# Reproduzindo ataques contra o controle de congestionamento do TCP (RFC 2581)

**Rodrigo Lampier**

[1]Departemento de Informática – Universidade Federal do Paraná (UFPR)
R. Evaristo F. Ferreira da Costa, 383-391 - Jardim das Americas, Curitiba - PRpreocupação

lampier@ufpr.br

***Abstract.*** *In this paper, a series of attacks against TCP congestion control (RFC 2581) is performed based on paper "TCP Congestion Control with a Misbehaving Receiver". Given the preoccupation with vulnerabilities that was explained by the authors, a Python TCP Reno implamentation was performed using the Scapy library, and Mininet simulator were used to prove that those attacks are susceptible to the scenarios pointed out in the article. In the end, we tried to meet all the requirements mentioned in the paper to get a closer result from the authors and we achieved significant numbers.*

***Resumo.*** *Neste artigo é realizada uma reprodução de uma série de ataques contra o controle de congestionamento do TCP (RFC 2581) baseado no artigo "TCP Congestion Control with a Misbehaving Receiver". Dada a preocupação que tais vulnerabilidades explanadas no artigo possam gerar ataques de maior impacto como um DDoS, foi realizada a implementação do TCP Reno em Python utilizando a biblioteca Scapy e o simulador Mininet para realizar a prova que os ataques são suscetíveis no cenário apontado no artigo. Por fim, buscamos atender todos os requisitos citados no artigo para que a reprodução dos ataques chegasse o mais próximo possível dos resultados dos autores e conseguimos números significativamente próximos.*

## 1. Introdução

O artigo escolhido para reprodução foi "TCP Congestion Control with a Misbehaving Receiver"por Savage et. al., no qual os autores descrevem três vulnerabilidades que podem ser exploradas no protocolo TCP (RFC 2581), e propõe soluções para elas.

O TCP, particularmente em suas implementações anteriores, depende da cooperação entre o sender e o reciever para funcionar corretamente. Sem esta cooperação, podem surgir diversos problemas como, por exemplo, uma falha no controle de congestionamento e o compartilhamento de recursos de forma desigual (unfairness). Conforme supracitado, o artigo ilustra três caminhos distintos para se obter um resultado: um recievers "ganancioso", fazendo com que um remetente aumente de forma rápida e volumosa sua taxa de transmissão.

A primeira vulnerabilidade, nomeada Ack Division, envolve dividir um ACK de um único pacote em vários ACKs para aumentar a janela de congestionamento, pois assim cada ACK é interpretado como pertencente a um pacote separado. Neste caso, a vulnerabilidade decorre da discrepância entre o controle de erros de granularidade de bytes e

o controle de congestionamento de granularidade de segmentos. A segunda vulnerabilidade, chamada de DupAck Spoofing, envolve o envio de duplicados segmentos ACK em resposta a um único pacote. Isso faz com que o algoritmo de controle de congestionamento TCP entre no modo Fast Retransmit, assim a cada ACK a janela aumenta, assim um invasor pode aumentar a janela para um tamanho arbitrariamente grande enviando um grande fluxo de ACKs duplicados. O terceiro ataque, nomeado Optimistic Acking, envolve o envio de ACKs ”otimistas”antes dos dados realmente enviados. Como os ACKs chegam cedo, os dados são enviados mais rapidamente do que o algoritmo de controle de congestionamento normalmente teria permitido.

Um ponto de atenção é que essas vulnerabilidades podem levar um atacante a produzir um ataque de negação de serviço distribuído (DDoS), por exemplo, um grupo de recievers maliciosos podem acionar um número considerável de senders, congestionando a rede e, em seguida, esgotando com os recursos dela.

Sendo assim, o intuito desse trabalho é realizar a reprodução das figuras 4, 5 e 6 do artigo original (Figura 1), onde os autores realizam experimentos para provar as vulnerabilidades supracitadas.

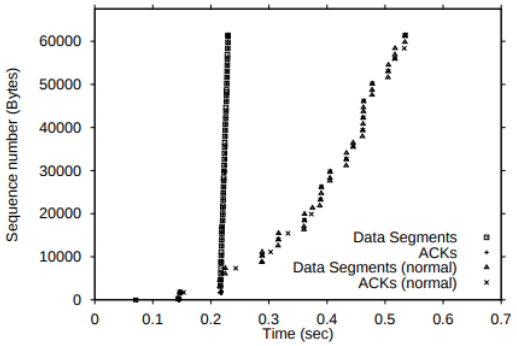


Figure 4: The TCP Daytona *ACK division* attack convinces the TCP sender to send all but the first few segments of a document in a single burst.

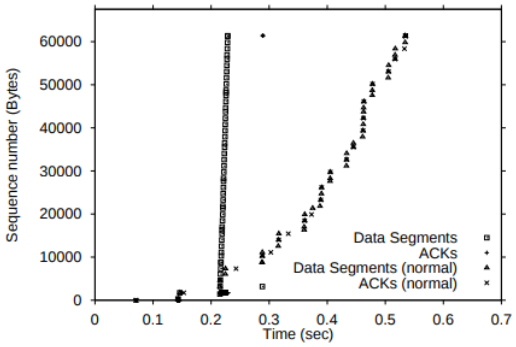


Figure 5: The TCP Daytona *DupACK spoofing* attack, like the ACK division attack, convinces the TCP sender to send all but the first few segments of a document in a single burst.

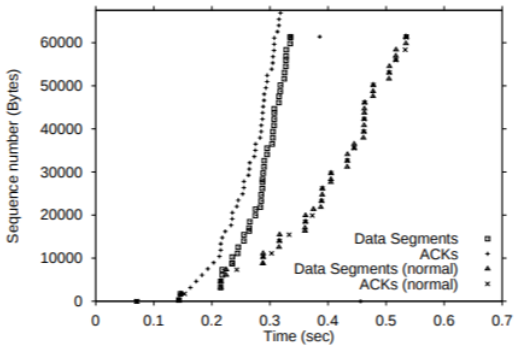


Figure 6: The TCP Daytona *optimistic ACK* attack, by sending a stream of early ACKs, convinces the TCP sender to send data much earlier than it normally would.

**Figura 1. Resultados do artigo original**

A organização do trabalho dá-se, primeiramente, pela seção 2 explicando os detalhes da reprodução, tal como topologia e ferramentas utilizadas, e em seguida, são mostrados os resultados do experimento e a comparação com o experimento do artigo de origem.

Na seção 3 são mostrados brevemente as dificuldades encontradas durante a reprodução do trabalho. Na seção 4, é realizada a conclusão do presente trabalho, e por fim, na seção 5 referências.

## 2. Reprodução e Resultados

Enquanto os autores utilizaram a linguagem C no artigo, foi utilizada na reprodução do experimento a implementação de um cliente TCP Reno utilizando o Scapy em Python. Scapy é uma ferramenta que utiliza a linguagem Python e fornece APIs de alto nível para codificar, decodificar, enviar, receber, capturar pacotes e etc. Sendo assim, nosso cliente TCP Reno foi modificado para simular todos os três ataques descritos no artigo. Estes ataques foram executados no Ubuntu 16.04 utilizando o emulador Mininet, ao contrário dos autores que utilizaram o NS2. No Mininet foi construída uma simples topologia onde temos um switch e dois hosts. Estes hosts estão conectados através do switch, sendo um o sender e o outro o recievers( ou receiver, fazendo alusão ao artigo original), como mostra a Figura 2.

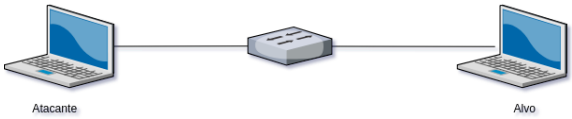


**Figura 2. Topologia utilizada na reprodução**

Ao executar o experimento, foi registrado o número de seqüência dos pacotes que o reciever recebe e o número de ack daqueles que ele envia, e a partir desses números os gráficos foram gerados.

De acordo com os gráficos mostrados nas figuras, Figura 3, Figura 4, e Figura 5, podemos observar que há uma discrepância em relação ao tempo gasto nos ataques e respostas comparado aos gráficos do artigo original. Isso acontece pois a linguagem Python e a biblioteca Scapy são muito mais lentas que a linguagem C, sendo assim foi utilizado RTT maior para acentuar a eficácia dos ataques e chegar mais próximo do gráfico original.

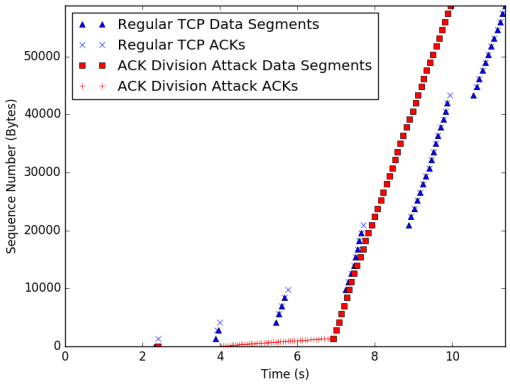


**Figura 3. Reprodução do ataque TCP Daytona ACK division**

Assim como no artigo original, os gráficos da Figura 3 e Figura 4 demonstram o reciever convencendo o sender a enviar diversos segmentos em uma só rajada, exceto os primeiros segmentos.

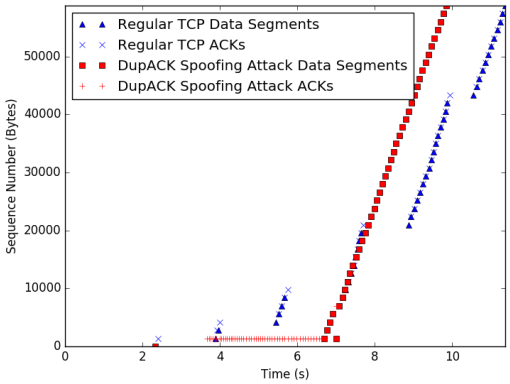


**Figura 4. Reprodução do ataque TCP Daytona DupACK spoofing**

Já na Figura 5, assim como o gráfico do artigo original, é demonstrado o receiver convencendo o sender a enviar os segmentos de dados e os acks muito mais cedo do que deveria ter sido enviado.

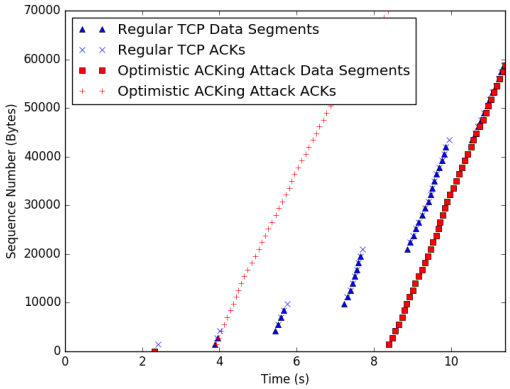


**Figura 5. Reprodução do ataque TCP Daytona optimistic ACK**

Acredito que, apesar do lapso temporal ser diferente do artigo original, o experimento realizado atendeu a expectativa, mostrando que os ataques funcionaram utilizando TCP Reno e provando novamente a existência das vulnerabilidades no TCP baseado na RFC 2581.

## 3. Dificuldades encontradas

Abaixo serão descritas algumas dificuldades encontradas durante o trabalho.

Como fazia muito tempo que havia realizado a disciplina de redes, tive que voltar ao estudo para entender alguns conceitos de como funcionavam os algoritmos de controle de congestionamento para o TCP Reno.

Embora o Scapy seja uma biblioteca que permite a manipulação de pacotes a partir de um programa de espaço do usuário, ele ainda usa sockets de kernel para suas APIs de rede. Assim, inicialmente, quando implementamos nossa linha de base TCP Reno, os pacotes foram interceptados pelo kernel e uma série de pacotes de reset foram injetados. Para contornar esse problema, ignoramos todos os pacotes de redefinição de nossa implementação para evitar a interferência do kernel em nosso fluxo de TCP.

Relacionado ao LaTex, eu nunca tinha utilizado. No começo não sabia qual editor utilizar, e tive problemas com o package de lingua para português, além de alguns erros de sintaxe. O modelo da SBC ajudou muito e é este que está sendo utilizado. Após a escolha de um editor, e instalação devida de pacotes, a escrita do relatório foi iniciada. Só tive o problema de não conseguir colocar os índices das seções de forma automática.

## 4. Conclusão

O TCP foi originalmente projetado para um ambiente cooperativo e contém algumas vulnerabilidades que um receiver mal intencionado pode explorar para obter um serviço aprimorado às custas de outros clientes da rede, ou para implementar um ataque de negação de serviço. Neste artigo foi realizada a reprodução fiel ao artigo original descrevendo os mecanismos ACK division, DupACK spoofing e Optimistic ACK, e suas implementações para demonstrar que os ataques são reais e amplamente aplicáveis.

Vale ressaltar que o artigo em questão foi escrito em 1999, época na qual a Internet estava começando a creser. As vulnerabilidades encontradas na época foram relacionadas a RFC 2581 (1999), e a correção delas foram realizadas nas seguintes RFCs: DupACK - RFC 5681 (2009), Opt Ack RFC 3540 (2003), e Ack Division RFC 3465 (2003).

Baseado nesse lapso temporal, podemos concluir que ficamos muito tempo expostos a tais vulnerabilidades, mas como mencionado anteriormente, a Internet ainda não era amplamente difundida e ainda estava sendo aperfeiçoada. Hoje em dia, vulnerabilidades são corrigidas de maneira mais rápida uma vez que temos diveresos estudiosos trabalhando nisso, além do uso da Internet em larga escala.

Como uma continuação do trabalho, seria interessante tentar realizar o mesmo experimento, porém utilizando a versão mais recente do TCP e avaliar se os ataques ainda podem acontecer apesar das correções realizadas.

## 5. Referências

Referências.

[1] Savage, Stefan, et al. “TCP congestion control with a misbehaving receiver.” ACM SIGCOMM Computer Communication Review 29.5 (1999): 71-78.

[2] RFC 3465 - TCP Congestion Control with Appropriate Byte Counting (ABC) https://tools.ietf.org/html/rfc3465, Dez 2018

[3] RFC 3540 - Robust Explicit Congestion Notification (ECN) Signaling with Nonces https://tools.ietf.org/html/rfc3540, Dez 2018

[4] RFC 5681 - TCP Congestion Control https://tools.ietf.org/html/rfc5681, Dez 2018

[5] RFC 2581 - TCP Congestion Control https://tools.ietf.org/html/rfc2581, Dez 2018