2018

Towards a Secure Zero-rating Framework with Three Parties

Zhiheng Liu
Lehigh University, liu.cmri@gmail.com

Towards a Secure Zero-rating Framework with Three Parties

by

Zhiheng Liu

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computer Science and Engineering

Lehigh University

November 2018

Certification of Approval

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science in Computer Science and Engineering.

_____

Date

_____

Prof. Mooi Choo Chuah

Thesis Advisor

_____

Prof. Daniel Lopresti

Chair of Department of

Computer Science and Engineering

Acknowledgements

I want to thank my advisors, Professor Yinzhi Cao and Professor Mooi Choo Chuah, for the guidance throughout my research. I am grateful to not only had their mentorship in my work but also in lifetime endeavors. I am thankful for my years in Lehigh University met and worked with many smart and wonderful people who enlightened me all the time on both research and daily life.

I would also like to thank department faculties, Professor Daniel Lopresti, Professor Hector Munoz-Avila, Professor Brian D. Davison, Professor Ting Wang, and Professor Sihong Xie, for teaching me the fantastic knowledges and technologies, as well as department staffs, especially Jeanne Steinberg, and Heidi Wegrzyn, for the strong support throughout the two years.

Thanks to my research colleagues, Zhen Zhang, Shihao Jing, Zhaohan Xi, Song Li, Shujiang Wu, and Dan Luo, for supporting me as I was stuck in the middle of my work all the times. It is my pleasure and fortunate to work beside you. I hope you all can have bright futures.

Most importantly, I am fortunate to have the support from my family. Thanks for my Mom and Dad for the consistent support in pursuit of my degree and all the decisions I made.

Finally, thank you to my wife, Lingzi. It is a huge sacrifice for you to quit the job you loved and came to the US with me for pursuing my dream. You inspire me always to be the best I can be. I cannot go this far without you! Thank you!

Contents

Abstract **1**

1 Introduction **2**

1.1 Problem Statement 2

1.1.1 Understand Existing ISP Zero-rating Framework 2

1.1.2 Vulnerability of Zero-rating Service 3

1.2 Exsiting Solution 3

1.3 Zero-rating FRamework with thrEe partiEs (ZFree) 4

1.4 Challenges in ZFree 4

1.5 Consideration of ZFree Control Plane Protocol 5

2 The Free-riding Attacks **7**

2.1 Threat Model 7

2.1.1 The Normal Zero-rating Charging Flow 7

2.1.2 The Free-riding Attack Flow 8

2.1.3 Intractability in Zero-rating Threat Model 9

2.2 Attack Methodologies and TestBeds Setup 10

2.3 Case Studies on Free-riding Attacks against real-world ISPs 12

2.3.1 Real-world Cellular Networks Attacks 12

2.3.2 Real-world WiFi Networks Attacks 15

2.4 Attack on Defensive Research Solutions 16

2.4.1 Introduction of Defensive Research Solutions 17

2.4.2 Attack on Defensive Research Solutions 17

2.5 Understanding the Severeness of Free-riding Attacks 20

3 ZFree Overview **21**

3.1 System Architecture 21

3.2 Deployment Model 22

3.2.1 Cellular Network 23

3.2.2 WiFi Network 23

3.3 Use Cases 24

3.3.1 Parental Control 24

3.3.2 Policy and Charging Control (PCC) 24

4 ZFree System Design **25**

4.1 ZFree components 25

4.1.1 Server Agent 25

4.1.2 ISP Assistant 26

4.2 ZFree Control Plane Protocol 26

4.2.1 Setup Phase 26

4.2.2 Control Phase 27

4.3 ZFree Algorithm Design 28

4.3.1 Server Agent Algorithm 28

4.3.2 ISP Assistant Algorithm 29

5 Formal Security Analysis **34**

5.1 Verification Goals 34

5.2 Formal Models 35

5.2.1 Network Cookies 35

5.2.2 IP Whitelist 35

5.2.3 ZFree 36

5.3 Verification Results 36

6 Experiment and Evaluation **38**

6.1 Environment Setup 38

6.2 Evaluation Results 40

6.2.1 End-to-end Performance 40

6.2.2 ZFree in ISP Core Network 43
6.2.3 Security 46

7 Discussions **47**

8 Related Work **49**

8.1 Existing Attacks 49
8.2 Existing Zero-rating Framework 50
8.3 Other Techniques 50

9 Conclusion **52**

Bibliography **53**

Appendix: Raw Trace Example Outputted by ProVerif **60**

Vita **1**

List of Figures

2.1 Normal Zero-rating Charging Flow 8
2.2 Free-riding Attack Flow 9
2.3 Masquerade CP Server Attack 10
2.4 Response Modification Attack 11
2.5 Network Cookie Architecture 18

3.1 ZFree's Architecture over Cellular Network 22
3.2 ZFree's Architecture over WiFi Network 23

4.1 ZFree Control Plane Protocol 31
4.2 Server Agent Algorithm 32
4.3 ISP Assistant Algorithm 33

6.1 Cellular Network Testbed 39
6.2 WiFi Network Testbed 40
6.3 ZFree Evaluation Graphs: (a) The CDF of Loading Time of Top 500 Alexa Websites; (b) The End-to-end Delay vs. the Network Bandwidth; (c) The End-to-end Delay vs. the Number of Connections in Mobile Network Environment with ZFree's Non-blocking Mode; and (d) The End-to-end Delay vs. the Number of Connections in WiFi Environment with ZFree's Blocking Mode. 41
6.4 LTE Handover Throughput in both Legacy Mobile Network and Mobile Network with ZFree Enabled 43
6.5 Network Traffic (Gbps) and CPU Usage (%) for the ISP Assistant (Top) and two CP Agents (Middle and Bottom) under Stress Test 45

9.1 Counter example traces on verifying response integrity for Network Cookies (TCP Connection) 61
9.2 Counter example traces on verifying CP authenticity for IP Whitelist based zero-rating framework (TLS Connection) 61
9.3 example traces on verifying ZFree 62
9.4 example traces on modifying ZFree packet key hash 62
9.5 example traces on modifying ZFree using weak hash function 63

Abstract

Zero-rating services allow mobile users to access contents from contracted CP free of data charge. In this thesis, we introduce attacks against the zero rating service which allows extra non-contracted traffic to be transported free of charge. We call this types of attack the 'free-riding' attack. Specifically, we create two types of free-riding attacks: 1) masquerade zero-rating CP attack; 2) response packets modification attack. We conducted multiple experiments on several major commercial cellular and WiFi ISPs in the United States and China. The experimental results show that all these ISPs are vulnerable to free-riding attacks.

In this thesis, we also propose a secure and backward compatible zero-rating framework, called ZFree. ZFree authorizes network traffic from valid CP to be zero-rated. Next, we perform a formal security verification using ProVerif on ZFree. The formal verification results show that ZFree is secure in preserving packet integrity and CP server authenticity.

We have implemented an open-source prototype of ZFree available at the Github repository (https://github.com/zfree2018/ZFREE). Our evaluation shows that ZFree is lightweight, scalable and secure.

Chapter 1

Introduction

Internet service providers (ISPs) offer zero-rating services to its users, thus allowing them to visit contents from contracted content providers (CPs) without being charged for the data usage. Here are two examples: 1) cellular ISP A provides its users a zero-rating program over LTE network. Users in this program can stream contents from over one hundred CPs, such as YouTube, without being charged of cellular data usage; 2) WiFi ISP D provides a WiFi zero-rating service. Users in the airplane can use ISP D's cabin WiFi to access certain websites, such as the airline company's official website, for free.

1.1 Problem Statement

1.1.1 Understand Existing ISP Zero-rating Framework

We research an existing ISP zero-rating framework. Real-world ISP first inspects network packets with inspection mechanism, such as Deep Packet Inspection (DPI). DPI differentiates zero-rating traffics based on packets' header. We discover that ISPs are using 'hostname' header in HTTP and 'Server Name Indication (SNI)' header in HTTPs as inspection identifier. Next, ISPs define zero-rating policies for these zero-rating traffic. The zero-rating framework is implemented across two types of mobile networks: cellular network and WiFi network. Both networks involve three parties: the ISP, the CP, and the user.

1.1.2 Vulnerability of Zero-rating Service

Although zero-rating service provides convenience for both users and CPs, it also contains vulnerabilities. Based on the existing ISP zero-rating framework, the user can modify non-zero-rated packets to zero-rating packets or inject zero-rating identifier into non-zero-rated packets, thus tricking ISP to zero-rate the traffic. This attack is called the free-riding attack.

The mobile users are malicious attackers in free-riding attack. The free-riding attack can bypass ISP's zero-rating policies, thus offering the attacker free data on charged traffic. This free-riding attack is new and different from the traditional ISP-side charging attacks. Traditional attacks, such as network domain service (DNS) attack [42] and TCP retransmission attack [20, 21], exploit protocol bugs. Free-riding attack exploits the vulnerabilities of ISP charging system, thus achieving the attack.

The vulnerabilities of zero-rating service bring severe losts to ISPs and CPs. One recent report from Sandvine [1] concludes that a major US network carrier loses $7,000,000 in a month due to such free-riding attacks. We also manually analyze the billing data from one cellular ISP in China. The result indicates that this ISP loses at least half a million US dollars per month for 71TB free-riding traffic due to such attacks.

1.2 Exsiting Solution

There are two existing solutions: 1) Yiakoumis et al. proposed Network Cookies [51] in which an authentication token (called Network Cookie) serves as a zero-rating ticket to the client. The client injects this cookie to the zero-rating traffic. ISP inspects the cookie and frees the date charge. 2)Facebook provides a framework based on IP Whitelist, called Facebook Zero [11]. IP Whitelist framework allows ISPs to obtain an IP list from CP for authentication. In other words, traffic from the servers that belong to this IP Whitelist can be zero-rated.

We discover that these two existing solutions are both vulnerable to free-riding attacks. For Network Cookie solution, we show that an attacker can either bind a zero-rating cookie to non-zero-rating traffic or inject non-zero-rating data into zero-rating traffic to bypass ISP zero-rating policies. For IP Whitelist solution, we show that the

malicious users can easily camouflage TCP/IP packets, thus bypassing the IP Whitelist.

1.3 Zero-rating FRamework with thrEe partiEs (ZFree)

We propose a brand-new Zero-rating FRamework with thrEe partiEs (ZFree) to defend against the powerful free-riding adversary. ZFree allows ISP and CP to exchange authentication information in zero-rating services. There are two points which explain why existing solutions, such as IP Whitelist solution, cannot defend against free-riding attacks. First, information should be kept from the user, the potential free-riding attacker. Existing work, such as Network Cookie [51], fails to defend against free-riding attacks, because it makes authentication information available to the user. Second, information should be able to authenticate the communication between the user and CP.

1.4 Challenges in ZFree

While the insight of ZFree is intuitively simple, there are still challenges in ZFree's design. We list the challenges as follows:

- Security. ZFree needs to validate CP's authenticity and the communication integrity between CP and the user.
- Backward Compatibility. ZFree needs to incur minimum deployment burden to both CPs and ISPs. The ZFree backward compatibility includes no changes to existing (*i*) codebase and (*ii*) network packets. Because any such changes may break existing network functionalities, such as intrusion detection systems and loader balancer.
- Privacy. ZFree needs to preserve the communication privacy between user and the CP. That is, CP cannot directly reveal any communication contents to ISP for authentication.
- Performance. The performance overhead added to the end-to-end communication needs to be minimum. If an unencrypted communication is sufficient between the CP and the user, we do not need to encrypt the control plane communication for authentication, which brings overhead.

1.5 Consideration of ZFree Control Plane Protocol

ZFree operates a secure protocol, called ZFree control plane protocol. This protocol allows ZFree to communicate between CP and ISP for authenticating zero-rating traffic.

ZFree control plane protocol has a trivial overhead. The protocol only needs to preserve server authenticity and data integrity for both control and data planes but not necessarily data secrecy. In particular, we make the following contributions in design ZFree control plane protocol to meet all the aforementioned properties.

- Conducting a formal security analysis. We formally model ZFree control plane protocol using ProVerif, a formal protocol cryptographic analysis tool. ProVerif concludes that ZFree is secure and robust to free-riding attacks.
- Deploying pluggable components at both ISP and CP. To ease the deployment burden and maintain backward compatibility, we deploy a ZFree *Server Agent* at CP gateway and a ZFree *ISP Assistant* at ISP's core network. Server Agent can: 1) sniff packets from the data plane; 2) covert packets into hash code; 3) send the hash code to ISP Assistant for authorization. ISP Assistant can: 1) sniff packet from data plane in core network; 2) covert packets into hashcode; 3) verify packet authorization with CP Server Agent.
- Verifying packet integrity without violating the end-to-end privacy. ISP Assistant verifies packet integrity by checking the secure hashes sent from Server Agent. When ISP Assistant finds a match of the corresponding packet, it authorizes the packet to be zero-rated. ZFree does not need to understand the application layer protocols, thus preserving end-to-end privacy.
- Matching hash values in a distributed manner. ISP Assistant matches hashes received from Server Agent. This matching process can be costly. We achieve this process by sharding the matching task to distributed nodes based on the hash value's prefix. Our evaluation shows: 1) ZFree in non-blocking mode, a mode used in cellular network, incurs only 1.26% overhead; 2)ZFree in blocking mode, a mode used in WiFi network, incurs 8.79% overhead; 3) both ZFree non-blocking mode and ZFree blocking mode introduce less network latency than TLS encryption.

Currently, ZFree is described in a draft in Internet Engineering Task Force (IETF). We have obtained support from vendors, real-world ISPs, and content providers, such as Cisco, China Mobile, China Telecom, and Alibaba. ZFree prototype implementation is open-sourced and available at the following repository (https://github.com/zfree2018/ZFREE).

Chapter 2

The Free-riding Attacks

In this chapter, we introduce the free-riding attacks by first describing the threat model in Section 2.1, followed by introducing the attack methodologies and the attack testbeds setup in Section 2.2. Next, we introduce the free-riding attacks against real-world ISPs in Section 2.3 and defense solutions which are research prototypes in Section 2.4. Finally, we analyze the severeness of free-riding attacks in Section 2.5.

2.1 Threat Model

In this section, we introduce the threat model by first describing the normal zero-rating charging flow in Section 2.1.1, followed by the traffic flow during free-riding attacks of zero-rating services in Section 2.1.2. Then, we discuss how the relationship between CP and ISPs can affect the threat model in Section 2.1.3.

2.1.1 The Normal Zero-rating Charging Flow

We introduce the normal zero-rating charging flow shown in Figure 2.1. There are three parties involved in zero-rating charging flow: CP, ISP, and the user. We introduce them as follows:

- Content Provider (CP). CPs, such as Netflix and Hulu, provide abundant contents, e.g., multimedia, news, games, to the user. In the normal zero-rating charging flow, there are two kinds of CPs. One kind of CPs participates in ISP's zero-rating program while the other kind does not. In Figure 2.1, zero-rating CPs are shown on the upper

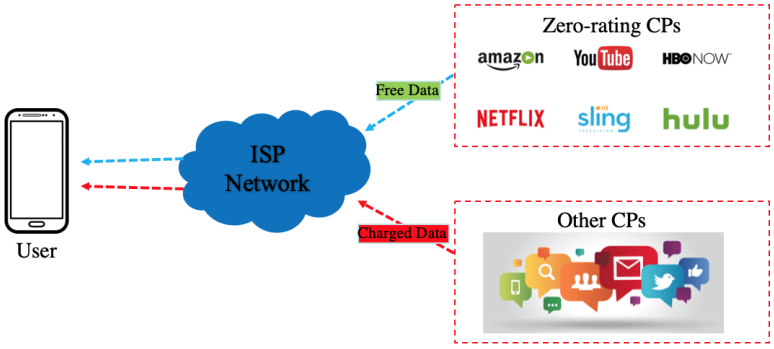


Figure 2.1: Normal Zero-rating Charging Flow

right. ISP zero-rates the traffic from zero-rating CPs to the users. CPs who are not participating in the zero-rating program, are shown as other internet traffic. ISP charges the user streaming from these CPs.

- Internet Service Provider (ISP). ISP provides Internet connectivity between CP and the user. It also controls the billing system. In the normal zero-rating charging flow, there are two kinds of ISPs. One kind of ISPs provides users with the cellular connection such as LTE; the other kind provides WiFi connection such as public WiFi, and airplane cabin WiFi. ISP can differentiate zero-rating and non-zero-rating traffic by the Deep Packet Inspection(DPI) network function. DPI can look for zero-rating packet header, thus identifying the packet. We discover that in zero-rating service, ISP uses 'hostname' field in HTTPs or 'Server Name Indication (SNI)' field in HTTPs to verify the packet's identity.
- User. The user visits the Internet via a *client*, e.g., a mobile phone or a laptop. The user does not need to pay the ISP for the data usage streaming from zero-rating CPs. In contrast, the user needs to pay for streaming from the non-zero-rating CPs.

2.1.2 The Free-riding Attack Flow

Comparing to the normal zero-rating charging flow, we introduce the free-riding attack flow shown in Figure 2.2.

- Content Provider (CP). CP is *benign* in the free-riding attack. CP is the victim, thus hoping to be protected from free-riding attack. Because in real-world zero-rating

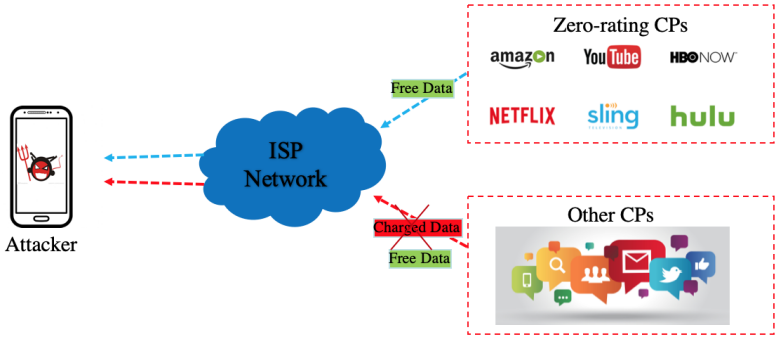


Figure 2.2: Free-riding Attack Flow

service, CP has to pay for the user's data fee to ISP.

- Internet Service Provider (ISP). ISP is also *benign* in the free-riding attack. ISP is trying to protect themself from free-riding attacks launched by the user. Note that we exclude the malicious ISP case, because such a scenario falls back to the traditional end-to-end connection problem where ISP is the man-in-the-middle attacker.
- User. In the free-riding attack flow, the user is potentially *malicious*. The user tries to bypass the ISP's charging policy and get free data. According to characteristics of HTTP/HTTPs, the user has full control of the traffic. The user can modify the contents of the connection, set up a man-in-the-middle proxy between ISP and CP, or set up a masquerade CP server to mislead the ISP.

2.1.3 Intractability in Zero-rating Threat Model

The relationship between CP and ISP makes zero-rating service intractable thus causing the free-riding attack. On the one hand, CP does not trust ISP. Because ISP can be the potential man-in-the-middle attacker, CP needs to use encryption connection to protect users' privacy. On the other hand, CP needs to allow ISP to authenticate the packets in zero-rating services. So in the current landscape, ISP differentiates zero-rating packets by only inspecting few HTTP/HTTPs headers without verifying the traffic with CP. This inspection mechanism leaves chances for the user(attacker) to trick the ISP, thus acquiring free data. Besides, ISP cannot inspect IP address in the header of the HTTP/HTTPs packets to identify CP, because many CPs are hosting their services on

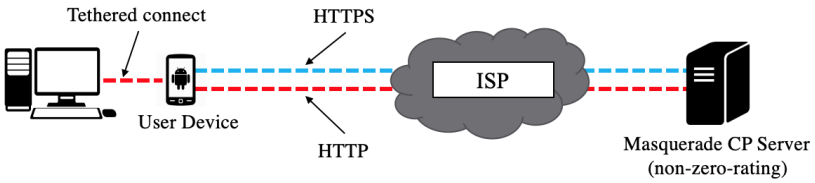


Figure 2.3: Masquerade CP Server Attack

the third-party CDN network. The IP addresses are changing quickly and difficult to synchronize with the ISP's billing system. We discover that ISP only inspects 'hostname' in HTTP or 'SNI' in HTTPs to authenticate zero-rating service.

2.2 Attack Methodologies and TestBeds Setup

In this section, we describe the free-riding attack methodologies along with their testbeds setup. First, we introduce two zero-rating attack methodologies. Then, we introduce the testbeds setup on cellular network and WiFi network to implement these two methodologies.

Real-world ISPs (both cellular ISPs and WiFi ISPs) inspect packets, thus differentiating zero-rating traffic. For the two types of connection, HTTP and HTTPs, ISP inspects different segment. For HTTP connection, ISP inspects the 'hostname' field in HTTP header which carries the CP's identity. For HTTPs connection, as all segments inside of the packet are end-to-end encrypted except the extension segments, ISP inspects the Server Name Indication (SNI) in the HTTPs extension segment. SNI also carries the CP's URL, which is also CP's identity.

Due to the simple inspection tactic, an attacker can launch two types of free-riding attacks. Figure 2.3 shows the first type of free-riding attack, called Masquerade CP Server Attack. Figure 2.4 shows the second type of free-riding attack, called Response Modification Attack.

In Masquerade CP Server Attack, the attacker first sets up a masquerade CP server in the cloud. Then, the attacker uses a computer tethered to a mobile device thus creating HTTP/HTTPs communication with the masquerade CP server. During the HTTP communication, the attacker can modify the 'hostname' field with zero-rating

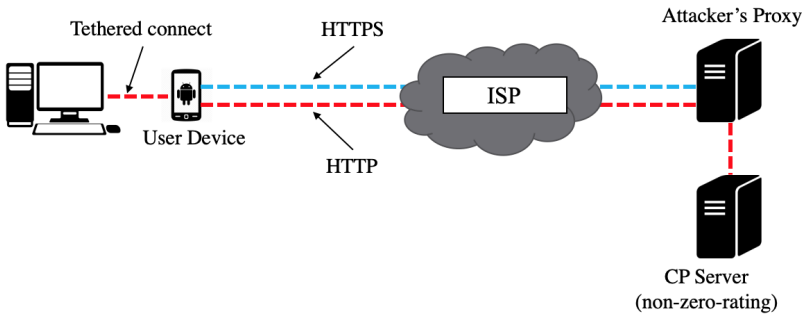


Figure 2.4: Response Modification Attack

identifier in HTTP 'GET' or 'POST' packets on both client side and masquerade CP server side. During the HTTPs communication, the attacker can modify the SNI field with zero-rating identifier. Since ISP inspects the fraud zero-rating identifier segment, it zero-rates the traffic. For example, 'video.youtube.com' is a zero-rating identifier segment as user visits youtube.com. Attacker modifies the packets to the attacker's masquerade CP server ('foo.com') with this zero-rating identifier. Because the packet routing is based on IP address, the packets can still reach the attacker's masquerade CP server. But ISP does not charge this traffic since it inspects the zero-rating identifier.

In Response Modification Attack, the attacker first creates a proxy node between the ISP and non-zero-rating CPs. This proxy is a transparent proxy set in between the connection to modify the CP's response. The modification is intuitive. For HTTP traffic, it modifies the 'hostname' in 'POST' or 'PUT' packet header with the zero-rating identifier. For HTTPs traffic, because the mobile users are malicious, they can decrypt the HTTPs connection, thus sharing the session key with the proxy. The proxy then uses the session key to open up packets, modify them with the zero-rating segment, and then encrypt them again. For example, the user (attacker) visits a non-zero-rating CP website, 'www.nonzero.com'. The traffic is handled by the attacker's proxy in between. The user shares the session key with the proxy. The proxy modifies the response packets from CP to the user with the zero-rating segment 'video.youtube.com.' As the modified packets go through the ISP, ISP zero-rates the traffic, since it inspects the zero-rating identifier.

To demonstrate our methodologies,, we implement the Masquerade CP Server Attack

and Response Modification Attack in both cellular ISP and WiFi ISP.

First, we describe the testbed setup in cellular network. To demonstrate the Masquerade CP Server Attack, we first create the masquerade CP server in the cloud. Next, we use a computer tethered to the cellular ISP's mobile phone, thus connecting the masquerade CP server. The computer can modify the HTTP and HTTPs packets with the zero-rating identifier. To demonstrate the Response Modification Attack in cellular network, we first create the proxy server in between ISP and normal non-zero-rating CPs. Next, we use a computer tethered to the cellular ISP's mobile phone, thus connecting the masquerade CP server. Note that, the computer can decrypt the HTTPs connect and share the session key with the proxy. The proxy modifies the HTTP and HTTPs response packets with the zero-rating identifier.

Next, we describe the testbed setup in WiFi network. The WiFi attack testbeds are similar to the cellular network testbeds except for the wireless connection method. In WiFi network, instead of tethered connection, we directly use computer to connect WiFi access point thus launching the attacks.

2.3 Case Studies on Free-riding Attacks against real-world ISPs

In this section, we introduce the free-riding attacks against real-world ISPs based on our attack testbeds. First, we describe the free-riding attack against 3 cellular ISPs in Section 2.3.1. Next, we describe the free-riding attack against 2 WiFi ISPs in Section 2.3.2. We conclude the test result in Table 2.1.

2.3.1 Real-world Cellular Networks Attacks

We launch the zero-rating attacks on 3 real-world cellular ISPs: Celluar ISP A in the United States which allows its user to visit over 100 CPs content free of charge, Celluar ISP B in China which provides a zero-rating music streaming service, and Celluar ISP C in China which offers a zero-rating video streaming service. In each test, we use the volume of charged data to verify whether the attack succeeds. Table 2.1 shows the overall results: except for those unavailable cases, all zero-rating programs of real-world

Table 2.1: Summary of the Attacks on Real-world ISPs.

		Cellular Network			WiFi Network	
		ISP A	ISP B	ISP C	ISP D	ISP E
Unencrypted traffic	CP masquerade	✗	✗	N/A	✗	✗
	Response modification	✗	✗	N/A	✗	✗
Encrypted traffic	CP masquerade	✗	N/A	✗	N/A	✗
	Response modification	✗	N/A	✗	N/A	✗

✗: The ISP is vulnerable to that free-riding attack; N/A: Corresponding zero-rating service is not available.

cellular ISPs are vulnerable to both types of free-riding attacks. We introduce the detail as follows:

Celluar ISP A The Celluar ISP A provides a program allowing its user to visit more than 100 content providers, such as HBOgo.com and history.com, over LTE free of data charge. We purchase a Celluar ISP A's SIM card with 6 GB data per month and opt-in to the zero-rating program. The amount of usage is measured by dialing Celluar ISP A's self-service code (#932#), which provides two values: the amount of total usage and the amount for charged usage. We use Wireshark in this computer to capture the zero-rating identifiers. We get 'www.hbo.com' as the HTTP identifier and 'video.youtube.com' as the HTTPs identifier.

To launch the Masquerade CP Server Attack on Cellular ISP A, we establish a Node.js server in Digital Ocean Cloud hosting HTTP and HTTPs video services. We then use our computer tethered to the Celluar ISP A's cell phone. Next, we modify all the 'GET' packets, which are sent to the masquerade CP server, with the zero-rating segment. We observe the data volume on the mobile account. Although the total volume amount increased, the charged the data volume remains unchanged.

To launch the Response Modification Attack on Cellular ISP A, we first establish the transparent proxy server in AWS. Next, we use our computer tethered to the Celluar ISP A's cell phone, and send all request packets from our computer through this transparent proxy to the non-zero-rating CPs, so that the response packets can also go through this transparent proxy. At this time, the transparent proxy modifies the packets with the zero-rating identifier which is same as the first attack. We observe that the data volume on the cell phone. The charged the data volume still remains unchanged.

Celluar ISP B The Celluar ISP B in China provides a program allowing its users to stream 'MIGU music' free of data charge over LTE. We purchased a Celluar ISP B's SIM card and opt-in to this zero-rating program. We check the data volume from the ISP's self-service portal. We use Wireshark in this computer to capture the zero-rating identifiers. Based on Wireshark result, we notice that this zero-rating service only provides HTTP connection. The HTTP identifier is 'music.migu.com'.

To launch the Masquerade CP Server Attack on Cellular ISP B, we also establish an HTTP Node.js server for video streaming in Alibaba Cloud. Note that although MIGU music provides music stream, we can still use video file, since ISP cannot tell the packet properties above TCP level. Next, we use our computer tethered to the Celluar ISP B's cell phone. Later, we modify the 'GET' packet with the zero-rating segment on our computer and send to the masquerade CP server. We observe that the charged data volume from ISP portal remains unchanged.

To launch the Response Modification Attack on Cellular ISP B, we first establish the transparent proxy in Alibaba Cloud in China. We then use our computer tethered to the Celluar ISP B's cell phone. Next, we send all HTTP request packets from our computer through this transparent proxy to the non-zero-rating CPs, and let the proxy modifies the response packets with the zero-rating identifier. We observe the charged data volume still unchanged.

Celluar ISP C The Celluar ISP C in China collaborates with the biggest video CP providing a zero-rating video stream program to its users. We get the SIM card from Celluar ISP C and opt-in to the zero-rating program. We check the data volume from the ISP C's online billing system. We use Wireshark in the tethered computer to capture the zero-rating identifiers. Based on Wireshark result, we notice that this zero-rating service only provides HTTPs connection. The HTTPs identifier is 'video.tencent.com'.

To launch the Masquerade CP Server Attack on Cellular ISP C, we set up our video server in Alibaba Cloud. Then, we use our computer tethered to the Celluar ISP C's cell phone. Next, we modify the HTTPs 'SNI' header with the zero-rating identifier on our computer and send to the masquerade CP server. We check the billing system and find that this traffic is uncharged.

To launch the Response Modification Attack on Cellular ISP C, we set up the proxy in Alibaba Cloud. We then use our computer tethered to the Celluar ISP C's cell phone. Next, the computer as the HTTPs client shares the session key with the proxy. The proxy uses this session key to decrypt and modify packets with zero-rating SNI identifier. We also check the billing system and find that this traffic is uncharged.

2.3.2 Real-world WiFi Networks Attacks

We lanuch the Masquerade CP Server Attack and Response Modification Attack on 2 real-world WiFi ISPs: WiFi ISP D providing public WiFi at airport and WiFi ISP E providing Cabin WiFi in the airliner.

Public WiFi ISP D For the public WiFi attack, we select one of the major WiFi ISPs in an international airport of Chicago, named WiFi ISP D. WiFi ISP D provides a free WiFi network for 30 minutes and then charges the users. After 30 minutes, it only provides passengers to visit major airlines' websites, such as united.com, to check flight status for free. If passengers want to have full access to the Internet, they have to pay by hours. We also define this as a zero-rating service, since some of the access is zero-rated while others have to be paid. To launch the attack, we first wait for the WiFi to pass the free 30 minutes and confirm that we can only access the dedicated website, e.g. 'united.com'. Then, we use Wireshark to capture the packet to get the header information in the packet in both HTTP and HTTPs. We get 'www.united.com' in the 'hostname' header of HTTP and 'SNI' in the HTTPs.

In the first scenario, we send out 'GET' request with the modified the zero-rating header to the masquerade CP server, i.e., our video server in both HTTP and HTTPs. We observe that even after 30 minutes free time, we can still access to our server using the modified request.

In the second scenario, we use the proxy to modify the response packet from any non-zero-rating CP server with the zero-rating segment in HTTP and HTTPs. we observe that we can successfully access the CP server.

Cabin WiFi ISP E In December 2016, we tested the Cabin WiFi on a flight from Newark, NJ to Miami, FL. The WiFi service is provided by WiFi ISP E. WiFi ISP E provides free WiFi networks when users visit certain partners' websites, such as united.com and hertz.com. The results were measured based on whether a connection to a non-partners' website succeeds.

We deployed the attack similar to how we peformed such attack in the public WiFi. First, we connect to the cabin WiFi and verify that we can only access the cabin entertainment and certain partners' websites. we do not have access to the full Internet at this time. Then, we use Wireshark to capture the header such as 'hertz.com'. We then modify the packet in the two scenarios with the zero-rating identifier and observe the result. We note that in-cabin WiFi only allows us to connect the HTTP-based website. If we connect to the HTTPs-based website, it will also redirect our request to the HTTP-based website. We think that HTTPs consumes more bandwidth which is inefficient for satellite communications.

In the first scenario, we modify the 'GET' request which routes to our video masquerade CP server with the 'hertz.com' segment in 'hostname' of HTTP. We observe that we can successfully access our video server.

In the second scenario, we use the proxy to modify the response packet from any non-zero-rating CP server with the zero-rating segment in HTTP and HTTPs. we observe that we can successfully access the CP server.

To summarize, Table 2.1 shows that we can launch a free riding attack on two scenarios with both HTTP and HTTPs connection on cellular network ISPs and WiFi network ISPs.

2.4 Attack on Defensive Research Solutions

In this section, we introduce the free-riding attacks against two defense solutions. Specifically, we test two prototypes: Network Cookies [51] and IP Whitelist. First, we describe the two defensive research prototypes respectively in Section 2.4.1. Next, we introduce how to hack the research prototypes and launch the free-riding attack in Section 2.4.2. We conclude the test result in Table 2.2.

2.4.1 Introduction of Defensive Research Solutions

We describe the Network Cookies and IP Whitelist in this section respectively.

Network Cookies In paper "Neutral Net Neutrality" [51], Yiakoumis et al. gave out cookie-based middle-box authentication framework. Shown in Figure 2.5, Network Cookies solution consists of four components: user(client), Middle-Box (ISP), CP server and Cookie descriptor server. The workflow of this authentication framework has three phases, described as follows:

Phase 1. The users and their clients inquiry CP service through discovery protocols such as DHCP and DNS. The Cookie Descriptor Server can associate with the Discovery Protocols Server and get information of the the inquiry URL. The Cookie Descriptor List is a pre-setted whitelist of zero-rating services. If the service is on the list, phase 2 can be triggered.

Phase 2. Cookie Descriptor Server sends the certain Cookie to the user's client. The user's client inserts the Cookie into the outgoing packets. There are several places to insert the Cookie: 1) at the application layer (e.g., HTTP header or TLS handshake extension); 2) at the transport layer(e.g., TCP long option, UDP based header); 3) at the network layer(e.g., IPv6 extension header).

Phase 3. The user's client sends the packets with the Cookie to the CP server via the ISP network. ISP inspects the Cookie, thus zero-rating the traffic.

IP Whitelist IP Whitelist is a simple technique that ISP uses for controlling the traffic. It is implemented as a subfunction in the DPI component of the ISP's core network. The IP Whitelist function only allows the traffic tagged with specific IP-address to pass or be zero-rated. Facebook Project Zero [11] is one implementation of the IP Whitelist solution. In this project, Facebook creates a control channel with the ISPs to dynamic synchronize its zero-rating IP-address with the ISP's DPI whitelist.

2.4.2 Attack on Defensive Research Solutions

We introduce the attack on the Network Cookies and the IP Whitelist in this section respectively.

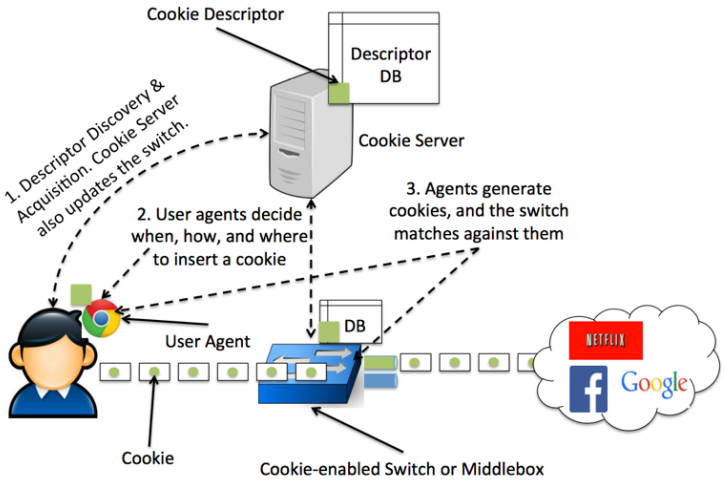


Figure 2.5: Network Cookie Architecture

Attack on Network Cookies We launch free-riding attacks against Network Cookies. Because the cookie server does not bind issued cookies to zero-rating traffic, a user can abuse the cookie for any traffic to the server. Furthermore, the communication integrity between zero-rating CP and user can be compromised by a man-in-the-middle attacker. Because the cookie does not validate the contents conveyed in the communication. We show that both their implementation and protocol are vulnerable to free-riding attacks. Details about the vulnerability in their protocol can also be found in Chapter 5. We now discuss the attack.

Specifically, we obtain the original implementation from the authors of Network Cookies paper and deploy the implementation in our lab environment. Their client, ISP middlebox, and cookie server are installed at three lab servers with Ubuntu 16.04 operating systems. The CP server's hostname is registered in the Cookie Server. The client asks for Network Cookies from the Cookie Server during DNS requests. The ISP set in the middle inspecting packets and verify Network Cookies via function called $verifycookie$.

We then perform the free-riding attacks and show that the Network Cookies prototype is vulnerable. First, we create a malicious non-zero-rating application in the client. We let this application to bind a zero-rating Network Cookie by attaching a valid cookie in the HTTP 'network-cookie' extension field. The results show that the ISP marks the

traffic as zero-rated. Second, we attack the cookie integration by the Response Modification Attack. We create the proxy between ISP and CP to inject a non-zero-rating photo to a cookie enhanced zero-rating traffic. The result shows that ISP still zero-rates the total traffic including the inserted non-zero-rating content.

Attack on IP Whitelist We introduce the Masquerade CP Server Attack and Response Modification Attack against IP Whitelist. First, we set up the testing environment for IP Whitelist. We create a zero-rating CP server in a campus network and a client in DigitalOcean Cloud. Next, we set up a computer in campus network representing the ISP in between the client and the CP server. This ISP contains the IP Whitelist that locks on the CP server's IP address.

In Masquerade CP Server Attack, we first set up a masquerade CP server in a different campus computer whose IP address is not on the ISP's whitelist. Then, the client—which cooperates with the masquerade CP server—establishes a connection with the zero-rating CP server. Once the connection is created, the client shares the connection information with the masquerade CP server. Such information includes the HTTP/HTTPs sequence number, the acknowledgment number, the destination port, the source port, and the TCP flags. The masquerade CP server, based on this information, crafts TCP packets to mimic the zero-rating CP server's behavior, thus communicating with the client. As a result, we observe that the ISP does not realize that the connection session is shifted from the zero-rating CP to the masquerade CP, thus still zero-rating the traffic. Our experiment result further shows that we only need a small amount (386 bytes) of charged traffic to initiate the free-riding attack. This small amount of charged traffic is used in the process of the communication information sharing from the client to the masquerade CP server. Another thing worth noting is that the masquerade CP server can also embed free-riding traffic in TCP retransmission packets. In this case, even if ISP checks the traffic volume, it cannot notice of the free riding attack.

In Response Modification Attack, we first set up the proxy node between the zero-rating CP server and the client. This proxy node is also a computer in the campus network. This proxy can inject and modify content inside the connection between zero-rating CP server and client. Because the ISP only inspects IP address based on the

Table 2.2: Summary of Attacks on Research Prototypes.

		Prototypes	
		Network Cookies	IP Whitelist
Unencrypted traffic	CP masquerade	✗	✗
	Response modification	✗	✗
Encrypted traffic	CP masquerade	✗	✗
	Response modification	✗	✗

✗: The ISP is vulnerable to that free-riding attack; N/A: Corresponding zero-rating service is not available.

IP Whitelist. It cannot notice any of the non-zero-rating content inside of the traffic. Interestingly, in previous Masquerade CP Server Attack, the packet integrity between the client and the real CP is also violate, as the client can directly receive the crafted packet from the masquerade CP server, .

2.5 Understanding the Severeness of Free-riding Attacks

In this section, we measure the severeness of the free-riding attacks. Specifically, we receive one-month usage data from the Cellular ISP B and estimate the amount of free-riding traffic. We understand that this is a difficult task because if we can accurately measure free-riding attacks, such approach can be used for defensive detection as well.

The detailed steps for estimating the amount of free-riding traffic are introduced as follows. First, we calculate the average amount of zero-rated data for a normal user, which is roughly 300MB/month. Second, we filter those users whose zero-rating traffic amount are significantly higher than the normal 300MB/month. We set the threshold of the abnormal amount of zero-rated data as 3GB/month. Lastly, we manually count the invalid zero-rating users and accumulate their total amount of free-riding traffic.

Our manual analysis is performed on the billing data of Cellular ISP B's network in January 2016. The result reveal 71TB free-riding traffic, equaling to half a million US dollar based on Cellular ISP B's charging rate. Note that one interesting finding is that some users consumed more than 30GB zero-rating data in the zero-rating music stream program per month, which is technically impossible because that means those users need to listen to music for more than 24 hours per day.

Chapter 3

ZFree Overview

To defend against free-riding attacks, we introduce the Zero-rating FRamework with thrEe partiEs (ZFree). ZFree provides a secure packet authentication in the zero-rating network. In this chapter, we give a high-level overview of how ZFree is organized and operated. First, we introduce the ZFree system architecture. Second, we describe the ZFree deployment model. We demonstrate how ZFree can be merged into the cellular network and WiFi network with minimum modification. Lastly, we discuss several ZFree use cases other than defending against free-riding attacks.

3.1 System Architecture

At high level, ZFree contains *data plane* and *control plane*. As previously discussed, a traditional zero-rating framework involves three parties: user, ISP, and CP. User connects to the Internet via the wireless access network, such as cellular base station (eNodeB) and WiFi Access Point (AP). ISP provides a core network for packet routing and inspects packet for differentiate zero-rating traffic. CP hosts multimedia zero-rating contents in a Data Center (DC). The communication between client and CP via the ISP forms into the *data plane*. On top of the data plane, it is the ZFree *control plane*. ZFree *control plane* contains two pluggable components, called Server Agent and ISP Assistant. The Server Agent is deployed in CP side and responses for: 1) sniffing packets at the CP gateway; 2) converting the packet into the hash code; 3) sending the hash code to the ISP Assistant via the ZFree control plane protocol. The ISP Assistant is

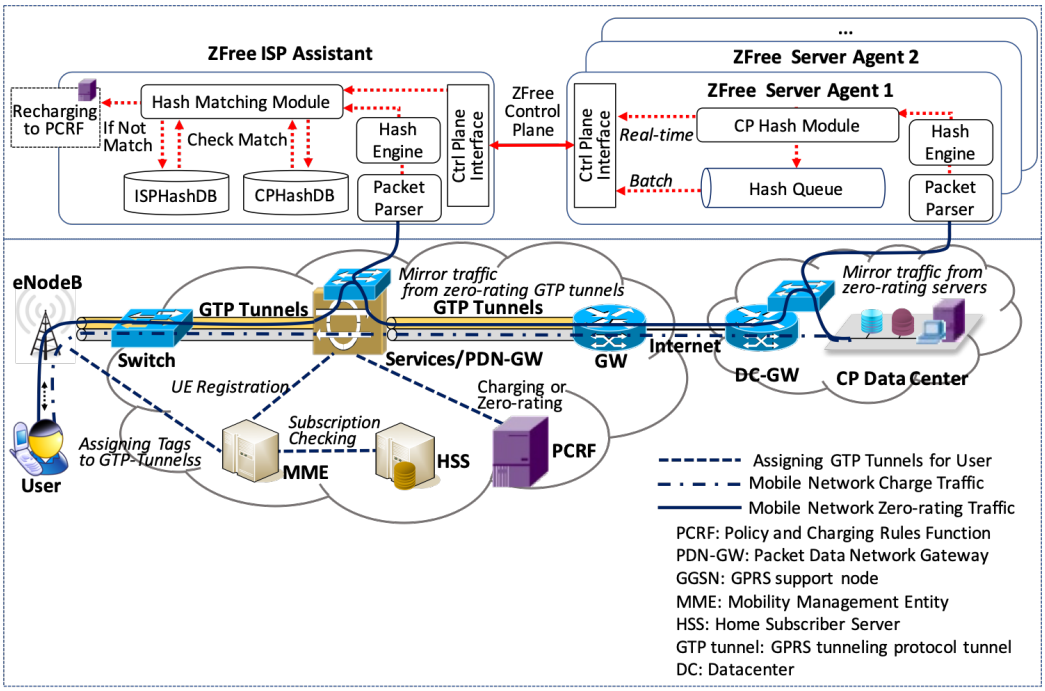


Figure 3.1: ZFree's Architecture over Cellular Network

deployed in ISP side and responses for: 1) communicating all the Server Agent from different CPs; 2) authenticating zero-rating packets; 3) communicating with the Policy and Charging Rules Function (PCRF) in the ISP core network's billing system.

ZFree supports two modes for zero-rating services, i.e., non-blocking mode and blocking mode. The non-blocking mode only sniffs and verifies packets without dropping them. If the packets are invalid (from free-riding attack), ZFree informs PCRF to charge this packet. In contrast, the ZFree blocking mode intercepts the packets if ISP assistant inspects invalid packets. As mentioned, the non-blocking mode can be used for cellular networks, because ISP controls the user's mobile bill for visiting zero-rating contents. The blocking mode can be used for WiFi networks as the ISP controls the accessibility and may not be able to charge the users later.

3.2 Deployment Model

We now discuss how to deploy ZFree in real-world ISPs over LTE or WiFi. ZFree's cellular deployment is shown in Figure 3.1. ZFree's WiFi deployment is shown in Figure 3.2. In these two deployment models, the Server Agent is deployed at the CP

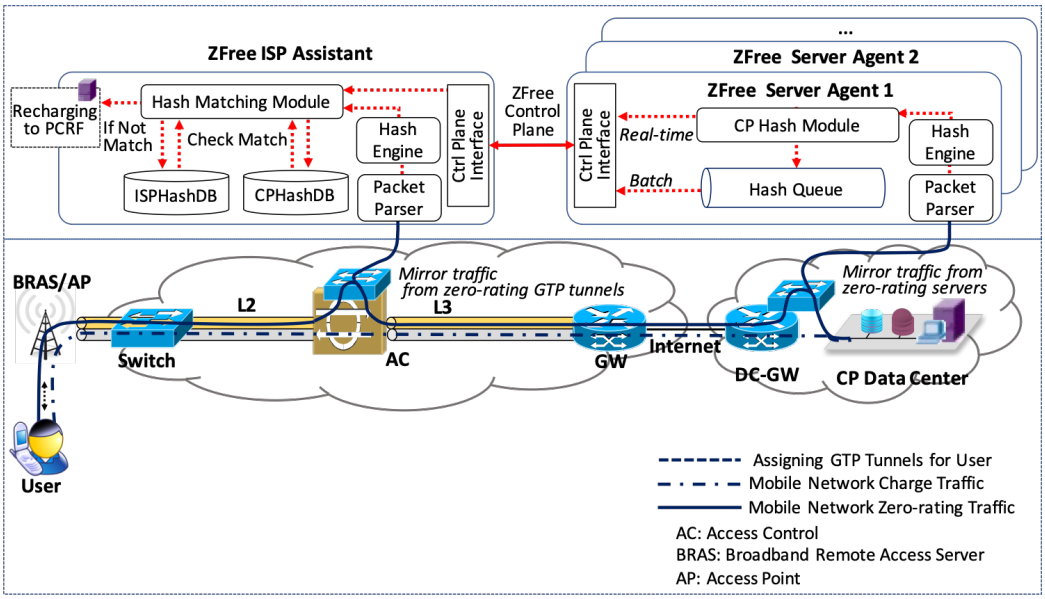


Figure 3.2: ZFree's Architecture over WiFi Network

gateway while the ISP Assistant is deployed based on different ISP network topologies as discussed below.

3.2.1 Cellular Network

Figure 3.1 shows the ZFree deployment model in the cellular network. The ISP Assistant is be deployed at the ISP core network, specifically the Packet Data Network (PDN) Gateway (P-GW) in LTEnetwork and GPRS support node (GGSN) in 3G network. The Server agent is deployed at the gateway router of CP data center. As user connects to the Access Network, the enodeB in the Access Network aggregates the packet to the Service-Gateway(S-GW). Then, S-GW aggregates the traffic and forwards to the P-GW or GGSN for packet switching. Here, the ISP Assistant sniffs packet.

3.2.2 WiFi Network

Figure 3.2 shows the ZFree deployment model in WiFi network. The ISP Assistant is deployed at the Access Controller (AC) which is the central control node of packet authentication and access management. When the user connects to the network through Access Point(AP), the AC inspects packets and interacts with ISP Assistant by blocking mode or non-blocking mode for zero-rating packet authentication.

3.3 Use Cases

Apart from providing zero-rating services and mitigating free-riding attacks, we envision that ZFree can also be used for other purposes. We discuss two use cases as follows.

3.3.1 Parental Control

Parental control is a feature to prevent underage users from accessing certain inappropriate contents. We can use ZFree blocking mode to serve as the parental control filter in the ISP's core network. ZFree allows parents to edit the control policies and monitor the traffic in underage users' devices.

3.3.2 Policy and Charging Control (PCC)

Policy and Charging Control (PCC) allows ISP to set Quality of Service (QoS) and Charging policy based on different contents. The ISP can use ZFree ISP Assistant to manage all the PCC policies and control charging rules.

Chapter 4

ZFree System Design

In this section, we introduce the ZFree system design by first describing the two ZFree components in Section 4.1. Next, we walk through the ZFree control plane protocol in Section 4.2. Finally, we introduce the critical algorithms in ZFree control plane protocol in Section 4.3.

4.1 ZFree components

ZFree has two components in the control plane, the Server Agent and the ISP Assistant. Server Agent is deployed in every CP gateways and ISP Assistant is deployed inside of the ISP. We introduce them respectively as follows.

4.1.1 Server Agent

Server Agent is deployed at CP side. All the CPs in the zero-rating program are equipped with the Server Agent. The Server Agent can be deployed as an attachment component of the CP Gateway or as a virtual function embedded inside of the CP Gateway.

Server Agent is in charge of 4 tasks: 1) establishing a connection with the ISP Assistant; 2)verifying the ISP identity; 3) sniffing packets from the CP data plane thus converting the packets into keyed hash code; 4) sending the keyed hash to the ISP Assistant.

4.1.2 ISP Assistant

ISP Assistant is deployed at the ISP side. ISP Assistant can be deployed as an attachment component of the ISP core network. Since the ISP's core network has large traffic capacity, we also need to consider the system scalability. To scale ZFree, we can deploy ZFree as several virtual network slicing components in the ISP core network [32].

ISP Assistant is in charge of 5 tasks: 1) maintaining the connection with all the Server Agents; 2) verifying the CP identity; 3) sniffing packets from ISP core network thus converting the packets into keyed hash code; 4) matching the two keyed hash code, one is received from the Server Agent while the other one is sniffed from the core network; 5) communicating with PCRF to update the billing information.

4.2 ZFree Control Plane Protocol

All the ZFree control processes are done by the ZFree control plane protocol between the Server Agent and the ISP Assistant. In this section we introduce the ZFree Control Plane Protocol by describing the two phases of the protocol, **Setup Phase** in Section 4.2.1 and **Control Phase** in Section 4.2.2 . Each phase consists of several steps. Figure 4.1 shows the details of the protocol.

4.2.1 Setup Phase

The setup phase is the first phase of ZFree control plane protocol. During this setup phase, ISP Assistant and Server Agent establish communication by agreeing on a list of options, such as cipher suite and connection type, exchanging session keys and then verifying each other's identity by the certificate. In detail, the setup phase consists of two steps, handshake and identity verification.

❶ **Handshake**: In this step, the ISP Assistant and the Server Agent exchange setup options via "ISP HELLO" and "SA HELLO" messages. Specifically, the exchange messages include session id, a random number for computing keys, cypher suite options, ZFree versions, connection types (e.g., blocking vs. non-blocking ,and real-time vs. batch), policies (e.g., zero-rating and parental control), and a list of IP address ranges

(e.g., 88.88.0.0/16) locking on the zero-rating clients. Next, based on the two "HELLO" messages, both the ISP Assistant and the Server Agent compute a traffic key used for establishing the communicating session.

❷ **Identity Verification**: In this step, the ISP Assistant and the Server Agent verify the identity by verifying each other's certificate. If the verification succeeds, the ISP Assistant and the Server Agent computes the session key based on the certificate and the traffic key to establish the final communication.

4.2.2 Control Phase

After the setup phase, the communication between the ISP Assistant and the Server Agent is established. ZFree moves to the control phase and ready to verify the zero-rating service. In detail, the control phase consists of 3 steps, introduced as follows:

Note that Step ❹ and Step ❺ are parallel steps. The ZFree control plane protocol in setup phase chooses either ❹ or ❺ via "HELLO" messages. The Step ❹ is communication in Realtime-type where the Server Agent sends every keyed hash to ISP assistant in real-time. Step ❺ is communication in Batch-type where the Server Agent gathers a pack of keyed hashes and waits for the ISP Assistant to poll periodically.

❸ **Communication (Realtime-type)**:

In this step, ZFree verifies the zero-rating packets in realtime-type. When the user streaming zero-rating contents from the CP, CP server sends the packets from CP datacenter through ISP to the user. In this flow, the Server Agent first sniffs all the response packets. Next, the Server Agent converts every packet into hash code using the control plane session key. Then the Server Agent sends the hash code to the ISP Assistant via "HASHPUSH" message constantly. Note that the hash code has a smaller size than the original packet to reduce the control plane overhead. For example, a 1024KB packet can be converted into a 64-byte hash code. Later, when ISP receives this "HASHPUSH" message, it matches the hash code with the one sniffed from ISP core network. If there is a match, ISP Assistant marks this packet as a valid packet, thus zero-rating it. Otherwise, ISP Assistant drops the packet in blocking mode (❸a ; informs PCRF to charge this packet in non-blocking mode (❸b).

❹ **Communication (Batch-type)**: In this step, when the control plane connection is in Batch-type, the Server Agent gathers all the hash code in group and waits for ISP Assistant to pull via "HASHPULL" message. The hash verification process in the ISP Assistant is the same as the realtime-type in ❸.

❺ **Status Report and Heartbeat Synchronization**: This step is used for maintaining the control plane communication. It's a daemon process with heartbeat message named "STATUS". Via this message, the ISP Assistant and the Server Agent can report the current status to each other, e.g., unmatched hashes, for diagnosis purpose.

4.3 ZFree Algorithm Design

In this section, we introduce two algorithms in the ZFree control plane protocol.

4.3.1 Server Agent Algorithm

The Server Agent Algorithm is used inside of the Server Agent. The algorithm is shown in Figure 4.2. The Server Agent Algorithm consists of two main functions, *Handshake* function and *ProcessHash* function. It also contains a daemon function called *ControlPlaneListener* for maintaining the control plane communication.

The *Handshake* function is used during the ZFree setup phase where the Server Agent establishes the connection with the ISP Assistant. Particularly, the Server Agent exchanges ZFree versions, connection types, policies as well as Diffie-Hellman cipher suites, pre-shared key and a random number with the ISP assistant via an "HELLO" message (Line 3–6). Based on the agreed Diffie-Hellman cipher, the Server Agent computes the traffic key (Line 4) and then sends its certificate to the ISP assistant (Line 8). After that, the Server Agent generates a finish message with its private signature (Line 9–10). At the same time, the Server Agent also generates a session key based on key share, master secret and traffic key (Line 11). Next, the Server Agent waits for the ISP certificate (Line 12–13) and the ISP finish message (Line 14–15). Lastly, the server agent verifies ISP's identity with the root CA: if verified, it calls *ProcessHash* to start ZFree control phase; otherwise, it terminates the socket (Line 16–20).

The *ProcessHash* function is used during the ZFree control phase. To ensure

system performance and scalability, $ProcessHash$ function is operated using multiple threads. In detail, the $ZFreeParseModule$ first sniffs each packet (Line 22) from the data plane. Then, the $ZFreeHashEngine$ calculates the keyed hash code of these packets using HMAC function (Line 23). Next, based on the connection type, the server agent chooses to send the keyed hash to ISP Assistant in realtime mode (Line 25–27) or in batch mode (Line 28–30).

4.3.2 ISP Assistant Algorithm

The ISP Assistant Algorithm is used inside of the ISP Assistant. The algorithm is shown in Figure 4.3. The ISP Assistant Algorithm consists of 4 main functions which are $Handshake$ function, $ISPProcessHash$ function, $StatueCheck$ function, and $DistributedHashMatch$ function. It also contains a daemon function for receiving the hash sent from the Server Agent, called $ReceiveSAHash$.

The $Handshake$ function is used during ZFree setup phase. Specifically, the ISP assistant exchanges "HELLO" messages with the Server Agent (Line 3–5), computes the traffic key (Line 6), decrypts the Server Agent's certificate and private signature (Line 11), and verifies the Server Agent's certificate (Line 12). Next, the ISP Assistant computes the session key (Line 13) and sends its own certificate, private signature, and a finish message to the Server Agent (Line 14–17). Lastly, after the connection is established, the ISP Assistant sends a "STATUS" messages to the server agent (Lines 38–39) for confirmation.

The $ISPProcessHash$ function (Line 23–29) is used during ZFree control phase. The $ISPProcessHash$'s main task is to interact with the ISP core network to sniff packets and convert packets into the hash code. Specifically, the $ZFreeParseModule$ first sniffs packets from the ISP core network (Line 24). Next, $HMAC$ function converts the packets into hash code (Line25). Here, the $ISPProcessHash$ function calls $DistributedHashMatch$ function (Line 40-41) to match the hash code with the one received from the Server Agent. If there is a match, the ISP Assistant informs PCRF for updating zero-rating information(set whether free or charge) (Line 26-29). Otherwise, ISP Assistant saves the hash code into the database and waits for the further matching

(Line 29).

The *DistributedHashMatch* function is a hyperfunction that is used for matching the hash code from the Server Agent with the one in the ISP Assistant. This function is also operated in multithreading to improve the performance.

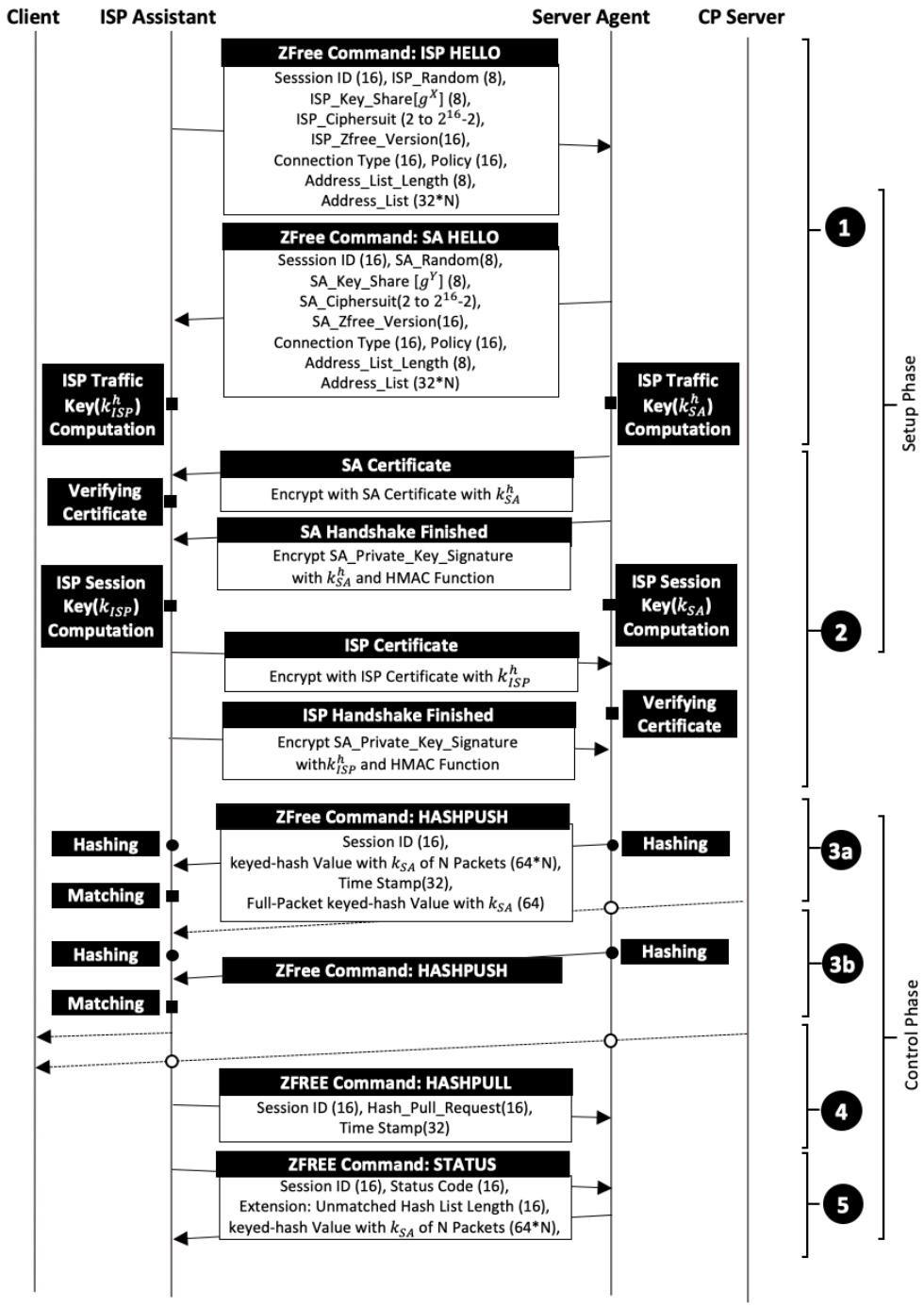


Figure 4.1: ZFree Control Plane Protocol

Algorithm 1: Server Agent Algorithm

```
Format ZFree::Command refers to these defined in ZFree Protocol.
Input: RawPacket, ZFree :: Command
struct {
    double Session_ID, SA_Random, SA_Key_Share, ZFree_Version
    Connection_Type
    float Policy
} SA_Hello
1  Function Handshake():
2      Socket ⟵ Awaiting ZFree :: Command
3      if Control_Plane_Interface (Socket) == ZFree::ISP_HELLO then
4          Compute Traffic_Key based on ISP_Key_Share &
             ISP_Random
5          Negotiate ZFree_Version and Connection_Type
6          Build_and_Send_Packet(ZFree::SA_HELLO)
7          Enc_SA_Cert ⟵ Encrypt(Traffic_Key , SA_Cert)
8          Build_and_Send_Packet(Enc_SA_Cert)
9          Enc_SA_Finish ⟵ Encrypt(Traffic_Key , SA_PKsa_Sign)
10         Build_and_Send_Packet(Enc_SA_Finish)
11         Compute Session_Key based on
             ISP_Key_Share, Master_Secret, Traffic_Key
12     else if Control_Plane_Interface (Socket) == ISP_Certificate then
13         ISP_Certificate ⟵ Decrypt(Traffic_Key, Enc_ISP_Cert)
14     else if Control_Plane_Interface (Socket) == ISP_Finish then
15         ISP_PKisp_Sign ⟵ Decrypt(Traffic_Key, Enc_ISP_Finish)
16         if Verify ISP_Certificate with CA PASS then
17             Thread
                 ProcessHash(Session_ID, Session_Key, Connection_Type)
18         else
19             Build_and_Send_Packet(ZFree::STATUS, disconnect)
20             Socket.close
21 Thread ProcessHash(Session_ID, Session_Key, Connection_Type):
22     Packet_Queue ⟵ ZFreeParseModule(DataPlane_Packet)
23     Keyed_Hash ⟵ ZFreeHashEngine.HMAC(Session_Key,
         Packet_Queue)
24     switch ConnectionType do
25         case Realtime do
26             Build_and_Send_Packet(ZFree::HASHPUSH, Session_ID,
27             Keyed_Hash, Time_stamp, HMAC(this.Packet))
28         case Batch do
29             Hash_Queue.save(Keyed_Hash)
30             Set ControlPlaneListener(Hash_Queue)
31 Function ControlPlaneListener(Hash_Queue):
32     Create Listener = Control_Plane_Interface (ZFree :: Command)
33     switch ZFree :: Command do
34         case ZFree :: HASHPULL do
35             Keyed_Hash ⟵ Hash_Queue.get(Keyed_Hash);
                 HASHPUSH ⟵ Session_ID, Keyed_Hash,
36                       Time_Stamp, HMAC(this.Packet)
                 Build_and_Send_Packet(ZFree::HASHPUSH)
37         case ZFree :: STATUS do
38             process corresponding status
```

Figure 4.2: Server Agent Algorithm

Algorithm 2: ISP Assistant Algorithm

```
Format ZFree::Command refers to these defined in ZFree Protocol.
Input: RawPacket, ZFree :: Command
struct {
    double Session_ID, ISP_Random, ISP_Key_Share, ZFree_Version
    Connection_Type, Address_List_Length
    float Policy
    array[ ] AddressList
} ISP_Hello
1  Function Handshake():
2      Socket ⟵ Establish to Server Agent
3      Build_and_Send_Packet(ZFree::ISP_HELLO)
4      Socket ⟵ Awaiting ZFree :: Command
5      if Control_Plane_Interface (Socket) == ZFree::SA_HELLO then
6          Compute Traffic_Key based on SA_Key_Share & SA_Random
7          Set ZFree_Version and Connection_Type
8      else if Control_Plane_Interface (Socket) == SA_Certificate then
9          SA_Certificate ⟵ Decrypt(Traffic_Key, Enc_SA_Cert)
10     else if Control_Plane_Interface (Socket) == SA_Finish then
11         SA_PKsa_Sign ⟵ Decrypt(Traffic_Key, Enc_SA_Finish)
12         if Verify SA_Certificate with CA PASS then
13             Compute Session_Key based on
                 ISP_Key_Share, Master_Secret, Traffic_Key
14             Enc_ISP_Cert ⟵ Encrypt(Traffic_Key , ISP_Cert)
15             Build_and_Send_Packet(Enc_ISP_Cert)
16             Enc_ISP_Finish ⟵ Encrypt(Traffic_Key ,
                 ISP_PKisp_Sign)
17             Build_and_Send_Packet(Enc_ISP_Finish)
18             Thread ReceiveSAHash()
19             Thread ISPProcessHash(Session_ID, Session_Key)
20         else
21             Build_and_Send_Packet(ZFree::STATUS, disconnect)
22             Socket.close
23 Thread ISPProcessHash(Session_ID, Session_Key):
24     Packet_Queue ⟵ ZFreeParseModule(DataPlane_Packet)
25     ISP_Keyed_Hash ⟵ HMAC(Session_Key, Packet_Queue)
26     if Distributed_Hash_Match(ISP, ISP_Keyed_Hash)==True then
27         PCRF_Charging_Module.Apply(Policy)
28     else
29         ISP_Distributed_HashDB.save(ISP_Keyed_Hash)
30 Thread ReceiveSAHash():
31     if ConnectionType == batch then
32         Build_and_Send_Packet(ZFree::HashPull)
33     SA_Keyed_Hash ⟵ Control_Plane_Interface(ZFree::HashPush)
34     if Distributed_Hash_Match(CP, SA_Keyed_Hash)==True then
35         PCRF_Charging_Module.Apply(Policy)
36     else
37         CP_Distributed_HashDB.save(SA_Keyed_Hash)
38 Function StatusCheck():
39     process corresponding status
40 Function DistributedHashMatch(Party, Keyed_Hash):
41     select database (Party) and matching node (Keyed_Hash&Mask)
```

Figure 4.3: ISP Assistant Algorithm

Chapter 5

Formal Security Analysis

In this chapter, we perform formal security analysis on the three zero-rating frameworks—Network Cookies [51], IP Whitelist [3,16] and ZFree—using ProVerif [14,15]. ProVerif is an automatic cryptographic protocol verifier. We organize this chapter in 3 parts. First, we discuss the verification goals in Section 5.1. Next, we introduce how to formalize the three zero-rating frameworks in Section 5.2. Note that we create the ProVerif models for the three zero-rating frameworks and publish the source code on Github (https://github.com/zfree2018/ZFREE). Finally, we introduce the formal verification results in Section 5.3.

5.1 Verification Goals

We ask ProVerif to verify the following three goals for the aforementioned zero-rating frameworks.

Goal 1: Packet Integrity. We ask ProVerif to verify the integrity of response packets from the CP server to the client. Specifically, the response packet sent from the CP server needs to match with the one received by the client. Note that $endResponseVerif$ and $beginResponseVerif$ are two ProVerif functions in the client and CP server. These two functions interact with each other to verify whether the packet integrity holds in the end-to-end communication.

Goal 2: CP Server Authenticity. We ask ProfVerif to verify whether the CP server identity matches with the ISP zero-rating CP list (pre-set contract list). Similarly,

$endServerVerif$ and $beginServerVerif$ are two ProVerif functions in the ISP and the CP server. These two functions interact with each other to verify whether the CP server identity remains unchanged during the end-to-end communication.

Goal 3: Application Data Secrecy. We ask ProVerif to verify the secrecy of application data between the client and the CP server. The secrecy is to verify whether there exists a man-in-the-middle attacker to modify the packet.

Note that the threat models are different for Goals 1&2 and Goal 3. Goals 1&2 assume that the client is malicious—i.e., even in encrypted mode, all the client-side data including the session key is available to a remote middlebox controlled by the client. Goal 3 assumes that the client is benign and a man-in-the-middle attacker may exist.

5.2 Formal Models

In this section, we introduce the formalization of the three zero-rating models in ProfVerif.

5.2.1 Network Cookies

We model a Network Cookie solution as described in the paper [51]. First, we create the Cookie Server in ProfVerif. This Cookie Server distributes zero-rating Cookies to all the clients. Specifically, when the Cookie Server receives a request from the client, it crafts the Cookie based on the Cookie Descriptor. The Cookie Descriptor allocates a Cookie ID, a Cookie key and a Cookie attribute. Second, the client attaches the Cookie in all the outgoing packets. Third, ISP inspects all traffic and verifies the Cookie.

5.2.2 IP Whitelist

We then model the IP Whitelist based on several industry proposals [3, 16]. We create the ISP node with IP Whitelist function. ISP locks on the IP Whitelist with the CP's IP address. We let ProVerif verify whether there exists other traffic that can jailbreak this whitelist.

Table 5.1: Summary of Formal Verification Results on Network Cookies, IP Whitelist and ZFree.

Goals	ProVerif Queries	Network Cookies [51]		IP Whitelist		ZFree	
		Unencrypted	Encrypted	Unencrypted	Encrypted	Unencrypted	Encrypted
Integrity	$event(endResponseVerif(response))$ ==> $event(beginResponseVerif(response))$	✗	✗	✗	✗	✓	✓
Authenticity	$inj\text{-}event(endServerVerif(server\_identity))$ ==> $inj\text{-}event(beginServerVerif(server\_identity))$	✗	✗	✗	✗	✓	✓
Secrecy	$attacker(AppData)$	✗	✓	✗	✓	✗	✓

✓: the property is satisfied; ✗: the property is not. Unencrypted and encrypted refer to data plane communication.

5.2.3 ZFree

We model ZFree in ProfVerif. We first create two control plane components, the ISP Assistant and the CP Server Agent. Next, we create the setup phase and control phase following the ZFree control plane protocol. The CP Server Agent can sniff all the packets in the data plane channel and send the hash code to ISP Assistant over the control plane channel.

5.3 Verification Results

In this section, we introduce the formal verification results. To summarize, both Network Cookies and IP Whitelist are vulnerable to free-riding attacks, because they cannot preserve either packet integrity or CP server authenticity. In contrast, ZFree can defend against free-riding attacks. At the same time, our verification also shows that none of the three frameworks change the application layer security, because the data secrecy is preserved if traffic is encrypted. An overview of our verification results can be found in Table 5.1. Now let us discuss several example violation outputs found by ProVerif.

Output 1 (Network Cookies): Authenticity Violation. When the ProVerif queries $endServerVerif(server\_identity)$, the outputs show a violation case for Network Cookies. Specifically, the violation shows that an attacker can acquire a zero-rating cookie and send the cookie together with non-zero-rating contents to another CP server.

Output 2 (Network Cookies & IP Whitelist): Integrity Violation. When ProVerif queries $endResponseIntegrity(response)$, it outputs violations for both Network Cookies and IP Whitelists. The violations show that an attacker can obtain the response

packet from a zero-rating CP server, modify the packet to inject contents from another CP server, and then send the modified packet to the client.

Output 3 (IP Whitelist): Authenticity Violation. When ProVerif makes an authenticity query to IP Whitelist, it outputs a violation showing that an attacker can obtain the IP address of the zero-rating CP. The attacker can also insert the IP into other response data from another non-zero-rating CP.

Next, we show that we need to carefully design ZFree so that a simple variation of the protocol may result in an insecure design. We show several possible violations if we modify ZFree model into weak variation.

Output 4 (Weak ZFree *Variation): Integrity Violation.* The first ZFree variation is that we adopt a weaker hash algorithm. Instead of SHA-256 in ZFree control plane protocol, we use SHA-1. When we make an integrity query to ProVerif for this weak variation, ProVerif reports that an attacker can compromise both the traffic key and the session key, and then modify the "HASHPUSH" message to embed its data of non-zero-rated packets.

Output 5 (Weak ZFree *Variation):* The second ZFree variation is that we skip the encryption of the control plane $HashPush$ packet. When we make an integrity query to ProfVerif, it reports a violation, in which an attacker can obtain the $HashPush$ message, modifies the message, and then changes the corresponding data plane packet as well.

Chapter 6

Experiment and Evaluation

In this chapter, we describe the experiments we design to evaluate the overhead caused by ZFree and present our evaluation results. First, we describe the experimental setup in Section 6.1. Next, we introduce the evaluation results in Section 6.2.

6.1 Environment Setup

We set up the environment by first implementing ZFree. Our ZFree implementation includes 1,890 lines of code (LoC), i.e., 1,100 LoC for the ISP Assistant and 790 LoC for the Server Agent. Additionally, we also set up a demo website with 836 LoC. Next, to establish the experiment, we set up an LTE network environment using ns-3 [6] and a WiFi network environment using Mininet-WiFi [4]. Both network simulators are popular and adopted by many existing works [34, 36, 49]. The LTE network consists of user equipment (UE), eNodeBs, PDN gateway, MME, and HSS while the WiFi network consists of the user equipment, access point (AP) and routers. Note that we use ZFree's non-blocking mode in cellular ISP networks while use ZFree's blocking mode in WiFi ISP network. The two environments are shown in Figure 6.1 and Figure 6.2.

Cellular Environment We establish the cellular environment shown in Figure 6.1. The environment is built on two physical machines with 3.2 GHz Intel due-core i7-6950x CPU, 32GB memory and Ubuntu 16.04 LTS OS. The cellular network environment consists of the ISP core network and two groups of 1,200 user equipment (UEs) (shows

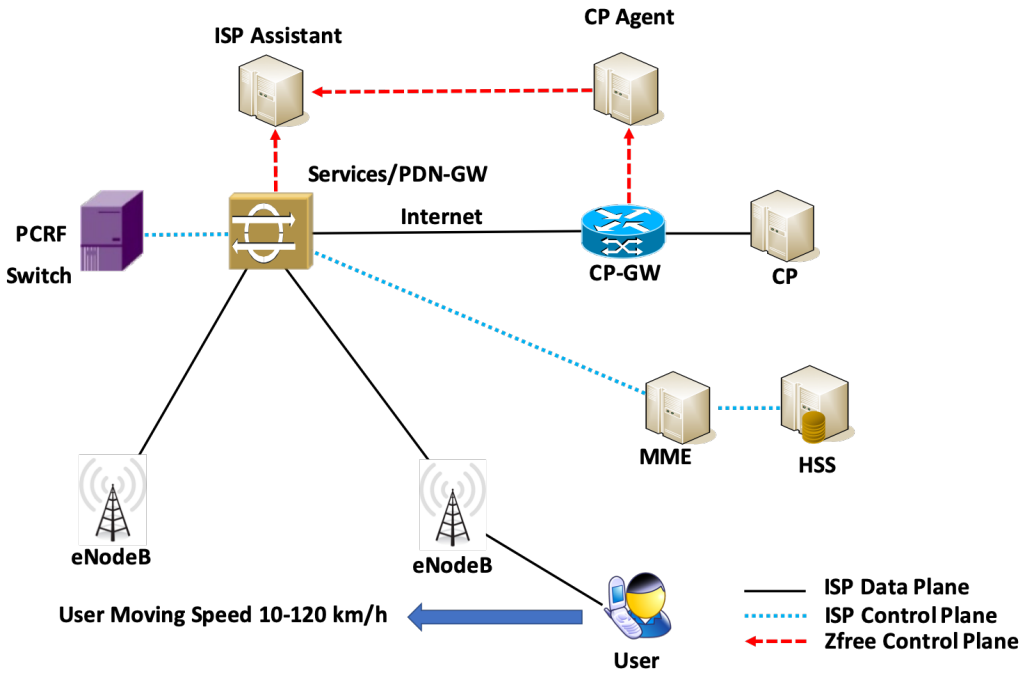


Figure 6.1: Cellular Network Testbed

as the user in the figure). Our ISP core network has a Serving/PDN gateway, an MME, an HSS, and a PCRF. The ISP core network is connected with two CP servers via a layer3 gateway router. We also simulate the user's mobility with speed from 10 km/h to 120 km/h between two eNodeB base stations.

WiFi Environment We establish airplane cabin WiFi environment based on Mininet-WiFi [4] shown in Figure 6.2. The environment is built on two physical machines with Intel i5-7400 CPU, 24GB memory and Ubuntu 16.04 LTS OS. The environment has 120 UEs and two 802.11n access point (AP) connected with one access controller (AC). The AC is connected to a CP server via a layer-three router. We also mimic the airplane cabin WiFi environment and limit the bandwidth between the APs and the AC to 30 Mbps. Our CP server is equipped with HTTP, HTTPs and iPerf stress testing service. We also simulate the user's mobility with speed from 5 km/h to 20 km/h between two APs.

We deploy ZFree upon these two environments: both the ISP Assistant and the Server Agent are hosted on Ubuntu 16.04 LTS virtual machines with 1.2GHZ CPU

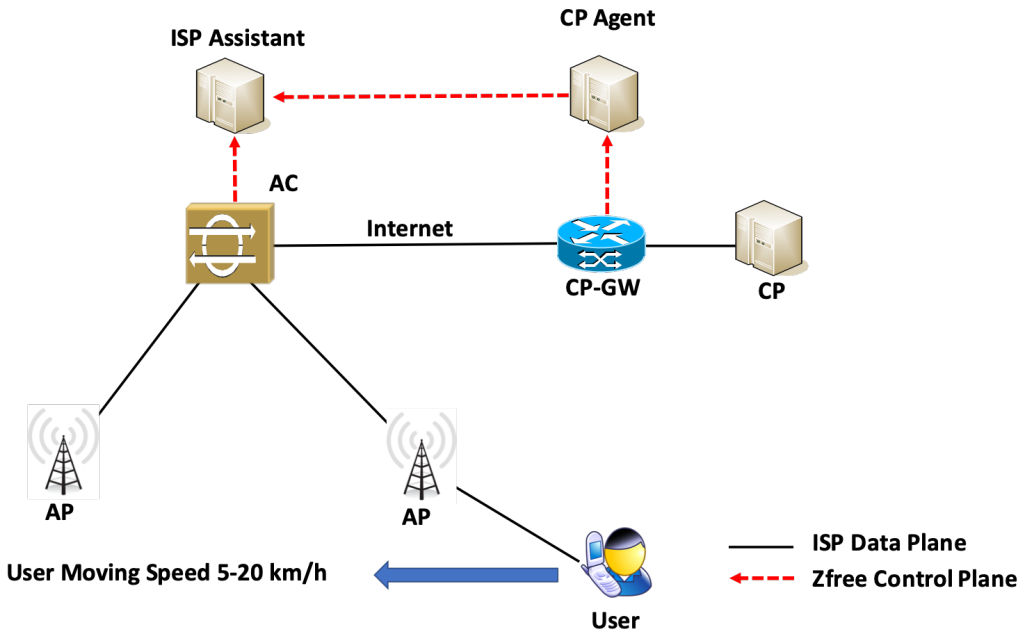


Figure 6.2: WiFi Network Testbed

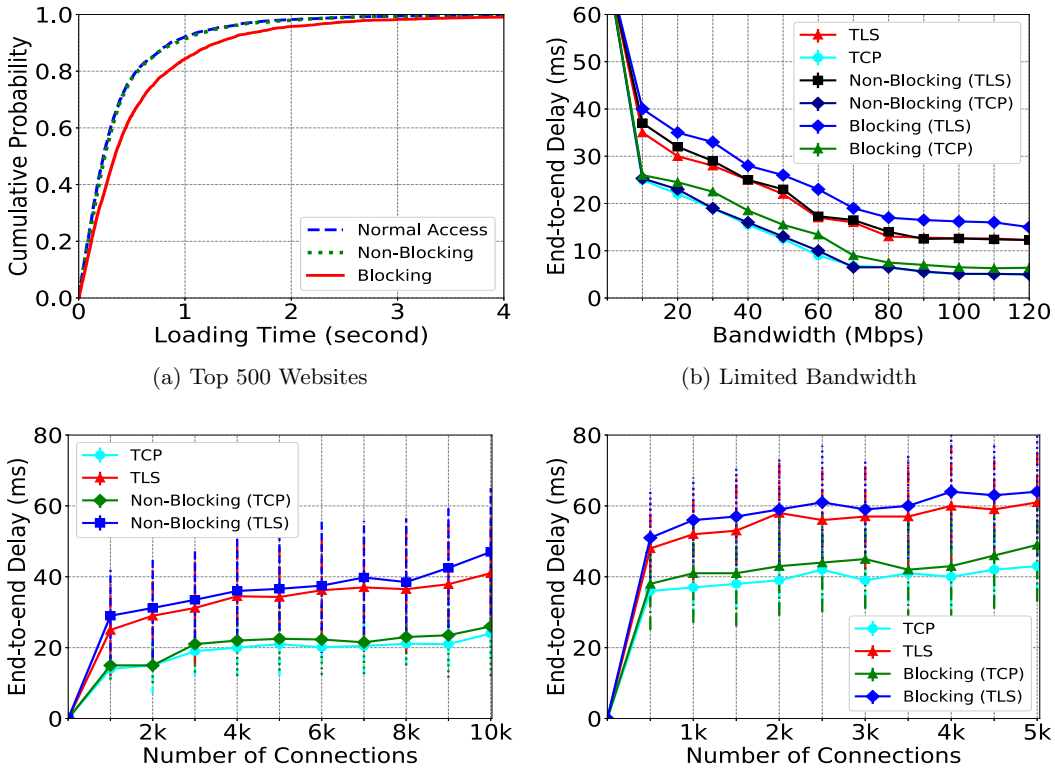
and 12 GB memory. They are connected with the corresponding network with a layer 3 OpenvSwitch (OVS) through NS3 real-time link model and Mininet-WiFi network bridge. The ISP Assistant and the Server Agent are connected via an OVS VxLAN based overlay network on top of the data plane.

6.2 Evaluation Results

In this section, we present the evaluation results on three aspects. First, we show the overhead, connection capacity, and download latency results of ZFree in end-to-end communication in Section 6.2.1. Next, we focus on evaluating ZFree's performance in ISP core network by launching the scalability test, street test, and control plane overhead test in Section 6.2.2. Final, we test the security of ZFree in Section 6.2.3.

6.2.1 End-to-end Performance

We measure the performance overhead of ZFree in end-to-end communication. We choose 4 factors to establish benchmark test, introduced as follows.



(a) Top 500 Websites

(b) Limited Bandwidth

(c) Scalability of Non-Blocking Mode

(d) Scalability of Blocking Mode

Figure 6.3: ZFree Evaluation Graphs: (a) The CDF of Loading Time of Top 500 Alexa Websites; (b) The End-to-end Delay vs. the Network Bandwidth; (c) The End-to-end Delay vs. the Number of Connections in Mobile Network Environment with ZFree's Non-blocking Mode; and (d) The End-to-end Delay vs. the Number of Connections in WiFi Environment with ZFree's Blocking Mode.

A. Page Loading Time

In this experiment, we measure the page loading time using the Top 500 Alexa Websites Benchmark. This benchmark is to test the loading time of 500 websites. We establish the experiments with and without ZFree to compare the overhead. Note that we count all the traffic as zero-rating for the measurement purpose.

Figure 6.3a shows the cumulative distribution function (CDF) graph of the loading time of this benchmark. The median overhead of ZFree's non-blocking mode is 1.26%, which mainly comes from hardware network port mirroring. The blocking mode of ZFree incurs 8.79% median overhead, which comes from the hash operations at both the ISP Assistant and the Server Agent.

B. End-to-end Delay with Different Bandwidth

In this experiment, we test the end-to-end delay in different bandwidth in the range from 0.1 Mbps to 120 Mbps. We test legacy TCP connection, legacy TLS connection, TCP connection with ZFree's non-blocking mode and TLS connection with ZFree's blocking mode.

Figure 6.3b shows the result. When the client downloads a 900MB video file, the end-to-end delay is shown in the y-axis and the network bandwidth in the x-axis. As expected, the end-to-end delay decreases as the network bandwidth increases. The result also shows that the delay of ZFree's non-blocking mode is almost the same as the native connection, such as TCP and TLS, and the delay of the blocking mode is constantly higher than the native connection.

C. LTE Handover Delay Testing

In this experiment, we test the handover delay in LTE. Specifically, we set up one UE to move from one eNodeB to another with traveling speed from 10km/h to 120km/h. We configure the transmission power of both eNodeBs as 46dBm and the handover algorithm as A2A4RSRQ [2,27]. Then, we adopt the iPerf stress test tool to keep the UE receiving data from our CP server. Figure 6.4 shows our LTE handover testing result. First, the result shows the transmission speed decreases as traveling speed increases. Second, the

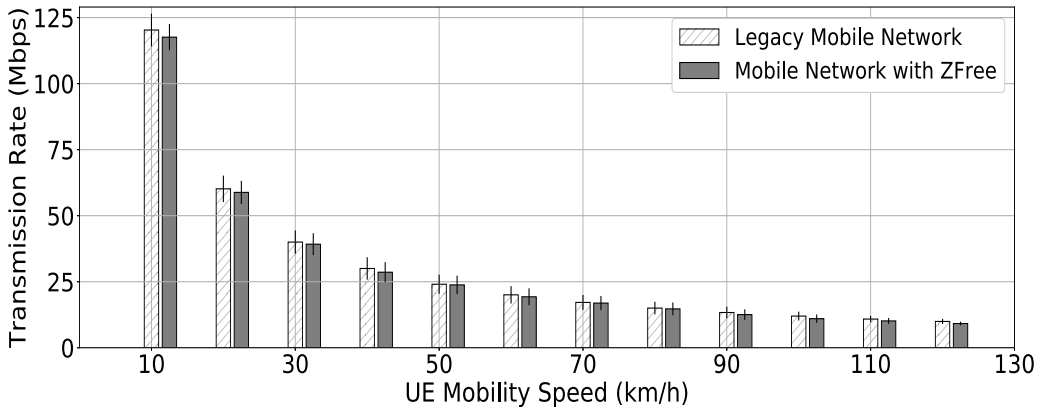


Figure 6.4: LTE Handover Throughput in both Legacy Mobile Network and Mobile Network with ZFree Enabled

result shows the transmission speed with ZFree enabled has 1.45% overhead than the one without ZFree. This means ZFree has little influence on the LTE handover.

6.2.2 ZFree in ISP Core Network

In this experiment, we measure ZFree performance inside the ISP core network by establishing scalability test and stress test.

A. Scalability Test

Scalability is essential to ISP core network. In this experiment, we measure whether ZFree can scale when the number of connections increases by measuring the setup end-to-end response time. In detail, we compare the timestamp in which the client sends a request, with the one in which the client receives the response. The experiment is performed using the non-blocking mode in the cellular network environment while using the blocking mode in WiFi environment. Figure 6.3c and 6.3d shows the end-to-end delay result where delay time of ZFree non-blocking and blocking modes is shown in the x-axis and the number of connections is showned in the y-axis. In both figures, we also display the end-to-end delays of the TCP and TLS connection without ZFree as a baseline for comparison. Our results show that the end-to-end delay hardly changes as the number of connections increases.

B. Stress Test

In this section, we perform the stress test on ZFree following RFC 2544 benchmark [7]. We set up one ISP server and two CP servers. The ISP server embeds the ISP Assistant while the CP servers embed the Server Agent in each. Specifically, we replay real-world traffic captured from netresec [5] and tcpReplay [12] in our testing environment. The Netresec network trace [5] has high-speed (8–10Gbps) network flows with 40 million packets from 1,982 applications, and the tcpReplay trace [12] low-speed (500Mbps) network flows with 791,615 packets from 132 applications. We replay the traffic using TCPreplay [9], a popular traffic replaying tool, for 5 hours. During the 5-hour period, the low-speed trace is repeated continuously from both CP servers to the UEs while the high-speed trace only from one CP server to the UEs every half an hour. The purpose is to simulate the bursty traffic scenario in the test.

Figure 6.5 shows the stress test result. The three figures show the speed and CPU usage in the ISP Assistant(Top figure) and two CP Agents (middle and bottom). During our replay, the legacy ISP network without ZFree has 9–10Gbps peak traffic with an average rate of 5.991Gbps in Figure 6.5 (top); the ISP network with ZFree also has 9–10Gbps peak traffic with a slightly lower average rate of 5.933Gbps. The CPU usage of the ISP Assistant is 70% during peak and 20% in normal case. Our first CP server (middle figure) has 1.582Gbp average traffic in the legacy network and 1.571Gbps average traffic in ZFree network. The average CPU usage in the first CP server is 15.4%. Our second CP server (bottom figure) has 4.332Gbps average traffic in the legacy network and 4.213Gbps with ZFree's network. The average CPU usage for the second CP server is 42.2%.

In sum, the evaluation results show that ZFree can support the needs for ISP core network with reasonable CPU overhead.

C. Control Plane Overhead Test

In this part, we measure two overheads: the control plane communication overhead and the control plane processing overhead. First, we replay a 900MB zero-rating video file from one CP to one UE. Next, we calculate the data volumes between the ISP

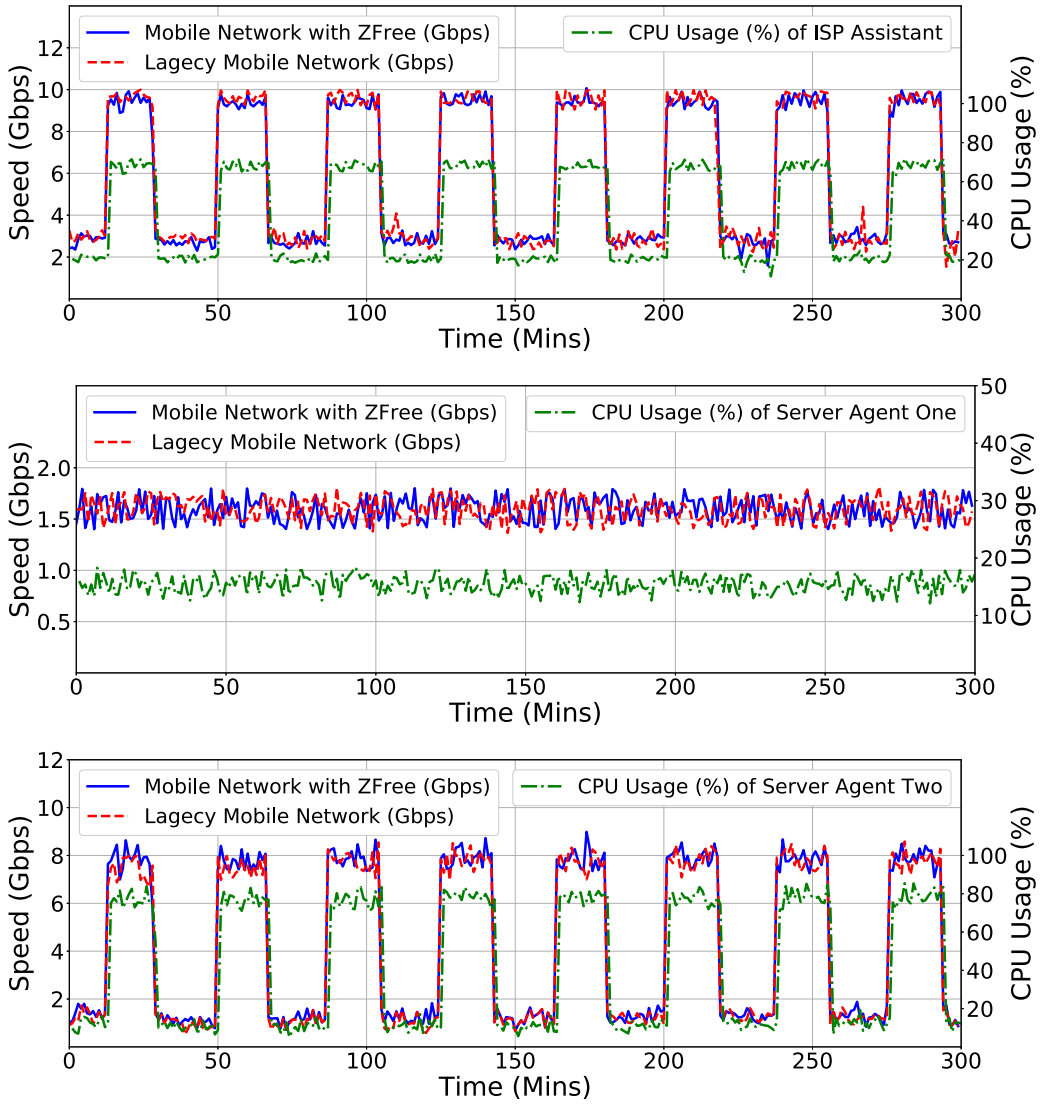


Figure 6.5: Network Traffic (Gbps) and CPU Usage (%) for the ISP Assistant (Top) and two CP Agents (Middle and Bottom) under Stress Test

Assistant and the Server Agent, thus comparing it with the total data plane traffic. Our evaluation shows that ZFree only introduces 4.2% control plane overhead.

Second, we compare ZFree control plane protocol with TLS hash function to transfer plain packet into hash code. Our evaluation shows that the TLS hash function incurs 2.8 times more overhead than the ZFree control plane protocol when processing 100MB data plane traffic.

6.2.3 Security

In this experiment, we evaluate the security of our ZFree implementation using three types of zero-rating attacks. Thre first two types of attack are documented in Chapter 2, which are masquerading CP server attack and Response Modification Attack. The third type is TCP retransmission-based free-riding attacks [20]. In this attack, we add two virtual switches, one between the ISP and the CP gateway, and the other between the client and the ISP. The first switch is used to modify response packets, e.g., encapsulating packets into TCP retransmission, and the second switch is used to recover the modified contents, e.g., stripping the added TCP headers. Our evaluation results show that ZFree is robust to all three types of free-riding attacks. Specifically, ZFree in its blocking mode rejects corresponding packets and the attacker cannot get the response.

Chapter 7

Discussions

In this chapter, we discuss several aspects of ZFree, namely ethics concerns, deployment issues, network neutrality, third-party contents, and Content Distribution Network (CDN).

First, we discuss the ethical concerns for the free-riding attacks that we launched against real-world ISPs. During all the experiments, we try to limit the damage that can occur to these ISPs. We only downloaded a small amount but enough data, so that the free-riding attack effect can be observed. The downloaded contents are hosted on our own server and contain no real information. Moreover, we paid these ISPs after all the experiments. For cellular networks, we paid the ISPs with extra data traffic fees for the amount that we used; for WiFi network, we purchased the WiFi, e.g., on United flight, after our experiment. We also try our best to inform the tested ISPs about the vulnerabilities we found. All the ISPs involved in our testing are informed of this vulnerability issue.

Second, we discuss the general issue about *network neutrality*. As mentioned by Yiakoumis et al. [51], some people raised concerns that zero-rating services could violate network neutrality. The general issue is orthogonal to our paper. The current status is that the Federal Communications Commission (FCC) determines whether a zero-rating service creates unfair conditions for consumers on a case-by-case basis. So far FCC approves most of existing ISP zero-rating services.

Third, we discuss how *third-party contents*, e.g., ads included in a webpage, are zero-

rated. The current ZFree prototype can only zero-rate first-party contents but not third-party. We note that this is a traditional hard problem in zero-rating framework and many real-world ISPs do not zero-rate third-party contents as well. For example, when we visit history.com, ISP only zero-rates contents from history.com but not the third-party ads embedded inside the webpage. We leave it as future work to include third-party contents.

Fourth, we discuss how to deal with CDN in ZFree. Each CDN server needs to install a Server Agent and communicate with the ISP Assistant. We realize that this is almost a non-issue in mobile network scenarios because many mobile ISPs host their own CDN and provide contents directly from their base stations. That is, the Server Agent and the ISP Assistant may be co-located in the same local network.

Chapter 8

Related Work

We discuss related work in this chapter.

8.1 Existing Attacks

We categorize existing attacks on ISP Policy and Charging Rules Function (PCRF) [8, 10] into two types, free-riding and overcharging.

First, an attacker as a malicious client can mislead ISP's PCRF and obtain access to illegitimate free data—defined as free-riding attacks. In the past, researchers show that an attacker may utilize different uncharged protocols, including TCP retransmission [20, 21], DNS [42] and ICMP [31], to launch free-riding attacks. The only three-party free-riding attack mentioned by Kakhki et al. [25] is to change the "Host" field of an HTTP packet to bypass charging. As a comparison, the measurement described in Section **??** studies the HTTPs protocol and also propose a new free-riding attack in which an attacker can modify the response from a zero-rating server and inject non-zero-rating contents.

Second, a man-in-the-middle attacker can generate huge amount of data between the client and the ISP to cause the users being charged for additional traffic, which is called overcharging attacks [21, 31, 43]. This type of attack is out of scope and one can refer to existing works [21, 31, 43] for solutions.

8.2 Existing Zero-rating Framework

In general, there are two types of zero-rating frameworks: ISP-only and ISP-CP approaches. First, many ISPs use traffic inspection techniques, such as Deep Packet Inspection (DPI) and its enhancement [33,48,52] to differentiate network traffic. Similarly, many other approaches [26, 28, 45, 46, 50, 53] can also be used to inspect network traffic. Although such approaches are effective in differentiating network traffic, especially on the protocol layer, they cannot be used to defend against our free-riding attacks. The reason is that the zero-rating contents in our scenario are generated by the CP and possibly encrypted, i.e., it is impossible and insecure for the CP to understand or inspect the traffic.

Second, people also propose to let the ISP and CP negotiate on a zero-rating policy. For example, Limited Use of Remote Keys (LURK) [35] and Session Protocol for User Datagrams (SPUD) [23] are two new protocols that allow middlebox to inspect end-to-end traffic. Yiakoumis et al. [51] propose a traffic authentication architecture so-called Network Cookie to provide on demand zero-rating services. Facebook Zero [3,11] allows CP to provide the ISP an IP Whitelist so that only traffic to an IP in the list is zero-rated. However, none of the aforementioned approaches can defend against free-riding attacks as they fail to authenticate zero-rating servers and verify packet integrity. Additionally, LURK and SPUD require the server codebase modifications, i.e., being incompatible with existing codebase.

8.3 Other Techniques

Packet hashing is also used by Chen et al. [18] for diagnosis purpose. Specifically, they use FPGA to compute all the packet hashes in the backbone network and deliver them to next hops for diagnosis. Note that packet hashing alone cannot defend against free-riding attacks, because ZFree needs to ensure both server authenticity and packet integrity. Middlebox enhancement include both blackbox and whitebox approaches. Blackbox enhancement [17, 22, 24, 29, 30, 39, 41, 44, 47] analyzes traffic without decryption or understanding the traffic. Such approach, though being effective in solv-

ing their own problem, cannot correctly zero-rate traffic without collaborating with the CP server. Whitebox approaches, such as mcTLS [38] and APIP [37], enhance TLS protocol to convey information for the middlebox. As a comparison, they require server code modifications and face backward compatibility problem in deployment. Certificate pinning [19, 40], or HTTP Public Key Pinning (HPKP), is a security mechanism embedded in HTTP header that defends against impersonation attack. Certificate pinning cannot prevent zero-rating attacks, because it requires the collaboration from the client.

Chapter 9

Conclusion

In conclusion, we first perform a measurement study of free-riding attacks against major real-world ISPs and research prototypes. The results show that all the existing ISPs and research prototypes are vulnerable, i.e., a malicious user can freely access any non-zero-rated websites without being charged.

To mitigate such free-riding attacks, we propose a secure and backward compatible zero-rating framework, called ZFree. ZFree authenticates and verifies all the communications between CP and user. ZFree is formally verified as a secure soluation against free-riding attacks. We implemented a prototype of ZFree and evaluated its performance on cellular network and WiFi network testbed. Our results show that ZFree is lightweight, secure, and scalable.

Bibliography

[1] (2017 global internet phenomena) spotlight: Zero-rating fraud. https://www.sandvine.com/downloads/general/global-internet-phenomena/2017/global-internet-phenomena-spotlight-zero-rating-fraud.pdf.

[2] Cisco:mobility management entity overview. https://www.cisco.com/c/en/us/td/docs/wireless/asr_5000/20/MME/b_20_MME_Admin/b_20_MME_Admin_chapter_01.pdf.

[3] The F5 handbook for service providers, page 31. https://f5.com/solutions/service-provider.

[4] Mininet-wifi: Emulator for wifi wireless networks. https://github.com/intrig-unicamp/mininet-wifi/wiki.

[5] netresec: Hands-on network forensics, training pcap dataset from first 2015. http://www.netresec.com/?page=PcapFiles.

[6] ns-3: discrete-event network simulator for internet systems. https://www.nsnam.org.

[7] [RFC 2544] benchmarking methodology for network interconnect devices. https://rfc-editor.org/rfc/rfc2544.txt, Mar 1999. RFC.

[8] 3gpp. ts32.240: Charging architecture and principles, 2003.

[9] Tcpreplay software. http://tcpreplay.appneta.com, Aug 2012. tcpreplay.

[10] 3gpp. ts 23.203: Policy and charging control architecture, 2013.

[11] Delivering zero-rated traffic, Dec. 2014. https://connect.limelight.com/blogs/limelight/2014/12/08/delivering-zero-rated-traffic.

[12] Sample captures for access network. http://tcpreplay.appneta.com/wiki/captures.html, May 2016. tcpreplay.

[13] Kenichi Arai. Formal verification of tls 1.3 full handshake protocol using proverif, 2015-2016. https://www.cellos-consortium.org/studygroup/TLS1.3-fullhandshake-draft11.pv.

[14] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[15] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, October 2016.

[16] Christopher E Caldwell and Janne P Linkola. System and method for authorizing access to an ip-based wireless telecommunications service, 2016. US Patent App. 15/396,192.

[17] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, pages 1–12, New York, NY, USA, 2007. ACM.

[18] Ang Chen, Andreas Haeberlen, Wenchao Zhou, and Boon Thau Loo. One primitive to diagnose them all: Architectural support for internet diagnostics. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, pages 374–388, New York, NY, USA, 2017. ACM.

[19] Sascha Fahl, Marian Harbach, Henning Perl, Markus Koetter, and Matthew Smith. Rethinking ssl development in an appified world. In *Proceedings of the 2013 ACM*

SIGSAC conference on Computer & communications security, pages 49–60. ACM, 2013.

[20] YH Go, Jongil Won, Denis Foo Kune, EunYoung Jeong, Yongdae Kim, and K Park. Gaining control of cellular traffic accounting by spurious tcp retransmission. In *Network and Distributed System Security (NDSS) Symposium 2014*, pages 1–15. Internet Society, 2014.

[21] Younghwan Go, Denis Foo Kune, Shinae Woo, KyoungSoo Park, and Yongdae Kim. Towards accurate accounting of cellular data for tcp retransmission. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13, pages 2:1–2:6, New York, NY, USA, 2013. ACM.

[22] Richard Gold, Per Gunningberg, and Christian Tschudin. A virtualized link layer with support for indirection. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, FDNA '04, pages 28–34, New York, NY, USA, 2004. ACM.

[23] Joe Hildebrand and Brian Trammell. Substrate Protocol for User Datagrams (SPUD) Prototype. Internet-Draft draft-hildebrand-spud-prototype-03, Internet Engineering Task Force, March 2015. Work in Progress.

[24] Dilip A. Joseph, Arsalan Tavakoli, and Ion Stoica. A policy-aware switching layer for data centers. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 51–62, New York, NY, USA, 2008. ACM.

[25] Arash Molavi Kakhki, Fangfan Li, David Choffnes, Ethan Katz-Bassett, and Alan Mislove. Bingeon under the microscope: Understanding t-mobiles zero-rating implementation. In *Proceedings of the 2016 Workshop on QoE-based Analysis and Management of Data Communication Networks*, Internet-QoE '16, pages 43–48, New York, NY, USA, 2016. ACM.

[26] Peyman Kazemian, Michael Chan, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In *NSDI*, pages 99–111, 2013.

[27] Furqan H Khan and Marius Portmann. A system-level architecture for software-defined lte networks. In *Signal Processing and Communication Systems (ICSPCS), 2016 10th International Conference on*, pages 1–10. IEEE, 2016.

[28] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P Godfrey. Veriflow: Verifying network-wide invariants in real time. *ACM SIGCOMM Computer Communication Review*, 42(4):467–472, 2012.

[29] Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. Embark: securely outsourcing middleboxes to the cloud. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 255–273. USENIX Association, 2016.

[30] Peter Lepeska. Trusted proxy and the cost of bits. In *90th IETF meeting*, 2014.

[31] Chi-Yu Li, Guan-Hua Tu, Chunyi Peng, Zengwen Yuan, Yuanjie Li, Songwu Lu, and Xinbing Wang. Insecurity of voice solution volte in lte mobile networks. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 316–327, New York, NY, USA, 2015. ACM.

[32] Chengchao Liang, F Richard Yu, and Xi Zhang. Information-centric network function virtualization over 5g mobile wireless networks. *IEEE network*, 29(3):68–74, 2015.

[33] Po-Ching Lin, Ying-Dar Lin, Yuan-Cheng Lai, and Tsern-Huei Lee. Using string matching for deep packet inspection. *Computer*, 41(4), 2008.

[34] Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. Ensuring connectivity via data plane mechanisms. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 113–126, Lombard, IL, 2013. USENIX.

[35] Daniel Migault. LURK Protocol for TLS/DTLS1.2 version 1.0. Internet-Draft draft-mglt-lurk-tls-01, Internet Engineering Task Force, March 2017. Work in Progress.

[36] Kanthi Nagaraj, Dinesh Bharadia, Hongzi Mao, Sandeep Chinchali, Mohammad Alizadeh, and Sachin Katti. Numfabric: Fast and flexible bandwidth allocation in datacenters. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 188–201. ACM, 2016.

[37] David Naylor, Matthew K Mukerjee, and Peter Steenkiste. Balancing accountability and privacy in the network. *ACM SIGCOMM Computer Communication Review*, 44(4):75–86, 2015.

[38] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-context tls (mctls): Enabling secure in-network functionality in tls. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 199–212, New York, NY, USA, 2015. ACM.

[39] Mark O'Neill, Scott Ruoti, Kent Seamons, and Daniel Zappala. Tls proxies: Friend or foe? In *Proceedings of the 2016 ACM on Internet Measurement Conference*, pages 551–557. ACM, 2016.

[40] J Osborne and A Diquet. When security gets in the way: Pentesting mobile apps that use certificate pinning. *Black Hat*, 2012.

[41] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Comput. Netw.*, 31(23-24):2435–2463, December 1999.

[42] Chunyi Peng, Chi-yu Li, Guan-Hua Tu, Songwu Lu, and Lixia Zhang. Mobile data charging: New attacks and countermeasures. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 195–204, New York, NY, USA, 2012. ACM.

[43] Chunyi Peng, Chi-Yu Li, Hongyi Wang, Guan-Hua Tu, and Songwu Lu. Real threats to your data bills: Security loopholes and defenses in mobile data charging. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 727–738, New York, NY, USA, 2014. ACM.

[44] Ashwin Rao, Justine Sherry, Arnaud Legout, Arvind Krishnamurthy, Walid Dabbous, and David Choffnes. Meddle: Middleboxes for increased transparency and control of mobile traffic. In *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, CoNEXT Student '12, pages 65–66, New York, NY, USA, 2012. ACM.

[45] Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Stefan Schmid, and Anja Feldmann. Opensdwn: Programmatic control over home and enterprise wifi. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 16:1–16:12, New York, NY, USA, 2015. ACM.

[46] Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Stefan Schmid, Anja Feldmann, and Roberto Riggio. Programming the home and enterprise wifi with opensdwn. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 117–118. ACM, 2015.

[47] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[48] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 213–226, New York, NY, USA, 2015. ACM.

[49] Elias Weingärtner, Florian Schmidt, Hendrik Vom Lehn, Tobias Heer, and Klaus Wehrle. Slicetime: a platform for scalable and accurate network emulation. In *Proceedings of NSDI'11: 8th USENIX Symposium on Networked Systems Design and Implementation*, page 253, 2011.

[50] Yang Wu, Ang Chen, Andreas Haeberlen, Boon Thau Loo, and Wenchao Zhou. Automated bug removal for software-defined networks. In *Proceedings of USENIX Symposium of Networked Systems Design and Implmentation (NSDI)*, March 2017.

[51] Yiannis Yiakoumis, Sachin Katti, and Nick McKeown. Neutral net neutrality. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 483–496, New York, NY, USA, 2016. ACM.

[52] Fang Yu, Zhifeng Chen, Yanlei Diao, TV Lakshman, and Randy H Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In *Architecture for Networking and Communications systems, 2006. ANCS 2006. ACM/IEEE Symposium on*, pages 93–102. IEEE, 2006.

[53] Ying Zhang, Zhuoqing Morley Mao, and Ming Zhang. Detecting traffic differentiation in backbone isps with netpolice. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, pages 103–115, New York, NY, USA, 2009. ACM.

Appendix:

Raw Trace Example Outputted by ProVerif

In the appendix, we show five examples of raw traces outputted by ProVerif. Figure 9.1 shows a counter example against response integrity for Network Cookies Model; Figure 9.2 shows another counter example trace against CP authenticity for IP Whitelist based zero-rating framework. Figure 9.3 shows a successful verification example of ZFree. Figure 9.4 shows a counter example trace if the ZFree packet hash is removed. Figure 9.5 shows a counter example trace if ZFree uses a weak hash function.

1. The attacker has some term cookie_attribute_1498. attacker(cookie_attribute_1498).
2. The attacker has some term cookie_key_1497.
attacker(cookie_key_1497).
3. The attacker has some term cookie_id_1496.
attacker(cookie_id_1496).
4. By 3, the attacker may know cookie_id_1496.
By 2, the attacker may know cookie_key_1497.
By 1, the attacker may know cookie_attribute_1498.
Using the function 3-tuple the attacker may obtain
Network_Cookie(cookie_id_1496,cookie_key_1497,cookie_attribute_1498).
attacker((cookie_id_1496,cookie_key_1497,cookie_attribute_1498)).
5. The attacker has some term transferred_server_certificate_1501.
attacker(transferred_server_certificate_1501).
6. We assume as hypothesis that attacker(response_data_1494).
7. By 6, the attacker may know response_data_1494.
By 5, the attacker may know transferred_server_certificate_1501.
Using the function 2-tuple the attacker may obtain (response_data_1494,transferred_server_certificate_1501).
attacker((response_data_1494,transferred_server_certificate_1501)).
8. The message (cookie_id_1496,cookie_key_1497,cookie_attribute_1498) that the attacker may have by 4 may be received at input {14}.
The message (response_data_1494,transferred_server_certificate_1501) that the attacker may have by 7 may be received at input {18}.
So event endResponseVerif(response_data_1494) may be executed at {19}.
end(endResponseVerif(response_data_1494)).

Figure 9.1: Counter example traces on verifying response integrity for Network Cookies (TCP Connection)

1. The attacker has some term response_Sequence_Number_136.
attacker(response_Sequence_Number_136).
2. The attacker has some term response_ACK_Number_135.
attacker(response_ACK_Number_135).
3. The attacker has some term response_Port_Number_134.
attacker(response_Port_Number_134).
4. The attacker has some term response_IP_133.
attacker(response_IP_133).
5. By 4, the attacker may know response_IP_133.
By 3, the attacker may know response_Port_Number_134.
By 2, the attacker may know response_ACK_Number_135.
By 1, the attacker may know response_Sequence_Number_136.
Using the function 4-tuple the attacker may obtain
Server_response(response_IP_133,response_Port_Number_134,
response_ACK_Number_135,response_Sequence_Number_136).
attacker((response_IP_133,response_Port_Number_134,response
_ACK_Number_135,response_Sequence_Number_136)).
6. We assume as hypothesis that attacker(server_identity_150).
7. The message response_Sequence_Number_136 that the attacker may have by 1 may be received at input 24.
The message (response_IP_133,response_Port_Number_134,response_ACK_Number_135,
response_Sequence_Number_136) at 26 in copy server_identity_150.
The message (response_IP_133,response_Port_Number_134,response_ACK_Number_135,
response_Sequence_Number_136) that the attacker may have by 6 may be received at input 27.
So event endIPVerify(server_identity_150) may be executed at 28 in session cid_181.
A trace has been found.
RESULT inj-event(endIPVerify(server_identity)) == inj-event(endIPVerify(server_identity)) is false.
RESULT (even event(endIPVerify(server_identity_150)) == event(endIPVerify(server_identity_150)) is false.)

Figure 9.2: Counter example traces on verifying CP authenticity for IP Whitelist based zero-rating framework (TLS Connection)

1. Starting query event(endResponseVerif_h(keyedhash)) == event(beginResponseVerif_h(keyedhash)) RESULT
event(endResponseVerif_h(keyedhash)) == event(beginResponseVerif_h(keyedhash)) is true.
2. Starting query event(endResponseVerif_d(response_data1,response_data2,
response_data3,response_data4)) == event(beginResponseVerif_d(response_data1,
response_data2,response_data3,response_data4)) RESULT
event(endResponseVerif_d(response_data1,response_data2,response_data3,
response_data4)) == event(beginResponseVerif_d(response_data1,response_data2,
response_data3,response_data4)) is true.
3. Starting query event(endintegrityVerif_c(response_data)) == event(begintegrityVerif_c(response_data))
RESULT event(endintegrityVerif_c(response_data)) == event(begintegrityVerif_c(response_data)) is true.
4. Starting query inj-event(endClient(s,t,u,v_2565544,w)) == inj-event(beginClient(s,t,u,v_2565544,w))
RESULT inj-event(endClient(s,t,u,v_2565544,w)) == inj-event(beginClient(s,t,u,v_2565544,w)) is true.
5. Starting query inj-event(endServerVerif(server_identity)) == inj-event(beginServerVerif(server_identity))
RESULT inj-event(endServerVerif(server_identity)) == inj-event(beginServerVerif(server_identity)) is true.
6. Starting query not attacker(data_c) RESULT not attacker(data_c) is true.

Figure 9.3: example traces on verifying ZFree

goal reachable: attacker(response_data4_759694) && attacker(response_data3_759695) &&
attacker(response_data2_759696) && attacker(response_data1_759697) -
end(endResponseVerif_d(response_data1_759697,response_data2_759696,
response_data3_759695,response_data4_759694))
1. We assume as hypothesis that attacker(response_data1_759707).
2. We assume as hypothesis that attacker(response_data2_759708).
3. We assume as hypothesis that attacker(response_data3_759709).
4. We assume as hypothesis that attacker(response_data4_759710).
5. The message response_data1_759707 that the attacker may have by 1 may be received at input 178. The message response_data2_759708 that the attacker may have by 2 may be received at input 179. The message response_data3_759709 that the attacker may have by 3 may be received at input 180. The message response_data4_759710 that the attacker may have by 4 may be received at input 181. So event endResponseVerif_d(response_data1_759707,response_data2_759708,response_data3_759709, response_data4_759710) may be executed at 182.
end(endResponseVerif_d(response_data1_759707,response_data2_759708,
response_data3_759709, response_data4_759710)).
A more detailed output of the traces is available with set traceDisplay = long.
new skCA creating skCA_759715 at 1
out(c, pk(skCA_759715)) at 3
new skS creating skS_759879 at 4
out(c, (HostInfoCA,HostInfoS,pk(skS_759879),
sign(H((HostInfoCA,HostInfoS,pk(skS_759879))),skCA_759715))) at 8
in(d, a) at 178 in copy a_759714
in(d, m1) at 179 in copy a_759714
in(d, a_759712) at 180 in copy a_759714
in(d, a_759713) at 181 in copy a_759714
event(endResponseVerif_d(a,a_759711,a_759712,a_759713)) at 182 in copy a_759714
The event endResponseVerif_d(a,a_759711,a_759712,a_759713) is executed. A trace has been found.
RESULT event(endResponseVerif_d(response_data1,response_data2,response_data3,
response_data4)) == event(beginResponseVerif_d(response_data1,
response_data2, response_data3,response_data4)) is false.

Figure 9.4: example traces on modifying ZFree packet key hash

goal reachable: attacker(response_data_1521060) - end(endintegrityVerif_c(response_data_1521060))
1. Using the function server_id the attacker may obtain server_id.
attacker(server_id).
2. The attacker has some term server_cipher_suite_1521414.
attacker(server_cipher_suite_1521414).
3. The attacker has some term server_version_1521412.
attacker(server_version_1521412).
4. By 3, the attacker may know server_version_1521412.
By 2, the attacker may know server_cipher_suite_1521414.
By 1, the attacker may know server_id.
Using the function 3-tuple the attacker may obtain (server_version_1521412,
server_cipher_suite_1521414,server_id).
attacker((server_version_1521412,server_cipher_suite_1521414,server_id)).
5. By 3, the attacker may know server_version_1521412.
By 2, the attacker may know server_cipher_suite_1521414.
Using the function 2-tuple the attacker may obtain (server_version_1521412,
server_cipher_suite_1521414).
attacker((server_version_1521412,server_cipher_suite_1521414)).
6. The message (server_version_1521412,server_cipher_suite_1521414,
server_id) that the attacker may have by 4 may be received at input 10.
So the message (server_version_1521412,client,client_legacy_session,
server_cipher_suite_1521414,server_id,exp(g,X_1521421)) may
be sent to the attacker at output 16.
attacker((server_version_1521412,client,client_legacy_session,
server_cipher_suite_1521414,server_id,exp(g,X_1521421))).
7. By 6, the attacker may know (server_version_1521412,
client,client_legacy_session,server_cipher_suite_1521414,
server_id,exp(g,X_1521421)).
Using the function 6-proj-6-tuple the attacker may obtain
exp(g,X_1521421).
attacker(exp(g,X_1521421)).
8. By 6, the attacker may know (server_version_1521412,client,
client_legacy_session,server_cipher_suite_1521414,server_id,
exp(g,X_1521421)).
Using the function 3-proj-6-tuple the attacker may obtain
client_legacy_session.
attacker(client_legacy_session).
9. By 6, the attacker may know (server_version_1521412,
client,client_legacy_session,server_cipher_suite_1521414,
server_id,exp(g,X_1521421)). Using the function 2-proj-6-tuple the attacker may obtain client.
attacker(client).
10. By 3, the attacker may know server_version_1521412.
By 9, the attacker may know client.
By 8, the attacker may know client_legacy_session.
By 2, the attacker may know server_cipher_suite_1521414.
By 1, the attacker may know server_id.
By 7, the attacker may know exp(g,X_1521421).
Using the function 6-tuple the attacker may obtain (server_
version_1521412,client,client_legacy_session,server_cipher
_suite_1521414,server_id,exp(g,X_1521421)).
attacker((server_version_1521412,client,client_legacy_sess
ion,server_cipher_suite_1521414,server_id,exp(g,X_1521421))).
11. The message (server_version_1521412,server_cipher_suite
_1521414) that the attacker may have by 5 may be received at input 91.
33. By 32, the attacker may know (server_version_1521412,
server_random_1521422,server_cipher_suite_1521414,exp(g,Y_1521423)).
Using the function 4-proj-4-tuple the attacker may obtain exp(g,Y_1521423).
attacker(exp(g,Y_1521423)).
34. By 32, the attacker may know (server_version_1521412,
server_random_1521422,server_cipher_suite_1521414,exp(g,Y_1521423)).
Using the function 2-proj-4-tuple the attacker may obtain server_random_1521422.
attacker(server_random_1521422).
35. By 3, the attacker may know server_version_1521412.
By 34, the attacker may know server_random_1521422.
By 2, the attacker may know server_cipher_suite_1521414.
By 33, the attacker may know exp(g,Y_1521423).
Using the function 4-tuple the attacker may obtain (server_version_1521412,
server_random_1521422,server_cipher_suite_1521414,exp(g,Y_1521423)).
attacker((server_version_1521412,server_random_1521422,
server_cipher_suite_1521414,exp(g,Y_1521423))).
event(endintegrityVerif_c(a_1521424)) at 83 in copy a_1521437
The event endintegrityVerif_c(a_1521424) is executed.
A trace has been found.
RESULT event(endintegrityVerif_c(response_data)) ==> event(begintegrityVerif_c(response_data)) is false.

Figure 9.5: example traces on modifying ZFree using weak hash function

Vita

Zhiheng Liu was born on November 9th, 1987 in Beijing, China. He studied at Beijing University of Technology in 2006 and received his Bachelor of Science Degree in Software Engineering in 2010. After graduation, Zhiheng decided to explore in Japan. He joined Waseda University to pursue his Master of Science in Information Telecommunication degree under the supervision of Prof. Takuro Sato. Zhiheng graduated in September 2010 and later joined China Mobile Research Institute in Beijing, China. He worked there as a research engineer for four years where he engaged himself in several projects related to cloud technology. In 2016, Zhiheng joined Lehigh University in the United States to pursue his Master of Science in Computer Science degree under the supervision of Prof. Yinzhi Cao. Zhiheng plans to graduate from Lehigh University in December of 2018.