proteger dados eletrônicos quando eles estão guardados ou sendo transmitidos [10].

A *Password-Based Key Derivation Function 2* ou função PBKDF2 - RFC 2898 [11] é usada para derivar chaves de suficiente tamanho. A PBKDF2 pode usar a função HMAC-SHA-256 - RFC 6234 [12] como *Pseudorandom Function* (PRF). A função HMAC-SHA-256 gera uma saída de 32 *bytes,* apesar de ser um algoritmo de hash mais lento, oferece mais proteção contra ataques de força bruta. Abaixo, descreve-se a função de derivação de chaves PBKDF2:

DK = PBKDF2(HMAC-SHA-256, *Password, Salt*, c, *dkLen*)

- *dklen*: tamanho desejado da chave derivada em *bytes*;
- DK: chave derivada com comprimento de dkLen *octetos*;
- PBKDF2: algorítimo proposto pela RFC 2898;
- HMAC-SHA-256: *Pseudorandom Function* usada;
- *Password*: chave mestra ou senha do usuário;
- *Salt*: caracteres incorporados ao *Password* e simétricos entre o cliente e o *Controller*, conhecido como salto criptográfico e de conhecimento público;
- c: número de iterações desejado.

D. On Time Password

Um atacante poderia capturar e retransmitir uma cópia de um SPA. Nesse sentido, os sistemas poderiam ser alvos de ataques de repetição. Uma solução possível seria a utilização de senhas de uso único (e.g., *One Time Password* - OTP) [13]. Na Especificação 1.0 [3] utiliza-se a função HOTP: *HMAC-Based One-Time Password* - RFC 4226 [4] que fornece leve grau de originalidade. Essa função é baseada em um contador mantido sincronizado entre o *Controller* e o cliente. Não há orientação para manter essa variável secreta durante o envio do SPA.

O protocolo HOTP não impediria que o atacante interceptasse o SPA por um tempo e depois o transmitisse. Além disso, caso uma senha HOTP seja comprometida, será potencialmente válida por um tempo indeterminado. O protocolo TOTP: *Time-Based One-Time Password* - RFC 6238 [14] normatiza o uso de senhas baseado no tempo humano e usa o relógio do sistema para criar e validar OTP. Cada implementação pode definir a tolerância de validade da senha ou considerar uma leve dessincronização dos relógios [15].

E. Integridade e Autenticidade de Dados

Um SPA vindo de um cliente pode ter seu conteúdo modificado depois que ele o criou. Integridade e autenticidade são, de certa forma, inseparáveis [16]. Um autenticador é um valor a ser incluído em um SPA transmitido, que pode ser usado ao mesmo tempo, para

verificar autenticidade e integridade dos dados do SPA. Um autenticador pode combinar confidencialidade de dados e uma função de *hash* criptográfico HMAC (*Hash-based Message Authentication Code*). Ao contrário da função *hash* que fornece apenas verificação de integridade, um MAC também fornece verificação de autenticidade, assumindo que a chave secreta é conhecida apenas por entidades confiáveis [6].

IV. Metodologia

Esta seção explora as características apresentadas na seção 3 e apresenta novos métodos para construção do SPA e estabelecimento de conexão TCP subsequente entre o *Iniatiating Host* (cliente) e o SDP *Controller.*

A seção está organizada como segue. A seção 4-A descreve um novo modelo para criação de SPA modular e a seção 4-B apresenta uma nova arquitetura para resolver o problema da conexão TCP subsequente de modo a estabelecer o TLS mútuo visando a autenticação inicial do usuário.

A. Criação de um SPA

A *Internet* é um ambiente particularmente compartilhado, sendo usada por empresas concorrentes, governos mutuamente hostis e criminosos oportunistas. Para que os dados transmitidos não sejam comprometidos segundo [16] é recomendável abordar os seguintes aspectos de segurança: confidencialidade de dados e de tráfego, integridade e autenticidade, originalidade e pontualidade.

O envio do SPA é o primeiro passo para acesso a um SDP. A Figura 2 ilustra como o SPA é abordado pela Especificação 1.0 [3]. Observa-se que o resultado do algorítimo HOTP gera um valor relativo ao campo *Password*. AID é um valor que identifica um cliente. O protocolo TCP é definido como protocolo de transporte e o servidor não deve responder a solicitação. Já o *Counter* (C) introduz o conceito de *One Time Passwords* baseado em eventos e o (K) é a senha simétrica compartilhada entre o cliente e o *Controller.*

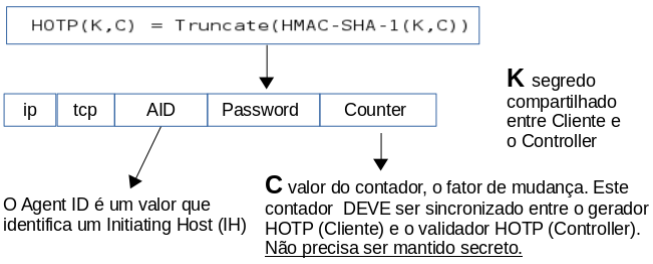

Fig. 2. Formato do pacote SPA definido pela Especificação 1.0. Adaptado de [3].

Com base no modelo proposto pela CSA (Figura 2), este trabalho propõe adequações na arquitetura visando aumento dos níveis de segurança. A construção de soluções de segurança em módulos podem aumentar consideravelmente o grau de dificuldade de detecção, replicação ou leituras dos dados em um ambiente hostil como a *Internet.* Neste sentido, incorporou-se ao modelo seis módulos propostos por [16] e acrecentou-se um padrão de derivação de chaves, e compatibilidade com o protocolo IPv6. A seguir, os oito módulos são descritos.

1) O Módulo de Derivação de Chaves: para evitar a exposição da chave mestra durante a transmissão do SPA é proposto o uso de funções de derivação de chaves (seção 3-C). Essa solução torna exponencialmente mais difícil a quebra de senhas. Com o uso da função PBKDF2, a senha mestra torna-se uma chave criptografada. Duas chaves derivadas são geradas, nomeadas de *Key1* e *Key2*.

De acordo com [11], o número mínimo recomendado de iterações para uso na função PBKDF2 é 1000. Logo, utilizou-se 1000 iterações para a *Key1* e 2000 iterações para *Key2*. Porém esses valores podem ser alterados conforme o poder de processamento de cada ambiente.

A derivação de chaves ***Key1*** teve seu uso aplicado no módulo originalidade, autenticidade e integridade de dados conforme a equação:

$$Key1^{(1000)} = (HmacSha256(pass \parallel salt1 \parallel 32))$$

No módulo de confidencialidade de dados, usou-se a derivação de chaves **Key2** como visto na equação:

$$Key2^{(2000)} = (HmacSha256(pass \parallel salt2 \parallel 32))$$

2) Módulo Pontualidade: um protocolo que detecta técnicas de atraso fornece pontualidade. É necessário incluir um leve **carimbo de tempo**[7] no SPA para prevenir o recebimento e processamento de pacotes desatualizados.

A Especificação 1.0 [3] não abordou um atributo que forneça pontualidade. O campo *timestamp,* conhecido como *Current Unix Timestamp* - RFC 3339 [17] foi incluído ao modelo para ser o fator de referência de tempo humano. O *Controller* recupera o valor do *timestamp* recebido via SPA para confrontar com a sua hora atual. Os objetivos do módulo pontualidade são:

- Detectar precisão de avanço ou atraso de tempo do relógio do cliente em relação ao *Controller* com precisão de segundos;
- Descartar o SPA e não processar os módulos subsequentes caso o *timestamp* não esteja no intervalo de tempo permitido. A RFC 6238 [14] orienta a divisão do tempo em 30 segundos. Nesse estudo, o *Controller* aceita até uma medida de tempo anterior e uma posterior do *timestamp*/30;
- Possibilitar ao usuário através de um canal *out-of-band* (*e*m redes de computadores, um fluxo de dados separado do fluxo principal) visualizar possível erro de sincronia do relógio do seu dispositivo em relação ao *Controller.*

Caso o *timestamp* do cliente esteja dentro do intervalo permitido, inicia-se o processamento do módulo originalidade.

3) O Módulo Originalidade: no protocolo TOTP, a combinação (*password* + *timestamp* = *hash* resultante) pode tornar-se alvo de ataques. A RFC 6238 [14] orienta a divisão do *timestamp/30* . Com isso, o atacante pode originar ataques de força bruta ou dicionário sincronizados no tempo.

[7]É uma seqüência de caracteres, indicando a data e ou o tempo em que um determinado evento ocorreu. Disponível: https://cryptoid.com.br/banco-de-noticias/voce-sabe-o-que-e-dpct-e-carimbo-do-tempo/

Isso se deve ao fato de que o *timestamp* é de conhecimento público e a rotação do relógio do *Controlle*r não impede novos ataques.

Já uma senha criada com (*counter+password)* pode ter validade indeterminada. Objetivando aumento da segurança contra ataques de repetição, força bruta ou dicionário, esse trabalho uniu os protocolos **HOTP** e **TOPT** para criar a função **HTOTP**. Com isso, têm-se senhas de uso único baseado em eventos (*counter*) e tempo humano (*timestamp*). Incorporar um *counter* para cada usuário, a sua senha simétrica e o valor do *timestamp*, torna ataques de força bruta ou *replay* praticamente ineficazes. Considerações importantes sobre a implementação do módulo originalidade:

- A RFC 4226 [4] recomenda um valor da janela de sincronização que pode ser aceito quando há diferença entre os contadores de eventos. Nessa implementação, o *Controller* aceita valores do *counter* com uma diferença de até 30 além do seu valor atual. Os contadores iniciam em 500, tanto no cliente como no *Controller;*
- O *Controller* ao receber os valores *counte*r e *timestamp* do cliente, verifica se ambos estão dentro dos seus intervalos permitidos. Caso não estejam o SPA é descartado, caso contrário o HTOPT resultante é gerado e comparado com o recebido.

A Figura 3 apresenta os detalhes da função HTOTP. No lado cliente, a função HTOTP usa a chave derivada *Key1*, o valor do *counter* e *timestamp*/30 para gerar o *hash* resultante. No *Controller*, a função usa o *timestamp* e *counter* recebido incluso no SPA e a chave derivada *Key1* simétrica do cliente para novamente gerar o *hash* resultante para comparação.

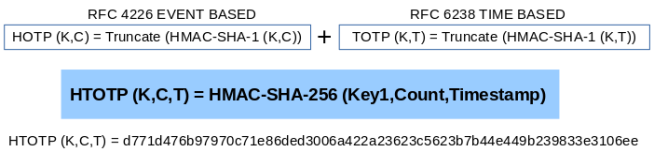

Fig. 3. Função HTOTP criada para fortalecer a originalidade e exemplo de *hash* resultante de 32 *bytes*. O truncamento, apesar de ser recomendado pelos protocolos HOTP e TOTP para facilitar a manipução pelo usuário, não foi usado visando o aumento dos níveis de segurança.

Deve-se ressaltar que o módulo originalidade considera o seguinte *nonce* para cada SPA e relativo a um único usuário:

counter || timestamp || Key1

4) Os Módulos Integridade e Autenticidade de Dados: a Especificação 1.0 [3] usa *o Secure Hash Algorithm* 1 (SHA-1) aplicado somente ao campo *counter*. Recentemente, pesquisadores descobriram técnicas que encontram colisões de forma muito mais eficiente que a força bruta [18]. Com o objetivo de melhorar a segurança é proposto o uso do *Secure Hash Algorithm* SHA-256. A Figura 4 apresenta a aplicação da integridade e autenticidade de dados.

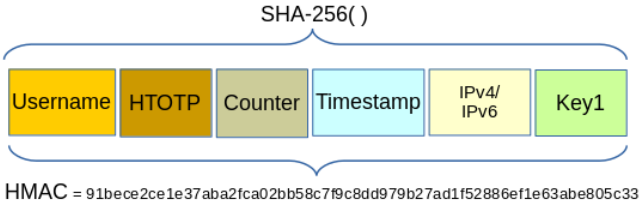

Fig. 4. Aplicação de integridade e autenticidade aos atributos do SPA.

O resultado da função SHA-256 retorna um HMAC de 32 *bytes* que será incorporado ao *payload* para ser criptografado com os outros campos.

5) O Módulo de Confidencialidade de Dados ou Privacidade: na atual abordagem da Especificação 1.0 [3], o SPA é enviado às claras, a qual os campos *AID, Password* e *Counter* podem ser facilmente descobertos. No entanto, é prático e possível cifrar os dados transmitidos de modo a impedir que um atacante entenda o conteúdo dos mesmos. O uso de algorítmos criptográficos baseados em cifras de chave simétricas foram introduzidos nesse trabalho, conforme a Figura 5.

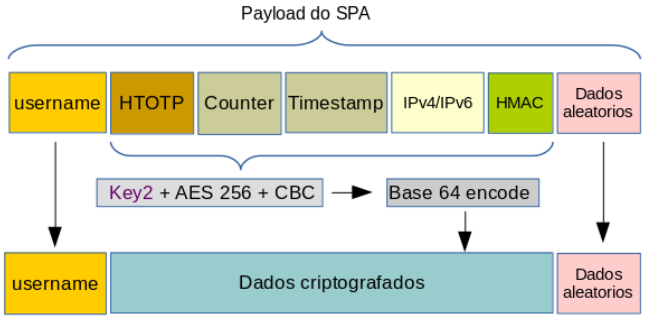

Fig. 5. Aplicação da criptografia ao *payload* do SPA.

O formato do SPA definido por este trabalho apresenta sete campos. O campo *username* é enviado às claras, usado para identificar o cliente quando o SPA for recebido pelo *Controller*. O campo "dados_aleatorios" é usado pelo módulo de confidencialidade de tráfego. Os campos criptografados foram: HTOTP (*hash* resultante OTP), *counter* (contador baseado em eventos), *timestam*p (tempo humano), IPv4/IPv6 (endereço IP que deseja ingressar no perímetro) e HMAC (*hash* resultante da integridade e autenticidade de dados).

Na abordagem com cifras de chave simétrica, tanto o cliente como o *Controller* compartilham a mesma chave para cifrar e decifrar. Ao longo dos anos, o Padrão Avançado de Cifração (AES - *Advanced Encription Standard*) tornou-se um padrão de cifra para chave simétrica. Chamado originalmente de *Rijndael* [19], admite tamanho de chaves de 128, 192 e 256 *bits* e encadeamento de bloco de cifra (CBC - *Cipher Block Chaining*) com tamanho de 128 *bits* [16, 20]. O AES permite implementações rápidas em *software* e *hardware* e não exige muita memória.

O resultado da criptografia (caracteres não imprimíveis), geralmente, apresentam falhas quanto transmitidos em protocolos TCP ou UDP. Nesse sentido, foram convertidos em

Base64[8] pela aplicação cliente para encaminhamento via datagrama UDP.

6) Módulo Compatibilidade IPv6: a compatibilidade com o protocolo IPv6 foi implementada. Para tanto, seguiu-se o princípio de que toda e qualquer funcionalidade executada em IPv4 deve ser estendida ao protocolo IPv6. Nesse cenário, o *Controller* implementa o conceito de pilha dupla ou *dual stack*[9].

O protocolo IPv6 apresenta como principal característica o aumento no espaço para endereçamento. No IPv4 o campo do cabeçalho reservado para o endereçamento possui 32 *bits*, já no IPv6 o espaço para endereçamento é de 128 *bits*. O endereço IP a ser autorizado a ingressar no perímetro é transportado dentro do *payload. C*aso o endereço for IPv6 pode haver um aumento de até 96 *bits* no tamanho do *payload.*

Diversas mudanças estruturais foram feitas no desenvolvimento das aplicações cliente e *Controller* para acomodar o novo espaço de endereçamento. As aplicações de *firewall* (do *Controller)* também tiveram que ser tratadas de forma distinta devido à diferença de *sintaxe* e operação.

7) Módulo Confidencialidade de Tráfego: avançando no conceito de confidencialidade, o ato de ocultar a quantidade de dados, origem ou destino é chamado de confidencialidade de tráfego. Saber a quantidade de dados que está indo para um determinado destino pode ser útil para um adversário em algumas situações. A Especificação 1.0 [3] prevê o envio de um único pacote implementado pelo protocolo TCP e de tamanho único. As seguintes funcionalidades que visam a confidencialidade de tráfego foram adicionadas ao modelo:

- Adicionou-se ao *payload* o campo "dados_aleatorios", formado por caracteres de *Base64* de tamanhos variados gerados aleatoriamente (e.g., entre 0 e 100 *bytes*). O SPA passa a ter tamanhos variados dificultando estabelecimento de um padrão. Esses dados são descartados pelo *Controller*;
- Geralmente em um mecanismo SPA, são usados protocolos UDP, TCP ou ICMP [7]. Este trabalho usa o protocolo UDP (*User Datagram Protocol*). É um protocolo não orientado a conexão da camada de transporte do modelo TCP/IP;
- O *Controller* deve ser identificado somente pelo seu endereço IP. O cliente não deve efetuar nenhuma resolução DNS para enviar o SPA;
- Não permitir que o *Controller* responda a qualquer requisição para dar a ideia de buraco negro (no contexto de redes de computadores, é um lugar onde os pacotes enviados para um determinado destino são todos descartados). Somente SPA enviados a uma porta específica serão recebidos e processados pelo *Controller* e nenhuma resposta dever ser retornada;
- Alternar o envio do SPA e TCP SYN da conexão subsequente através dos protocolos IPv4 e IPv6 desde que o cliente implemente o conceito de pilha dupla.

Após o recebimento e validação do SPA, o *Controller* permite que o dispositivo cliente conecte-se via TLS mútuo a porta 443 dando início ao processo de autenticação do usuário.

B. A Conexão TCP Subsequente

A conexão TCP subsequente a ser estabelecida após o envio do SPA é um problema que ainda persiste em ambas as técnicas de autenticação (SPA ou *port knocking*), conforme mencionado na seção 2.

Neste trabalho, não foram transportados dados no *header* dos protocolos IP ou UDP, pois não se deseja alterar suas funcionalidades padrões. Em relação ao *Controller*, não houve pré-alocação de recursos (e.g., portas de destino). Contudo, desenvolveu-se uma arquitetura escalável, redundante e com dupla abordagem (IPv4 e IPv6).

Para evitar que um invasor se conecte a um *Controller* em nome de um cliente válido após a validação do SPA, a arquitetura SDP foi dividida sistemas complementares: dois *Controllers front end,* um *Controller back end* e um modelo de dados centralizado com consistência sequencial[10]. A Figura 6 apresenta o projeto desenvolvido.

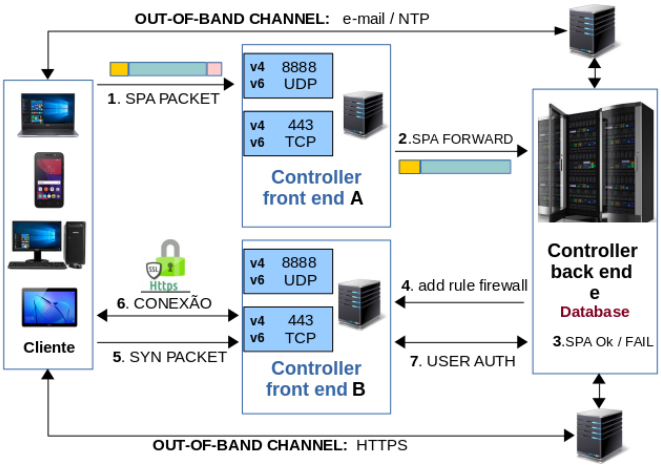

Fig. 6. Arquitetura proposta para envio do SPA e autenticação do cliente.

O detalhamento e funcionamento da arquitetura possui sete principais etapas descritas a seguir:

1) SPA PACKET: o cliente possui uma lista ordenada de IPs dos *Controllers front end* que pode conectar-se, recebidos por um canal *out-of-band* e atualizados com frequência. O envio do SPA tem como porta destino 8888 (UDP) e intercalado entre os *Controllers front end*;
2) SPA (FORWARD): o *Controller front end* implementa condições de recebimento de um SPA. Somente é aceito 1 datagrama UDP a cada 30 segundos de um mesmo IP de origem endereçado a porta 8888. Após o recebimento,

[8]Esquema de codificação comumente usados quando há necessidade de codificar dados binários que precisam ser armazenados e transferidos em mídias projetadas para lidar com dados textuais. Disponível: https://www.base64decode.org/

[9]Consiste na convivência do IPv4 e do IPv6 nos mesmos equipamentos, de forma nativa, simultâneamente. Disponível: http://ipv6.br/post/transicao/

[10]Todas as operações são realizadas de acordo com uma ordem, e nenhuma operação é iniciada antes do término da operação anterior.

verifica-se se o *payload* contém os 3 atributos previstos (*usename*, dados_criptorados e dados_aleatorios). Caso sim, o atributo "dados_aleatorios" é retirado do *payload* e o restante dos atributos são encaminhados ao *Controller back end* para validação;

3) SPA OK/FAILURE: O *Controller back end* recebe o SPA para processamento. Se as condições de todos os módulos de segurança forem satisfeitas o SPA é validado, caso contrário é descartado;
4) ADD RULE FIREWALL: O *Controller back end* verifica a lista de *front ends* ativos e solicita ao subsequente (de sua lista) para permitir o acesso a porta 443 TCP do IP informado pelo SPA;
5) SYN PACKET: Alguns segundos após o envio do SPA o dispositivo cliente já pode estabelecer conexão HTTPS com o IP seguinte de sua lista. Para isso, o *Controller front end* aguarda por até um minuto pelo SYN PACKET do cliente. Não é permitido a um *Controller front* end receber um SPA e estabelecer a conexão HTTPS subsequente para autenticação de um mesmo cliente;
6) CONEXÃO HTTPS: O cliente estabelece o TLS mútuo com *Controller front end* e recebe uma página para informar suas credenciais (*username* e *password*). O cliente tem até um minuto para informar suas credenciais;
7) USER AUTH: O *Controller back end* recebe as credenciais do cliente e dá continuidade ao processo de autenticação conforme a sua política de acesso.

Complementarmente, canais *out-of-band* (HTTPS, *email,* NTP) são disponibilizados ao usuário para prover serviços como: lista de *Controllers front end* atualizada, servidor NTP, *download* do aplicativo para envio do SPA, *logs* de recebimento de SPA, políticas de uso, resolução de problemas, recuperação e alteração de senhas. Todos os SPA recebidos pelo *Controller back end* validados ou não, tem suas informações guardadas em uma base de dados.

V. Experimentos e Resultados

Esta seção apresenta os resultados obtidos através da avaliação da abordagem proposta. Inicialmente é realizado uma análise comparativa de aspectos de segurança relacionados entre este trabalho e esquemas existentes. Posteriormente, experimentos foram realizados em dois conjuntos de testes: quantitativos e qualitativos.

Nos experimentos quantitativos foram medidos os tempos de processamento em milissegundos à execução de aplicações de teste:

1) Dos módulos acoplados. Concentraram-se na aplicação cliente e *Controller back end*;
2) Avaliação do desempenho da função HTOTP criada. Aplica-se somente a aplicação *Controller back end.*

O experimento qualitativo apresenta a variação do *delay* ou atraso tendo em vista a quantidade de *bytes* transportados entre o cliente e o *Controller* conforme o acoplamento dos módulos.

As aplicações de testes foram desenvolvidas na linguagem de programação ANSI C. A aplicação cliente possui 404 linhas de código e a aplicação *Controller* 1240 linhas de código, sendo 300 linhas do *Controller* f*ront end* e 940 linhas do *Controller back end.* O Controller b*ack end* usou banco de dados *Mysql* e o *Controller front end* utilizou o filtro de pacotes *Netfilter* (*Iptables). O Controller front end* implementa condições de recebimento do SPA relatadas na seção IV-B-2 e encaminha os dados para o *Controller back end* para validação.

De modo a auxiliar na obtenção dos valores quantitativos desenvolveu-se uma função em *ANSI C* que realiza pontos de marcação do tempo no início e fim do processamento de cada módulo. Na análise qualitativa utilizou-se a ferramenta "ping" contabilizando somente o tempo de ida. Em cada módulo foram realizadas 20 repetições para cada conjunto de testes.

Os experimentos foram realizados em um ambiente de virtualização Xen Source[11] com máquinas virtuais paravirtualizadas Debian 9 de 64 *bits*. O dispositivo cliente possui 512MB de memória RAM, dois processadores Intel(R) Xeon(R) 1.86GHz. Os *Controllers* possuem 1024MB de memória RAM e quatro processadores Intel(R) Xeon(R) 1.86GHz.

A. Análise da Segurança

Nesta seção é apresentado uma análise comparativa de proteção contra vulnerabilidades, ameaças ou ataques em relação aos esquemas [5, 6, 7, 8]. Tais esquemas fornecem proteção para: *c search, packet crafting, IP spoofing, connection hijacking e stolen-verifier attack.*

Este trabalho fortalece a proteção dos esquemas acima mencionados e ainda acrescenta aspectos de segurança para: *spoofed packets, man-in-the-middle attacks, port scanning, DNS spoof, DNS cache poisoning, DOS e DDOS attacks, man-in-the-browser, 0-day exploit attack, phishing attacks, evesdropping attack, key-stealing, pharming attacks, defacement, SYN flooding, rainbow attack,* extração ou alteração de dados do *payload*, exploração de vulnerabilidades dos *Controllers*, *scan* em redes e interrupção da disponibilidade.

B. Análise Quantitativa

1) Análise dos Módulos Acoplados: o parâmetro mais crítico para estudo analítico do envio e processamento do SPA, segundo [5, 7] é o fator de desempenho.

A sobrecarga de processamento foi avaliada baseando-se na quantidade de tempo em milissegundos, à medida que cada módulo é processado no cliente e no *Controller back end*. A Tabela I apresenta os resultados.

A tabela apresenta os resultados dos tempos de processamento total e individual obtidos durante o acoplamento dos módulos no dispositivo de teste cliente e no *Controller back end*. O tempo total é incrementado à proporção que cada módulo é processado. O tempo individual representa o custo de processamento de cada módulo.

[11] https://www.xenproject.org/

Tabela I. RESULTADOS DA SIMULAÇÃO QUANTITATIVA EM MILISSEGUNDOS.

Módulos Acoplados	**Cliente**		***Controller***	
	Total	Indiv.	Total	Indiv.
SPA básico[a]	0,26	0,26	0,49	0,49
Pontualidade	0,38	0,12	1,60	1,11
Originalidade	0,44	0,06	3,03	1,43
Integrid. e Autenticid.	0,54	0,10	3,81	0,78
Confidenc. de dados	2,96	2,42	6,59	2,78
Derivação Key1[b]	5,17	2,21	8,78	2,19
Derivação Key2[c]	9,25	4,08	13,36	4,58
Compat. IPv6	9,34	0,09	13,44	0,08
Confidenc. de tráfego	9,37	0,03	13,44	0,00

[a]Atributos: username e password
[b]Aplicada em: originalidade, integridade e autenticidade de dados.
[c]Aplicada em: confidencialidade de dados.

A Figura 7 sintetiza os resultados da tabela anterior mostrando a carga de processamento individual de cada módulo à medida que ocorre a criação do SPA pelo cliente e a validação do SPA pelo *Controller back end.* Os resultados da simulação são relativos a um único SPA.

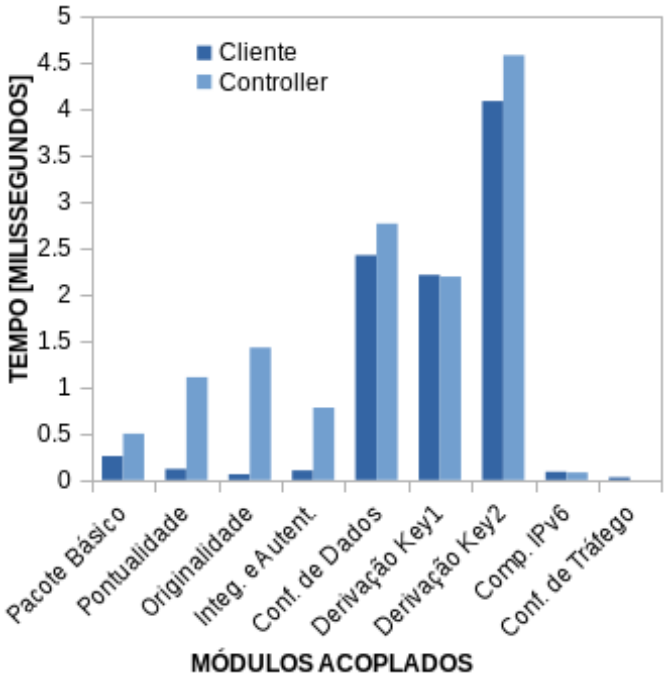

Fig. 7. Tempo do processamento individual de cada módulo.

Observa-se que a derivação de chaves *Key2* apresenta o maior tempo de processamento que um módulo alcançou pelo modelo proposto: 4.08 ms pelo cliente e 4.58 ms pelo *Controller back end*. O número de iterações a ser usado está ligado diretamente ao custo de processamento e memória. Um número de 1000 iterações para a *key1* e 2000 iterações para a *Key2* provavelmente não será significativo para as partes legítimas ao computar a chave, mas será enorme para os atacantes.

Para demonstrar o tempo de processamento total ao passo que os módulos são processados, apresenta-se a Figura 8 com o somatório em milissegundos. A proporção que os módulos foram incluídos, observa-se o aumento do tempo de processamento. O crescimento do processamento se manteve constante até a derivação *Key2*. Compatibilidade IPv6 e confidencialidade de tráfego apresentaram processamentos não significativos.

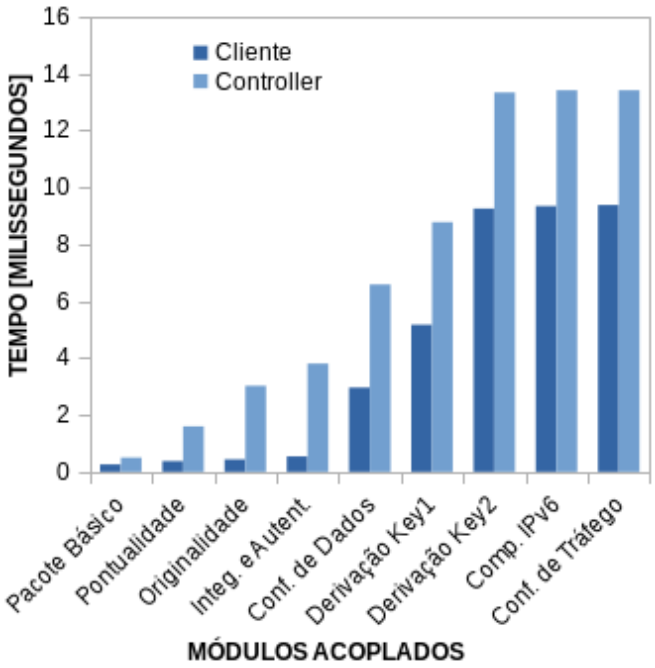

Fig. 8. Somatório dos tempos de processamento dos módulos.

Constata-se que o tempo de processamento dos módulos de segurança pelo *Controller back end* variam de 0.49 até 13.44 ms. Pelo cliente oscilam entre 0.26 e 9.37 ms. O tempo gasto em processamento é compensado pelo aumento significativo dos níveis de segurança do SPA conforme mencionado na seção V-A.

2) Análise da Função HTOTP: na definição de projeto, optou-se por enviar os atributos *counter* e *timestemp* junto com os demais campos do *payload,* todos criptografados. O objetivo dessa forma de implementação é diminuir o tempo de processamento do SPA. Com os valores de *counter* e *timestamp* já validados dentro dos intervalos permitidos, o *Controller back end* gera o *hash* resultante uma única vez **(1x)**.

Considerando que os atributos *counter* e *timestemp* não fossem transportados pelo SPA teríamos o seguinte cenário*:* 1. caso o relógio do cliente estivesse dessincronizado, o

Controller aceita uma medida de tempo posterior e uma anterior, podendo gerar até três vezes **(3x)** o *hash* resultante para validação ou descarte do pacote. 2. Já com o valor do *counter* dessincronizado, o *hash* resultante poderá ser gerado até trinta vezes **(30x)** antes de aceitar ou rejeitar o SPA. 3. Supondo que os valores do *counter* e do *timestamp* estivessem desatualizados, o *Controller* terá que testar todas as possíveis combinações até noventa vezes **(90x)** antes de aceitar ou descartar o SPA.

A Figura 9 apresenta a variação do tempo de processamento de um único SPA. Ele poderia variar de 1.43 até 128.43 ms.

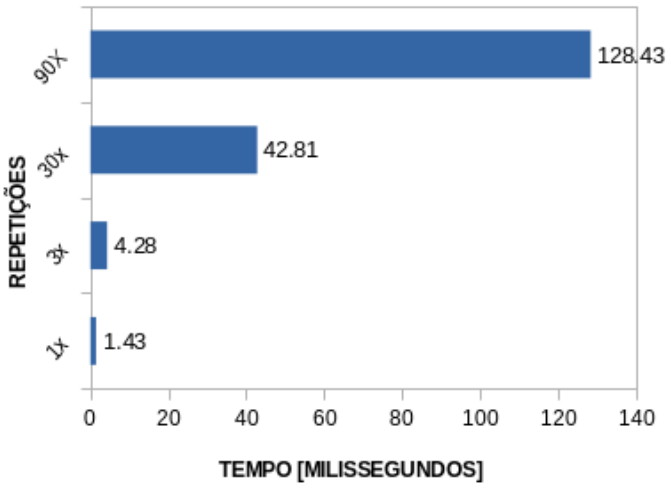

Fig. 9. Gráfico resultante da função HTOTP.

Com base nos resultados, observa-se que o transporte dos campos *timestamp* e *counter* dentro do *payload* SPA é extremante viável e com expressivos ganhos no tempo de processamento. Com isso, tem-se a certeza de que o SPA será processado uma única vez (**1x**).

C. Análise Qualitativa

Analisou-se o *delay* ou atraso em milissegundos tendo em vista a quantidade de dados em *bytes* transportados do cliente até o *Controller front end.* A Tabela II apresenta os resultados dos testes realizados em IPv4 e rede local *gigabit*.

A evolução em *bytes do payload* e o *delay* correspondente em milissegundos a cada módulo incorporado ao modelo. O crescimento do *payload* se mantém linear até o módulo de confidencialidade de dados e constante na derivação *Key1, Key2* e compatibilidade IPv6. Na confidencialidade de tráfego, o *payload* tem um acréscimo entre 0 e 100 *bytes*, podendo variar entre 275 a 375 *bytes*.

A análise qualitativa mostra que o acréscimo do *delay* tendo em vista o aumento dos *bytes* do *payload* não apresenta atrasos significativos. Diante disso, é completamente viável o aumento do tamanho do *payload* em prol do aumento significativo dos níveis de segurança conforme mencionado na seção V-A.

Tabela II. RESULTADOS DA SIMULAÇÃO QUALITATIVA.

Módulos Acoplados	***Payload* (*bytes*)**	***Delay* (ms)**
SPA Básico[a]	42	0,115
Pontualidade	88	0,132
Originalidade	97	0,135
Integridade e Autenticidade	185	0,141
Confidenc. de Dados	275	0,145
Derivação *Key1*[b]	257	0,145
Derivação *Key2*[c]	275	0,145
Compat. IPv6	275	0,145
Confidencialidade Tráfego	275∼375	0,146

[a] Atributos: *username* e *password.*
[b] Aplicada em: originalidade, integridade e autenticidade de dados.
[c] Aplicada em: confidencialidade de dados.

VI. Discussão

Os resultados experimentais demonstram o envio e recebimento de um *Single Packet Authorization* projetado sob o aspecto da pontualidade, originalidade, integridade e autenticidade de dados, derivação de chaves, compatibilidade com o protocolo IPv6 e confidencialidade de dados. A construção de soluções de segurança em módulos aumentam consideravelmente o grau de dificuldade de detecção, replicação ou leituras dos dados de um SPA. Cada vez que ocorre a iminência de um ataque há mecanismos que fazem com que os níveis de segurança sejam fortalecidos diminuindo sua probabilidade de sucesso.

Os resultados do experimento quantitativo mostram que a adição dos módulos propostos são viáveis. O aumento do tempo de processamento de um SPA se justifica pelos ganhos significativos de proteção. A análise da segurança (seção 4-A) apresenta aspectos de segurança fortalecidos e acrescidos. A análise da função HTOTP mostra que o transporte criptografado dos campos *timestamp* e *counter* pelo SPA garante que o *Controller back end* gere o *hash* resultante uma única vez para validação, podendo reduzir em até 90 vezes o tempo de processamento do módulo originalidade.

As conclusões da experimentação qualitativa revelam que o *delay* imposto pelo acréscimo de *bytes* adicionado ao *payload* pelos módulos geram impactos poucos significativos.

Cada tentativa de ataque de força bruta ou dicionário pode ter a mesma probabilidade de sucesso, independentemente da rotação do relógio do *Controller back end.* A RFC 6238 [14] orienta a divisão do tempo em 30 segundos e o *timestamp* é de conhecimento público. Assim, a combinação (*password* + *timestamp*) pode ser alvo de ataques. No protocolo HOTP, caso o *hash* resultante for

comprometido, o seu tempo de validade é indeterminado. A junção dos protocolos TOTP e HOTP resultaram na função HTOTP. Esta função une o tempo humano e um contator individual para fortalecer a segurança das senhas.

O uso de *Key Derivation Function* torna exponencialmente mais difícil a quebra de senhas. A derivação de chaves *Key1* (com 1000 iterações) teve seu uso aplicado no módulo originalidade, autenticidade e integridade de dados. No módulo de confidencialidade de dados, usou-se a derivação de chaves *Key2* (com 2000 iterações). Com isso, a chave mestra não foi exposta durante a transmissão do SPA. Através da derivação de senhas criptográficas aplicadas aos módulos, torna-se praticamente impossível um atacante descobrir a senha mestra correta em um ataque de força bruta ou dicionário.

Evitar que um invasor conecte-se a um *Controller front end* em nome de um cliente válido após a validação do SPA e antes do estabelecimento do TLS mútuo foi um grande desafio abordado por esse trabalho. Organizar a infraestrutura em dois *Controllers front end* e um *Controller back end,* além de canais *out-of-band,* fornece escalabilidade e redundância ao modelo proposto. Alternar o envio do SPA e da conexão TCP subsequente entre os *Controllers front end* propicia a ocultação desejada. Também é possível intercalar o envio do SPA entre os protocolos IPv4 e IPv6, desde que o cliente os implemente.

VII. Conclusões e Trabalhos Futuros

Esse trabalho analisou a Especificação 1.0 [3] e propôs outras definições ao padrão proposto pela *Cloud Security Alliance*, tendo como objetivo a definição de um novo modelo de criação e envio de um *Single Packet Authorization*, passo inicial para ter acesso a um Perímetro Definido por Software. Complementarmente se definiu uma nova arquitetura para estabelecimento do TLS mútuo para autenticação do cliente.

Por meio da avaliação experimental qualitativa e quantitativa, e análise da segurança, demonstrou-se o aumento significativo nos níveis de proteção e confiabilidade do modelo proposto não existindo, no entanto, uma solução única. Constatou-se que é necessário várias tecnologias e processos que ofereçam ocultação da infraestrutura, de aplicativos e controles de acesso, bem como prover conexão segura para proteger redes e servidores de ataques cibernéticos. As soluções propostas contribuem para o aumento dos níveis de proteção e de resiliência do padrão de referência SDP.

A validação do dispositivo Cliente, apesar de recomendada, está além do escopo da Especificação 1.0 [3]. Como trabalho futuro, pretende-se incluir o inventário de dispositivos Clientes para ingresso a um SDP.

Referências

[1] K. Narayanaswamy, "Dynamic access control policy with port restrictions for a network security appliance," Juniper Networks Inc, 10 2008, patent US8572717B2. [Online]. Available: https://patents.google.com/patent/US8572717B2/en

[2] R. Ward and B. Beyer, "BeyondCorp: A New Approach to Enterprise Security," *;login:*, vol. Vol. 39, No. 6, pp. 6 – 11, 2014.

[3] B. Bilger, A. Boehme, B. Flores, and Z. Guterman. (2014, Aplil) Cloud Security Alliance - SDP - Specification 1.0. [Online]. Available: https://cloudsecurityalliance.org/download/sdp-specification-v1-0/

[4] M. View, D. M'Raihi, F. Hoornaert, D. Naccache, M. Bellare, and O. Ranen, "HOTP: An HMAC-Based One-Time Password Algorithm," RFC 4226, dec 2005. [Online]. Available: https://rfc-editor.org/rfc/rfc4226.txt

[5] M. Tariq, M. S. Baig, and M. T. Saeed, "Associating the Authentication and Connection-Establishment Phases in Passive Authorization Techniques," *World Congress on Engineering*, vol. I, July 2008.

[6] J. Liew, S. Lee, I. Ong, H. Lee, and H. Lim, "One-Time Knocking framework using SPA and IPsec," in *2010 2nd International Conference on Education Technology and Computer*, vol. 5, 2010, pp. V5 – 213.

[7] H. Zorkta and B. Almutlaq, "Harden Single Packet Authentication (HSPA)," *International Journal of Computer Theory and Engineering*, vol. 4, no. 5, pp. 717 – 721, 2012.

[8] R. Kumar and I. Talwar, "Network Security using Firewall and Cryptographic Authentication," *International Journal of Computer Applications (0975 – 8887)*, vol. 57, no. 23, November 2012.

[9] M. Rash. (2016, Janeiro) Single Packet Authorization with Fwknop, A Comprehensive Guide to Strong Service Concealment with fwknop. [Online]. Available: http://www.cipherdyne.org/fwknop/docs/fwknop-tutorial.html

[10] Chai Wen Chuah, Edward Dawson, and Leonie Simpson, "Key Derivation Function: The SCKDF Scheme," in *Security and Privacy Protection in Information Processing Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 125 – 138.

[11] B. Kaliski, "PKCS #5: Password-Based Cryptography Specification Version 2.0," RFC 2898, sep 2000. [Online]. Available: https://rfc-editor.org/rfc/rfc2898.txt

[12] T. Hansen and D. E. E. 3rd, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)," RFC 6234, may 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6234.txt

[13] J. X. Zhao, "Research and Design on an Improved TOTP Authentication," in *Information Technology Applications in Industry II*, ser. Applied Mechanics and Materials, vol. 411. Trans Tech Publications, 12 2013, pp. 595 – 599.

[14] M. View, J. Rydell, M. Pei, and S. Machani, "TOTP: Time-Based One-Time Password Algorithm," RFC 6238, may 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6238.txt

[15] A. Beikverdi and I. K. T. Tan, "Improved look-ahead re-synchronization window for HMAC-based one-time password," in *IET International Conference on Wireless Communications and Applications (ICWCA 2012)*, 2012, pp. 1 – 5.

[16] L. L. Peterson and B. S. Davie, *Redes de Computadores - uma abordagem de sistemas*, 5th ed. Elsevier, 2013.

[17] C. Newman and G. Klyne, "Date and Time on the Internet: Timestamps," RFC 3339, jul 2002. [Online]. Available: https://rfc-editor.org/rfc/rfc3339.txt

[18] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1," *CWI Amsterdam and Google Research*, pp. 1 – 23, 2017. [Online]. Available: http://shattered.io/static/shattered.pdf

[19] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," 1999.

[20] A. Desai, K. Ankalgi, H. Yamanur, and S. S. Navalgund, "Parallelization of AES algorithm for disk encryption using CBC and ICBC modes," in *2013 Fourth (ICCCNT)*, 2013, pp. 1 – 7.