Towards a Secure Zero-rating Framework with Three Parties

Zhiheng Liu, Zhen Zhang, Yinzhi Cao[†], Zhaohan Xi, Shihao Jing, and Humberto La Roche[‡]
{zhl416, zhza16, yinzhi.cao, zhx516, shj316}@lehigh.edu, hlaroche@cisco.com
Lehigh University, [†] The Johns Hopkins University/Lehigh University, [‡] Cisco Systems

Abstract

Zero-rating services provide users with free access to contracted or affiliated Content Providers (CPs), but also incur new types of free-riding attacks. Specifically, a malicious user can masquerade a zero-rating CP or alter an existing zero-rating communication to evade charges enforced by the Internet Service Provider (ISP). According to our study, major commercial ISPs, such as T-Mobile, China Mobile, Boingo airport WiFi and United cabin WiFi, are all vulnerable to such free-riding attacks.

In this paper, we propose a secure, backward compatible, zero-rating framework, called ZFree, which only allows network traffic authorized by the correct CP to be zero-rated. We perform a formal security analysis using ProVerif, and the results show that ZFree is secure, i.e., preserving both packet integrity and CP server authenticity.

We have implemented an open-source prototype of ZFree available at this repository (https://github.com/zfree2018/ZFREE). A working demo is at this link (http://zfree.org/). Our evaluation shows that ZFree is lightweight, scalable and secure.

1 Introduction

Internet service providers (ISPs) often provide so-called zero-rating services, in addition to the normal charged ones, for contracted or affiliated content providers (CPs) to either attract more users or shift the payment responsibility from users to corresponding CPs. For example, T-Mobile provides a program called BingeOn with over one hundred CPs, such as Youtube, so that T-Mobile users can access free services provided by these CPs, e.g., watching Youtube videos. United Airline also allows passengers to access United.com and its partners' websites without paying fees over cabin WiFi.

Although zero-rating services provide convenience for both users and CPs, attackers—i.e., malicious users in our threat model—can launch so-called free-riding attacks to bypass the pre-set zero-rating policies and visit normal websites beyond zero-rating services for free. Such free-riding attacks involve three parties, i.e., the user, the ISP, and the CP. The threat model is different from traditional two-party attacks which exploit ISP-side charging bugs via uncharged protocols such as network domain service (DNS) and TCP retransmission. Specifically, Kakhki et al. [25] show that an attacker can masquerade a non-zero-rating HTTP server to be BingeOn enabled, i.e., zero-rated. One recent report from Sandvine [1] concludes, based on manual analysis of the small amount of real-world HTTP traffic, that a major US network carrier could lose $7,000,000 in a month due to such free-riding attacks alone. Our own manual analysis, as stated in Section 2.3, also reveals that in just one province of China, China Mobile loses at least half a million US dollar per month for 71TB free-riding traffic due to such attacks.

To better understand such free-riding attacks, we need to describe how existing zero-rating framework adopted by ISPs differentiates charged and zero-rating traffic. The tactic widely adopted in real-world ISPs is to directly inspect the traffic based on meta-data thus differentiating zero-rating contents. However, because a zero-rating policy involves three parties, the ISP can never tell whether the contents are indeed authorized by the CP as zero-rated, especially under the condition that one of the communicating party, i.e., the client, is malicious. Specifically, according to the nature of end-to-end communication, the client has the ability to modify or inject any non-zero-rated contents in between the ISP and the CP, even if the communication is encrypted.

To demonstrate this point, in Section 2.2 we go beyond the attacks proposed by Kakhki et al. [25] by introducing two new types of free-riding attacks: one compromising the end-to-end communication integrity and the other masquerading a HTTPS server. Our results show that many major real-world ISPs, such as T-Mobile LTE and Boingo Airport WiFi, are vulnerable to these two types of free-riding attacks. That is, even if the ISP

fixes the vulnerability proposed by Kakhki et al. [25] by authenticating the CP, free-riding attacks still exist.

As ISPs cannot differentiate zero-rating contents without the involvement of CPs, several recently-proposed zero-rating frameworks also include CPs in the process. In fact, many CPs, such as Facebook [11], also express their interest in a zero-rating framework, because free-riding traffic is eventually being charged to the CPs in terms of a payment responsibility shift. In some other cases such as United cabin WiFi and China Mobile's Migu video, the CPs are controlled by the ISPs, i.e., they are automatically involved.

Although theoretically it is possible to build a secure, zero-rating framework with both the CP and the ISP are involved, such effort is not straightforward. In fact, existing frameworks—no matter from academia or industry—are both vulnerable to free-riding attacks according to our analysis and experiments in Section 2.2.3. For example, Yiakoumis et al. proposed network cookies [50] in which an authentication token (called network cookie) serves as a ticket for the ISP to zero-rate corresponding traffic. We show that an attacker can either bind a network cookie designated for zero-rating traffic to normal traffic or inject non-zero-rating data into zero-rating traffic to bypass the zero-rating policy. For another example, Facebook provides an IP whitelist-based framework, called Facebook Zero [11], which allows ISPs to obtain an IP list for authentication. We also show that such approach is vulnerable, because a malicious user with the knowledge of all the communication information, such as TCP sequence number, can easily camouflage TCP/IP packets. To summarize, none of existing frameworks realize that the free-riding adversary, having access to all the end-to-end communication information, is different from a traditional network attacker, such as a man-in-the-middle—therefore, they cannot defend against free-riding attacks.

In this paper, we propose a brand-new Zero-rating FRamework with thrEe partiEs (ZFree) to defend against the powerful free-riding adversary. The key insight of ZFree is that the ISP and the CP need to exchange authentication information of the CP-user communication exclusively from the user. There are two points worth noting. First, the information should be kept from the user, a potential free-riding attacker. That is why some existing work, like Network Cookies [50] making the cookie information available to the user, fail to defend against free-riding attacks. Second, the information should be able to authenticate the communication between the user and the CP. Therefore, an IP whitelist-based approach, which adopts IP, a piece of forgeable information, cannot defend against free-riding attacks.

While the insight of ZFree is intuitively simple, the challenges lie in that the ZFree's design needs to satisfy the following properties:

- Security. ZFree needs to validate the authenticity of the zero-rating CP and verify the integrity of the communication between the CP and the user.
- Backward Compatibility. ZFree needs to incur minimum deployment burden to both CPs and ISPs, including no changes to existing (*i*) codebase and (*ii*) network packets. Specifically, any such changes may break existing network functionalities, such as intrusion detection systems and loader balancers.
- Privacy. ZFree needs to preserve the communication privacy between the user and the CP. That is, the CP cannot directly reveal any communication contents to the ISP for authentication.
- Performance. The performance overhead added to the end-to-end communication needs to be minimum. For example, if an unencrypted communication is sufficient between the CP and the user, we do not want to encrypt the communication for authentication, which brings overhead.

Specifically, we design a secure protocol, called ZFree control plane protocol, which transfers keyed hash, such as hash-based message authentication code (HMAC), of the CP-user communication (i.e., defined as data plane). Our protocol is simple and minimize—the protocol only needs to preserve server authenticity and data integrity for both control and data planes but not necessarily data secrecy like TLS. In particular, we make the following contributions in design ZFree control plane protocol to meet all the four properties as mentioned above. .

- 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, i.e., robust to free-riding attacks and we also discover that such protocol design is subtle because a simple variation can lead to a vulnerable protocol.
- Deploying pluggable components at both the ISP and the CP. To ease the deployment burden and maintain backward compatibility, we deploy a so-called *server agent* at the gateway of the CP that sniffs the traffic, hashes necessary packets and sends secure hashes to the ISP for authorization purpose. Meanwhile, we deploy a so-called *ISP assistant* at the ISP's core network that also sniffs the traffic, hashes packets and communicates with the server agent.
- Verifying packet integrity without violating end-to-end privacy. The ISP assistant verifies packet integrity by checking the secure hashes sent from the server agent: Only when the ISP assistant finds a match, the corresponding packet will be authorized for zero-rating service. That is, ZFree does not need

to understand the application layer protocols, thus preserving end-to-end privacy.

- Matching hash values in a distributed manner. The ISP assistant matches hashes received from the server agent by parallelizing the task to distributed nodes based on the prefixes of the hash values. Our evaluation shows that the non-blocking mode of ZFree—a mode used in mobile network as users can pay bills afterward—incurs only 1.26% overhead on the loading time of Top 500 Alexa websites and the blocking mode—a mode used in WiFi network—incurs 8.79% overhead. Our evaluation also shows both non-blocking and blocking modes introduce less network latency than TLS encryption.

We implemented an open-source prototype version of ZFree at the following repository (https://github.com/zfree2018/ZFREE) as well as a demo website (http://zfree.org/).

2 Free-riding Attacks

We first describe the threat model by presenting the roles of three parties in Section 2.1. Then, in Section 2.2, we present how to launch free-riding attacks on a broad range of real-world ISPs and research prototypes. Lastly, in Section 2.3, we introduce a manual analysis of free-riding attacks in China Mobile, a major ISP in China.

2.1 Threat Model

Our threat model has three parties, i.e., the user, the ISP and, the CP, as described below.

- User. A user visits the Internet under the service provided by the ISP via a *client* in terms of User Equipment (UE), e.g., mobile phone, in the mobile network. Normal traffic from the user is charged, and a small portion is zero-rated under the policy between the ISP and the CP. Our threat model assumes that the user is potentially *malicious*, i.e., trying to bypass the charging policy enforced by the ISP.
- Internet Service Provider (ISP). An ISP provides Internet service to the user. Our threat model assumes that the ISP is *benign*, i.e., trying to protect itself from free-riding attacks launched by users. Note that we exclude a malicious ISP because such scenario will fall back to the traditional end-to-end connection problem where the ISP is the man-in-the-middle.
- Content Provider (CP). A CP provides abundant contents, e.g., multimedia and games, to users. Our threat model assumes that the CP is *benign*, although a user may masquerade zero-rating CPs to mislead ISP.

2.2 Case Studies on Free-riding Attacks against real-world ISPs and Research Prototypes

In this section, we describe how to launch free-riding attacks against ISPs, such as real-world mobile networks, WiFi networks, and research prototypes.

2.2.1 Real-world Mobile Networks

Real-world mobile ISPs adopt different tactics to zero-rate unencrypted (HTTP) or encrypted (HTTPS) traffic. Specifically, mobile ISPs adopt Deep Packet Inspection (DPI) to inspect the Host field of the HTTP header and determine whether the field belongs to a zero-rating CP. As for HTTPS traffic, mobile ISPs extract the destination host name from the Server Name Indication (SNI) in Server Name Extension segment of the client hello message and uses it as the determining factor of the zero-rating policy.

Due to the simple inspection tactics, an attacker can launch two types of free-riding attacks as follows. First, the attacker can masquerade a zero-rating traffic by modifying either the Host or SNI field in the HTTP(S) request packet. Second, the attacker can create a proxy between the ISP and a zero-rating CP, which modifies the CP's response. Such response modification is intuitive for unencrypted traffic; as for attacking encrypted traffic, because the client is malicious, the client can decrypt the content using the session key, modify packet, and then encrypt it again.

Now let us look at how these two types of free-riding attacks work for real-world ISPs. Particularly, we tested three zero-rating programs of different real-world ISPs, i.e., the BingeOn program of T-Mobile, the Migu video service of China Mobile, and the 'Wo+Tencent' video streaming service of China Unicom. In each case, we use the volume of charged data to verify whether the attack succeeds. Table 1 shows the overall results: except for these cases when the corresponding service is unavailable, all zero-rating programs of real-world ISPs are vulnerable to both types of free-riding attacks.

2.2.2 Real-world WiFi Networks

There is no official documentation about how real-world WiFi networks zero-rate traffic. According to our analysis, the tactics are similar to mobile networks and we can launch the same free-riding attacks as in mobile networks. Specifically, we tested two types of free WiFi networks, i.e., United airline cabin WiFi and Boingo WiFi in Chicago O'Hare International Airport. United airline provides free WiFi network when users visit certain partners' websites, such as united.com and hertz.com. Boingo in Chicago O'Hare international airport provides a free WiFi network for 30 minutes and then charges the users.

Table 1 shows that both WiFi networks are vulnerable to free-riding attacks when the corresponding service is available. There are two things worth noting. First, we test the United cabin WiFi networks on a United flight from Newark Liberty International Airport, NJ to Miami International Airport, FL in December 2016. On that specific flight, United WiFi only allows users to

Table 1: Summary of the attacks on various defenses, such as these deployed on real-world ISPs and prototypes.

		Mobile Network			WiFi Network		Prototypes	
		T-Mobile	China Mobile	China Unicom	United	ORD	Network Cookies	IP Whitelist
Unencrypted traffic	Request masquerade	✗	✗	N/A	✗	✗	✗	✗
	Response modification	✗	✗	N/A	✗	✗	✗	✗
Encrypted traffic	Request 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.

visit HTTP version of united.com and hertz.com but not HTTPS version. Because all the HTTPS traffic is blocked by default when the user does not pay for the Internet, an attacker cannot masquerade HTTPS traffic. Second, we launch the free-riding attacks against the boingo WiFi in the ORD airport after the 30-minute free trial expires.

2.2.3 Research Prototypes

In this part, we launch free-riding attacks against research prototypes that receive information from CPs for authentication. Specifically, we tested two prototypes: Network Cookies [50] , a zero-rating framework utilizing cookie-like tokens for authentication, and IP whitelist, which authenticates traffic based on a preset whitelist of the CP's IP addresses.

Network Cookies We first 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 a zero-rating CP and a user can be compromised by a man-in-the-middle attacker as the cookie does not validate the contents conveyed in the communication. We show that both of the implementation and protocol design in Network Cookies is vulnerable to free-riding attacks. Details about the vulnerability in their protocol can be found in Section 6. We now discuss their implementation. Specifically, we obtain the original implementation from the authors of Network Cookies paper and deploy the implementation in our lab environment. Network Cookies client, ISP middlebox and cookie server are installed at three lab servers with Ubuntu 16.04 operating systems: The client asks for Network Cookies together with DNS requests and the ISP middlebox verifies Network Cookies received from the client via a *verifycookie* function. We also setup a CP server, i.e., a NGINX web server, as the zero-rating content provider, and configure the hostname of the CP server to be zero-rated in the cookie server.

We then perform the aforementioned free-riding attacks and show that the prototype is vulnerable in Table 1. First, we create a malicious client application that binds the zero-rating network cookie obtained from the cookie server to a non-zero-rating traffic, i.e., attaching a valid network cookie in the HTTP header field 'network-cookie' with a non-zero-rating hostname. The results show that the ISP marks the traffic as zero-rated, thus exposing the vulnerability to free-riding attacks. Second, we create a proxy between the ISP and the CP server to modify the HTTP traffic. The results show that the proxy can successfully inject any arbitrary contents into a zero-rated traffic.

IP Whitelist We then launch free-riding attacks against a zero-rating framework based on IP whitelist. Specifically, here is how we setup the testing environment. We establish a CP server in a campus network and then a client in DigitalOcean Cloud. Then, we setup an IP whitelist server in between the client and the CP server that only allows zero-rating packet to be forwared. Now let us explain how we launch these two types of free-riding attacks.

First, we setup a masqueraded CP server in a different campus network, which pretends to be the zero-rating CP server. Then, the client—which is cooperating with the masqueraded server—establishes a connection, either encrypted or unencrypted, with the real CP server. Once the connection is created, the client forwards all the connection information, such as the sequence number, the acknowledgement number, the destination port, the source port and the TCP flags, to the masqueraded CP server. The masqueraded server, based on the received information, crafts TCP packets with zero-rating header mimicking the real CP server's behavior and send it to the client. As shown in Table 1, we can successfully launch these free-riding attacks against an IP whitelist based zero-rating framework. Our experiment results further show that we can launch such free-riding attack with only small amount of charged traffic, i.e., the information about the TCP connection to the real CP server. Specifically, the attack only requires 386 bytes for such information to the masqueraded server and the rest will be all free-riding traffic. Note that the masqueraded server needs information about the TCP connection to the real CP server because the ISP may have a firewall that checks all the connections and blocks malformed ones. Another thing worth noting is that the masqueraded server can embed free-riding traffic in TCP retransmission packets so that even

if the ISP checks the traffic volume, it cannot notice the difference.

Second, we setup a proxy in between the real CP server and the client to inject or modify the contents and the results prove the feasibility. Interestingly, in our prior experiment about masqueraded CP, the packet integrity between the client and the real CP is also violated, because the client can directly receive the crafted packet from the masqueraded CP if we use the next sequence number of the client-CP communication in the crafted packet.

2.3 Manual Analysis of Free-riding Attacks in China Mobile

In this section, we measure the severeness of free-riding attacks from an ISP's perspective. Specifically, we try to estimate the amount of free-riding traffic in China mobile's network. We understand that this is a generally difficult task, because if we can accurately measure free-riding attacks, such approach can be used for detection as well. In this subsection we gauge a lower bound for the amount of free-riding traffic.

The detailed steps for calculation is as follows. First, we calculate the average amount of zero-rated data for a normal user, which is roughly 300MB/month. Second, we filter these users whose zero-rating traffic amount is significantly higher than that of a normal user, say 3GB/month, from China mobile's billing system. Lastly, we manually inspect the zero-rating traffic of such users, e.g., looking at the communication contents if unencrypted, to decide whether it is free-riding traffic.

Our manual analysis is performed on the billing system of China Mobile's network in one province in January 2016. The results reveal 71TB free-riding traffic, equaling to half a million US dollar based on the China Mobile data charging rate. Note that one interesting finding is that some users consumed more than 30GB zero-rating data with Migu music per month, which is technically impossible for that zero-rating service because the user stream music for more than 24 hours per day.

3 Overview

In this section, we describe ZFree's architecture using mobile network as a deployment example shown in Figure 1. ZFree has two pluggable components: ISP assistant and CP server agent. The ISP assistant, located in the ISP's core network, is responsible for interacting with the server agent from different CPs, authenticating CPs and verifying zero-rating traffics with the information obtained from the server agent. The server agent, located in CP side, sniffs zero-rating outgoing traffic and sends information, i.e., packet keyed hashes, to the ISP assistant via ZFree control plane protocol.

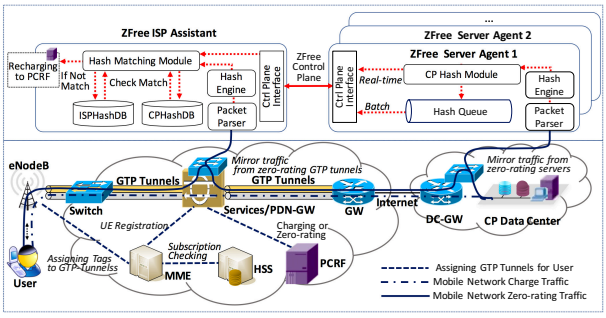


Figure 1: ZFree's Architecture over Mobile Network

We demonstrate how to zero-rate traffics in ZFree, from a mobile connection's perspective. When a user connects to a CP server via a request, mobile ISP looks up the user's IP multimedia private identity (IMPI) via Mobility Management Entity (MME), find the user's subscription information from Home Subscriber Server (HSS), and determine whether the user is subscripted for zero-rating service. If yes, the mobile ISP checks the Host or SNI field, depending on HTTP or HTTPS connection, of the request, and assign a zero-rating GPRS Tunneling Protocol Tunnel (GTP Tunnel) to route the packets to ISP assistant where the ISP assistant sniffs data. Next, the request is transferred via GTP tunnel to the gateway (GW) thus forwarding to the CP data center. CP reply back a response based on the request. ZFree's server agent obtains the response, e.g., via mirroring the traffic, generate keyed hashes and send to ISP assistant over ZFree control plane. At the same time, the original response is transferred to the ISP and encapsulated from the GW back to the zero-rating GTP tunnel. The ISP assistant also obtains the response, generates keyed hashes, matches the hashes with those received from ZFree control plane, and decides whether to zero-rate the traffic. The ISP assistant talkes with ISP Policy and Charging Rules Function (PCRF) if the response traffic should not be zero-rated.

We note that ZFree is designed to prevent free-riding attacks. The ISP assistant will verify CP server's authenticity to prevent the client from connecting to a masqueraded server. At the same time, although the client has free access to modify the end-to-end communication, any modification will be monitored by the ISP assistant via matching packet hash values without intruding users' privacy.

4 ZFree Control Plane Protocol

In this section, we introduce the two-phase, six-step control plane communication protocol in Figure 2, which is triggered by the data plane communication, between the ISP assistant and the server agent. We first discuss the Setup Phase, which is used to establish a connection between the CP and the ISP assistant, in Sec-

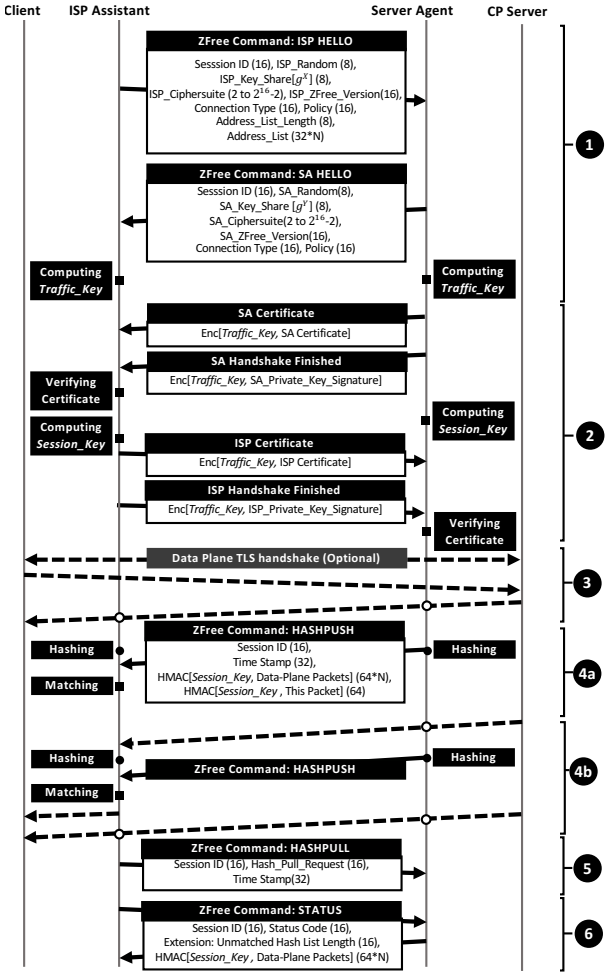


Figure 2: ZFree Control Plane Protocol

tion 4.1 and then the Control Phase, which is used to authenticate the communication between the user and the CP, in Section 4.2.

4.1 Setup Phase

The setup phase, inspired by TLS 1.3, establishes a communication between the ISP assistant and the server agent, which agrees on a list of options, such as cipher suite and connection type, exchanges session keys and then verifies each other's certificate.

1 Handshake. The ISP assistant and the server agent exchanges setup options in the handshake step via "HELLO" messages. Specifically, both parties include a random number of computing keys, exchange cryptographic options, i.e., cipher suites, and agree on a list of ZFree options, such as policies (e.g., zero-rating and parental control), connection type (e.g., blocking vs. non-blocking and real-time vs. batch) indicating how packet hashes are sent to the ISP assistant, and a list of IP address ranges (e.g., 88.88.0.0/16) defining clients behind the ISP. Then, based on the "HELLO" message, both the ISP assistant and the server agent compute a traffic key similar to TLS 1.3 and can be used for future communications in the setup phase.

2 Certificate Verification. In this step, the ISP assistant and the server agent verify each other's certificate and compute a session key for control phase communication. Specifically, the server agent first sends its certificate and private key signature to the ISP assistant for verification. Then, both parties calculate the session key for communication. Next, the ISP assistant also send its certificate and private key signature to the server agent for verification. It is worth noting that the verification of a client certificate is uncommon in TLS communication, but we include it in ZFree protocol so that both parties are verified. Each ISP assistant and server agent has its own certificate. That is, different ISPs may assign different certificates to their ISP assistant, and the same applies to different CPs. Both ISP assistant and server agent have a certificate whitelist that only accepts certain certificates—the whitelist is created based on the mutual agreement between the ISP and the CP.

4.2 Control Phase

After setup, the communication between the ISP assistant and the server agent is triggered by the data plane communication, and we call this control plane communication the control phase. Note that a key point here is that we need to ensure the data integrity but not necessarily data secrecy.

3 Data Plane TLS Setup (Optional). In this step, the client talks with the CP server in the data plane. The communication is in plaintext using TCP or optionally with encryption using TLS. The choice is purely made by the agreement of the client and the CP server.

4 Realtime-type Connection. When a response is sent from the CP server to the client during their communication in the data plane, the control plane communication is correspondingly triggered. Say, the connection type is Realtime (defined in Step 1). The server agent sniffs all the response packets and send the keyed hashes of the responses via "HASHPUSH" messages. Note that the "HASHPUSH" message itself also needs to be hashed with a key to ensure control plane data integrity. Accordingly, the ISP assistant also sniffs and hashes all the response packets with the session key, and matches the hashes with what it receives from the server agent. The ISP assistant takes different actions depending on the ZFree configuration mode.

4a **Non-blocking Mode**. The ISP assistant only sniffs data plane packets.

4b **Blocking Mode**. The ISP assistant blocks data plane packets and allows them only after a match.

5 Batch-type Connection. When the connection

typeturns to Batch (defined in Step ❷). The server agent waits for a "HASHPULL" message from the ISP assistant and then sends a "HASHPUSH" message.

❻ **Status Report.** Both the ISP assistant and the server agent can report the current status to each other, e.g., unmatched hashes, for diagnosis purpose. Similarly, the message is accompanied with a keyed hash of itself to ensure integrity.

5 System Design

We present the system design part of both the server agent and the ISP assistant in this section.

Algorithm 1: 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)
```

5.1 Server Agent

Algorithm 2: 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
```

Algorithm 2 shows the system design of the server agent. In the setup phase, the server agent first establishes a connection with the ISP assistant in the *Handshake* function. Notably, the server agent exchanges ZFree version, connection type, policy as well as Diffie-Hellman cipher suite, pre-shared key and 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 using the traffic key (Line 8). After that, the server agent generates a finish message with its private key signa-

ture encrypted with the traffic key (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 ISP finish message (Line 14–15). Lastly, the server agent verifies ISP's identity with the CA: if verified, it calls *ProcessHash* to start data plane inspection, and otherwise terminates the socket (Line 16–20).

Packets in the data plane trigger the control phase of the server agent. Specifically, the *ProcessHash* function (Lines 21–30), a multithreaded function to efficiently process packets, parses each data plane packet using *ZFreeParseModule* (Line 22), and then calculates the keyed hash value of the packet using *ZFreeHashEngine* (Line 13) with HMAC function. Based on the connection type, the server agent chooses to send the keyed hash in realtime mode (Line 25–27) or batch mode (Line 28–30).

5.2 ISP Assistant

Algorithm 1 shows how the ISP assistant works. In the setup phase, the ISP assistant first creates a connection with the server agent in the *Handshake* function (Line 1–22). 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 key signature with the traffic key (Line 11), and then verifies the server agent's certificate (Line 12). Next, the ISP computes the session key (Line 13) and send its own certificate, private key signature and a finish message to the server agent (Line 14–17). After the connection is established, the ISP assistant checkes "STATUS" messages from the server agent (Lines 38–39).

In the control phase, ISP assistant is triggered by (*i*) a data plane packet, and (*ii*) a control plane "HASH-PUSH" message. First, when a data plane packet comes, the *ISPProcessHash* function (Line 23–29) parses the packet, calculate the keyed hash, and send it to the corresponding distributed hash matching module, i.e., based on the first two bits of the hash (bit and with a *mask* in Line 41), for matching. If match, the ISP assistant sends the packet to the *PCRF_Charging_Module* (Line 27). If no match is found, the ISP assistant saves the hash into the database and wait (Line 29). Second, in *ReceiveSAHash* function (Lines 30–37), when a control plane "HASHPUSH" message comes (Lines 31–36), the ISP assistant also gets the keyed hash value from server agent and uses the distributed hash matching module for matching. Procedures are similar to the first case.

6 Formal Security Analysis

In this section, we perform a formal security analysis on three zero-rating frameworks—Network Cookies [50], IP whitelist [16, 3] and ZFree—using ProVerif [15, 14], an automatic cryptographic protocol verifier. Our ProVerif models are open-source, which can be found in ZFree's repository (https://github.com/zfree2018/ZFREE).

6.1 Formal Models

We model the general zero-rating framework in ProVerif by describing three parties, the client, the CP and the ISP. The client talks with the CP server through the ISP via a bi-directional communication, either unencrypted or encrypted. The unencrypted communication is plaintext; the encrypted communication is based on an existing TLS model [13] and we also introduce a Certificate Authority that issues and verifies the CP's certificate. Now, let us introduce how each framework is modeled.

- Network Cookies. We model a cookie server distributing cookies to all the clients as described in the paper [50]. Specifically, when the cookie server receives a request from a client with both the CP and the client's IP address, the cookie server responds to both the client and the ISP with a cookie descriptor consisting of a cookie ID, a cookie key and a cookie attribute. Next, each message from the client to the CP server has the cookie descriptor to let the ISP verify the message.
- IP Whitelist. We model the IP whitelist to let the ISP check whether the source IP addresses of all the responses match the whitelist. The whitelist is obtained from the ISP via an encrypted communication. Note that such IP whitelist is adopted by several industry proposals [16, 3].
- ZFree. We add two components, i.e., the ISP assistant and the CP server agent, and model the control and data planes described in Section 4 as two communication channels. During the setup phase, the ISP assistant first exchanges handshake information, such as ZFree version, cipher suites, and a policy set, with the server agent, and then calculates session keys. Next, during the communication phase, the CP server agent sniffs all the packets in the data plane channel, generates keyed hashes and sends the information in the control plane channel.

6.2 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. (The request packets are irrelevant because they are generated by the client and can have arbitrary contents.) Specifically, the response sent from the CP server needs to match with the one received by the client as shown in our query to ProVerify at the second row of Table 2. Note that *endResponseVerif*

Table 2: Summary of Formal Verification Results on Network Cookies, IP Whitelist and ZFree.

Goals	ProVerif Queries	Network Cookies [50]		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.

and *beginResponseVerif* are in the client and CP server functions respectively for the verification.

Goal 2: CP Server Authenticity. We ask ProfVerif to verify that the server identity matches with the zero-rating list at the ISP side. Specifically, the *server_cert* of the CP server needs to be verified by the ISP as shown in our query to ProVerify in the third row of Table 2. Similarly, *endServerVerif* and *beginServerVerif* are in the ISP and the CP server.

Goal 3: Application Data Secrecy. We ask ProVerif to verify the secrecy of application data between the client and the CP server as shown at the last row of Table 2.

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.

6.3 Verification Results

An overview of our verification results can be found in Table 2. Some detailed, raw traces can also be found in Appendix A. 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; by contrast, ZFree can defend against free-riding attacks. At the same time, our verification also shows that none of three frameworks changes application layer security, i.e., data secrecy is preserved if traffic is encrypted. Now let us discuss several example violation outputs found by ProVerif.

Output 1 (Network Cookies): Authenticity Violation. When we query $endServerVerif(server\_identity)$, ProVerif outputs 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 server.

Output 2 (Network Cookies & IP Whitelist): Integrity Violation. When we query ProVerif with $endResponseIntegrity(response)$, ProVerif 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 we make an authenticity query to ProVerif for IP whitelist, ProVerif outputs a violation showing that an attacker, as both a client and a man-in-the-middle, can obtain the IP address of the zero-rating CP and insert the IP into the 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 of weak ZFree variations below.

Output 4 (Weak ZFree Variation): Integrity Violation. The first ZFree variation is that we adopt weak hash algorithm, such as SHA-1, instead of SHA-256 in ZFree control plane protocol. 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 "HASH-PUSH" message to include her own hashes of non-zero-rated packets.

Output 5 (Weak ZFree Variation): The second ZFree variation is that we skip the keyed hashes 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, modify the message, and then change the corresponding data plane packet as well.

7 Implementation

We implemented ZFree with 1,890 lines of code (LoC), i.e., 1,100 LoC for the ISP assistant and 790 LoC for the server agent. We also setup a LTE network using ns-3 [6] and another WiFi network using Mininet-WiFi [4]—both network simulators are popular and adopted by many existing works [33, 35, 48]. The LTE network consists of several user equipment (UEs), eNodeBs, PDN gateway, MME and HSS; the WiFi network consists access point (AP) and routers. The entire setup has 950 LoC and detailed configuration can be found in Section 8. Additionally, we also setup a demo website with 836 LoC. In our formal verification, we model Network Cookie, ISP whitelist and ZFree the integrity, secrecy and authenticity queries with 450, 380 and 850 LoC respectively. All the aforementioned source code

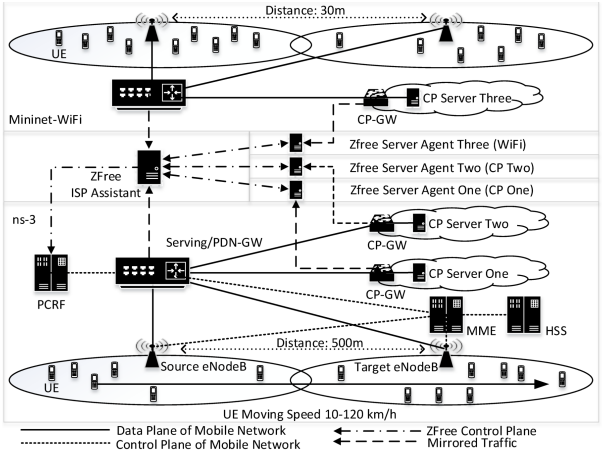


Figure 3: Two Evaluation Test Beds (Mobile and WiFi Environments)

can be found in the following anonymous repository (https://github.com/zfree2018/ZFREE).

8 Evaluation

In this section, we start by describing our environment setup and then introducing each experiment respectively.

Environment Setup. We setup two environments, as shown in Figure 3, to test ISP networks, one mobile network for ZFree's non-blocking mode, and the other WiFi network for ZFree's blocking mode. First, the mobile testing environment is built based on ns-3 [6] in a physical machine with 3.2 GHz Intel due-core i7-6950x CPU, 32GB memory and Ubuntu 16.04 LTS OS. The mobile network consists of the ISP core network and two groups of 1,200 user equipments (UEs) with two by two MIMO antennas. Our ISP core network has a Serving/PDN gateway, a MME, a HSS and a PCRF. The ISP core network is connected with two CP servers via a layer 3 gateway router.

Second, the airplane cabin WiFi testing environment is built based on Mininet-WiFi [4] in a physical machine 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 environment and limit the bandwidth between the APs and the AC as 30 Mbps. Our CP server is equipped with HTTP, HTTPs and iPerf stress testing service.

We deploy ZFree upon these two testing networks: both the ISP assistant and the server agent are Ubuntu 16.04 LTS virtual machines with 1.2GHZ CPU and 12 GB memory. They are connected with the corresponding network with a layer 3 OpenvSwitch (OVS) throught 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 separating from the data plane.

8.1 End-to-end Communication

We first measure the overhead from the perspective of a user of the ISP network with ZFree enabled.

8.1.1 Page Loading Time

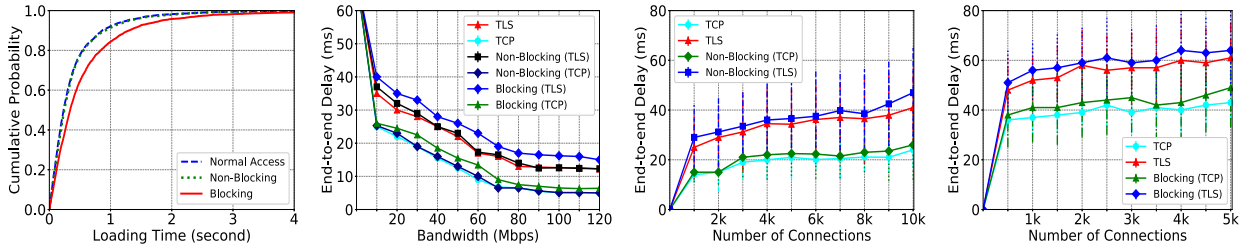
In this experiment, we measure the page loading time for Top 500 Alexa websites with and without ZFree in both blocking and non-blocking modes. Specifically, we setup one of our CP servers as a proxy that relays network traffic from Top 500 Alexa websites. Note that we count all the traffic as zero-rating for the measurement purpose. Figure 4a shows the cumulative distribution function (CDF) graph of the loading time of Top 500 Alexa websites. The median overhead of ZFree's non-blocking mode is very small, i.e., 1.26%, which mainly comes from 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.

8.1.2 Download Time with Different Bandwidth

In this experiment, we test the end-to-end performance when the user accesses the CP server under different bandwidth limits ranging from 0.1Mbps to 120Mbps. Note that each UE is setup with peak downlink speed as 150Mbps and all the experiments are performed six times using legacy TCP connection, legacy TLS connection, TCP connection with ZFree's non-blocking mode and TLS connection with ZFree's blocking mode. Figure 4b shows the results, i.e., the download time of a 900MB video file in the y-axis v.s. the network bandwidth in the x-axis. As expected, the download time decreases as the network bandwidth increases, because the network becomes less crowded. The download time of ZFree's non-blocking mode is almost the same as the native connection, such as TCP and TLS, and the download time of the blocking mode is constantly higher than the native connection.

8.1.3 LTE Handover Testing

In this experiment, we test the end-to-end performance during LTE handover with and without ZFree. Specifically, we setup one UE moving from a cell in the source eNodeB to a cell in another eNodeB located 500meters away 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 [27, 2], and then adopt the iPerf stress test tool to keep the UE receiving data from our CP server. Figure 5 shows our LTE handover testing results. First, the transmis-



(a) Top 500 Websites (b) Limited Bandwidth (c) Scalability of Non-Blocking Mode (d) Scalability of Blocking Mode

Figure 4: 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.

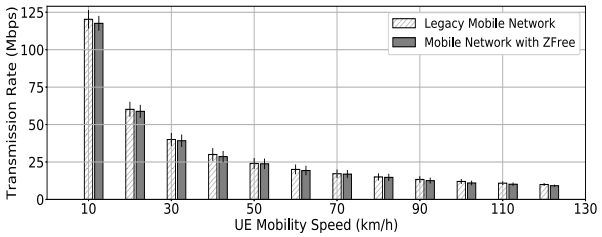


Figure 5: LTE Handover Throughput in both Legacy Mobile Network and Mobile Network with ZFree Enabled

sion speed decreases as traveling speed increases with and without ZFree because the UE has to quickly switch from one cell to another. Second, the transmission speed with ZFree enabled is a little bit smaller than the one without ZFree, i.e., incurring 1.45% overhead. The overhead number is very similar to the one during normal transmission without handover, which means ZFree has little influence on the handover process.

8.2 ISP Core Network

In this experiment, we measure ZFree from the perspective of the ISP core network.

8.2.1 Scalability

In this experiment, we measure whether ZFree can scale when the number of connections increases. Particularly, we measure the end-to-end delay, i.e., the interval between the timestamp at which the client sends a request and the one at which the client receives the response. The experiment is performed in cellular network environment for non-blocking mode and in airplane WiFi environment for blocking mode. Figure 4c and 4d shows the end-to-end delay of non-blocking and blocking modes in the x-axis when the number of connections in the y-axis increases. In both figures, the end-to-end delays of TCP and TLS without ZFree are shown as a baseline for comparison. Our results show that the end-to-end delay is almost flat as the number of connections increases.

8.2.2 Stress Test

In this section, we perform a stress test of ZFree in terms of network latency and bandwidth following RFC 2544 [7], which documents benchmarking methodology for network interconnect devices. Specifically, we replay real-world traffic captured from netresec [5] and tcpReplay [12] in network access point. The Netresec network trace [5] has high-speed (8–10Gbps) network flows with 40 million packets from 1,982 applications, and the other [12] low-speed (500Mbps) network flows with 791,615 packets from 132 applications. 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 a bursty traffic scenario in the test.

The testing methodology works as follows. We use TCPreplay [9], a popular replaying software, to rewrite the packet header including the source IP, the destination IP, the source MAC address and the destination MAC address of the traffic. We also uniformly randomize the destination IP and MAC addresses of all the flows to different UEs so that the traffic can be evenly distributed inside the network.

Figure 6 shows the network traffic in data plane and corresponding CPU Usage for the ISP Assistant (top) 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 (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 has 1.582Gbps peak traffic in legacy network and 1.571Gbps with ZFree's server agent as shown

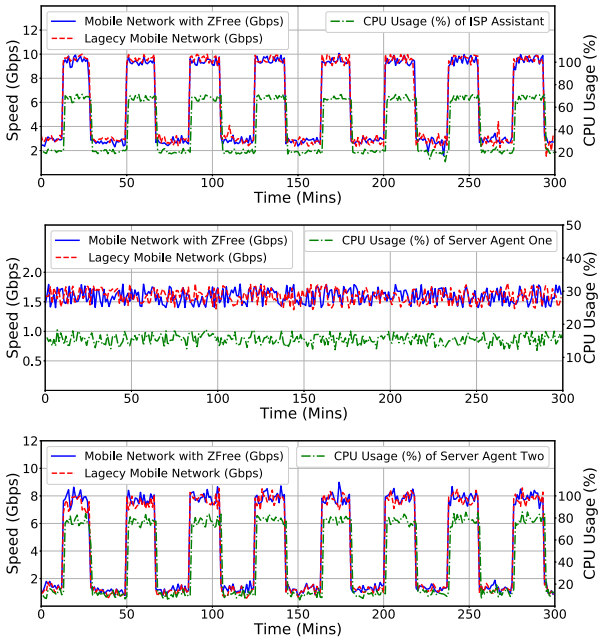


Figure 6: Network Traffic (Gbps) and CPU Usage (%) for the ISP Assistant (Top) and two CP Agents (Middle and Bottom) under Stress Test

in Figure 6 (middle). The average CPU usage for the first CP server is 15.4%. Our second CP server has 4.332Gbps peak traffic in legacy network and 4.213Gbps with ZFree's server agent as shown in Figure 6 (middle). 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.

8.2.3 Control Plane Overhead

In this part, we measure two overhead: 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 and calculate the volumes of packets between the ISP assistant and the server agent compared with the total amount of traffic. Our evaluation shows that ZFree introduces a small amount, i.e., 4.2%, of additional traffic in terms of control plane communication overhead.

Second, we compare ZFree control plane protocol with a naive implementation that transfers plain hash values in a TLS connection. Our evaluation shows that such implementation incurs 2.8 times more overhead than ZFree control plane protocol when processing 100MB data plane traffic. The reason is that keyed hash is cheap as compared to encryption, such as AES.

8.3 Security

In this experiment, we evaluate the security of our ZFree implementation by using three types of zero-rating attacks. Two types are documented in Section 2, i.e., masquerading a CP server and modifying the response packet from the CP server. We also perform a TCP retransmission-based free-riding attacks [20] against ZFree. Specifically, we add two virtual switches, one between the ISP and the CP gateway, and the other between the client and the ISP. The former is used to modify response packets, e.g., encapsulating packets into TCP retransmission, and the latter 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 client can not get the response until timeout.

9 Discussions

In this section, we discuss several aspects of ZFree. First, we discuss *ethics concerns* for the free-riding attacks that we launched against real-world ISPs. During all the experiments, we try to limit the damage that could occur to these ISPs. We only downloaded a small amount of 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 mobile networks, we paid the ISP 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 tried our best to inform the tested ISPs about the found vulnerabilities. All the tested ISPs are informed of this issue.

Second, we discuss the general issue about *network neutrality*. As mentioned by Yiakoumis et al. [50], some people raised concerns that certain 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 zero-rating services provided by ISP.

Third, we discuss how *third-party contents*, e.g., ads included in a webpage, are zero-rated. The current prototype of ZFree 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, T-Mobile 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 in mobile network scenario the case is even sometimes simplified, because many mobile ISPs host their own CDN and provide contents directly from their base station. That is, the server agent and the ISP assistant may be co-located in the same local network.

Lastly, we talk about the robustness of ZFree against DoS attacks. ZFree computes the hashes of server responses but not requests. That is, if there exists DoS attacks, the CP server is the target before ZFree, which can help ZFree to filter DoS requests. In practice, a DoS attack filter is deployed at the CP's gateway and ZFree is located behind this DoS attack filter.

10 Related Work

We discuss related work in this section.

10.1 Existing Attacks

We categorize existing attacks on ISP Policy and Charging Rules Function (PCRF) [10, 8] 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 [21, 20], DNS [41] 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 2 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 [31, 21, 42]. This type of attack is out of scope and one can refer to existing works [31, 21, 42] for solutions.

10.2 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 [47, 32, 51] to differentiate network traffic. Similarly, many other approaches [26, 28, 49, 44, 45, 52] 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) [34] and Session Protocol for User Datagrams (SPUD) [23] are two new protocols that allow middlebox to inspect end-to-end traffic. Yiakoumis et al. [50] propose a traffic authentication architecture so-called network cookie to provide on demand zero-rating services. Facebook Zero [11, 3] 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.

10.3 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 [30, 43, 38, 29, 46, 24, 17, 22, 40] analyzes traffic without decryption or understanding the traffic. Such approach, though being effective in solving their own problem, cannot correctly zero-rate traffic without collaborating with the CP server. Whitebox approaches, such as mcTLS [37] and APIP [36], 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 [39, 19], 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.

11 Conclusion

To mitigate such free-riding attacks, in this paper, we propose a secure, backward compatible, zero-rating framework, called ZFree, which authenticates and verifies all the communications between the CP and the client. ZFree is formally verified as secure against free-riding attacks. We implemented a prototype of ZFree and our evaluation on two test beds, one mobile network and the other WiFi network, shows that ZFree is lightweight, secure, and scalable.

12 Acknowledgement

We would like to thank anonymous reviewers for their helpful comments and feedback. This work was supported in part by a gift from Cisco and National Science Foundation (NSF) grants CNS-15-63843. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF.

References

[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] (2014, Dec.) Delivering zero-rated traffic. 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] K. Arai. (2015-2016) Formal verification of tls 1.3 full handshake protocol using proverif. https://www.cellos-consortium.org/studygroup/TLS1.3-fullhandshake-draft11.pv.

[14] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," in *14th IEEE Computer Security Foundations Workshop (CSFW-14)*. Cape Breton, Nova Scotia, Canada: IEEE Computer Society, Jun. 2001, pp. 82–96.

[15] ——, "Modeling and verifying security protocols with the applied pi calculus and proverif," *Foundations and Trends in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, Oct. 2016. [Online]. Available: https://doi.org/10.1561/3300000004

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

[17] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '07. New York, NY, USA: ACM, 2007, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/1282380.1282382

[18] A. Chen, A. Haeberlen, W. Zhou, and B. T. Loo, "One primitive to diagnose them all: Architectural support for internet diagnostics," in *Proceedings of the Twelfth European Conference on Computer Systems*, ser. EuroSys '17. New York, NY, USA: ACM, 2017, pp. 374–388.

[19] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking ssl development in an appified world," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 49–60.

[20] Y. Go, J. Won, D. F. Kune, E. Jeong, Y. Kim, and K. Park, "Gaining control of cellular traffic accounting by spurious tcp retransmission," in *Network and Distributed System Security (NDSS) Symposium 2014*. Internet Society, 2014, pp. 1–15.

[21] Y. Go, D. F. Kune, S. Woo, K. Park, and Y. Kim, "Towards accurate accounting of cellular data for tcp retransmission," in *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '13. New York, NY, USA: ACM, 2013, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/2444776.2444779

[22] R. Gold, P. Gunningberg, and C. Tschudin, "A virtualized link layer with support for indirection," in *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, ser. FDNA '04. New York, NY, USA: ACM, 2004, pp. 28–34. [Online]. Available: http://doi.acm.org/10.1145/1016707.1016713

[23] J. Hildebrand and B. Trammell, "Substrate Protocol for User Datagrams (SPUD) Prototype," Internet Engineering Task Force, Internet-Draft draft-hildebrand-spud-prototype-03, Mar. 2015, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-hildebrand-spud-prototype-03

[24] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 51–62. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402966

[25] A. M. Kakhki, F. Li, D. Choffnes, E. Katz-Bassett, and A. 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*, ser. Internet-QoE '16. New York, NY, USA: ACM, 2016, pp. 43–48. [Online]. Available: http://doi.acm.org/2940136.2940140

[26] P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *NSDI*, 2013, pp. 99–111.

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

[28] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, "Veriflow: Verifying network-wide invariants in real time," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 467–472, 2012.

[29] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: securely outsourcing middleboxes to the cloud," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, 2016, pp. 255–273.

[30] P. Lepeska, "Trusted proxy and the cost of bits," in *90th IETF meeting*, 2014.

[31] C.-Y. Li, G.-H. Tu, C. Peng, Z. Yuan, Y. Li, S. Lu, and X. Wang, "Insecurity of voice solution volte in lte mobile networks," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: ACM, 2015, pp. 316–327. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813618

[32] P.-C. Lin, Y.-D. Lin, Y.-C. Lai, and T.-H. Lee, "Using string matching for deep packet inspection," *Computer*, vol. 41, no. 4, 2008.

[33] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 113–126. [Online]. Available: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/liu_junda

[34] D. Migault, "LURK Protocol for TLS/DTLS1.2 version 1.0," Internet Engineering Task Force, Internet-Draft draft-mglt-lurk-tls-01, Mar. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-mglt-lurk-tls-01

[35] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti, "Numfabric: Fast and flexible bandwidth allocation in datacenters," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 188–201.

[36] D. Naylor, M. K. Mukerjee, and P. Steenkiste, "Balancing accountability and privacy in the network," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 75–86, 2015.

[37] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. 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*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 199–212. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787482

[38] M. O'Neill, S. Ruoti, K. Seamons, and D. Zappala, "Tls proxies: Friend or foe?" in *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM, 2016, pp. 551–557.

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

[40] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, no. 23-24, pp. 2435–2463, Dec. 1999. [Online]. Available: http://dx.doi.org/10.1016/S1389-1286(99)00112-7

[41] C. Peng, C.-y. Li, G.-H. Tu, S. Lu, and L. Zhang, "Mobile data charging: New attacks and countermeasures," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 195–204. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382220

[42] C. Peng, C.-Y. Li, H. Wang, G.-H. Tu, and S. 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*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 727–738. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660346

[43] A. Rao, J. Sherry, A. Legout, A. Krishnamurthy, W. Dabbous, and D. Choffnes, "Meddle: Middleboxes for increased transparency and control of mobile traffic," in *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, ser. CoNEXT Student '12. New York, NY, USA: ACM, 2012, pp. 65–66. [Online]. Available: http://doi.acm.org/10.1145/2413247.2413286

[44] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann, "Opensdwn: Programmatic control over home and enterprise wifi," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15. New York, NY, USA: ACM, 2015, pp. 16:1–16:12. [Online]. Available: http://doi.acm.org/10.1145/2774993.2775002

[45] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, A. Feldmann, and R. Riggio, "Programming the home and enterprise wifi with opensdwn," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 117–118.

[46] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[47] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 213–226. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787502

[48] E. Weingärtner, F. Schmidt, H. Vom Lehn, T. Heer, and K. Wehrle, "Slicetime: a platform for scalable and accurate network emulation," in *Proceedings of NSDI'11: 8th USENIX Symposium on Networked Systems Design and Implementation*, 2011, p. 253.

[49] Y. Wu, A. Chen, A. Haeberlen, B. T. Loo, and W. Zhou, "Automated bug removal for software-defined networks," in *Proceedings of USENIX Symposium of Networked Systems Design and Implmentation (NSDI)*, Mar. 2017.

[50] Y. Yiakoumis, S. Katti, and N. McKeown, "Neutral net neutrality," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 483–496. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934896

[51] F. Yu, Z. Chen, Y. Diao, T. Lakshman, and R. 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.* IEEE, 2006, pp. 93–102.

[52] Y. Zhang, Z. M. Mao, and M. Zhang, "Detecting traffic differentiation in backbone isps with netpolice," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 103–115. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644905

A Raw Trace Example Outputted by ProVerif

In the appendix, we show five examples of raw traces outputted by ProVerif. Figure 7 shows a counter example against response integrity for Network Cookies Model; Figure 8 shows another counter example trace against CP authenticity for IP whitelist based zero-rating framework. Figure 9 shows a successful verification example of ZFree. Figure 10 shows a counter example trace if the ZFree's packet hash is removed. Figure 11 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 7: 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 8: 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: Successful example traces on verifying all properties of ZFree (TLS Connection)

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 endResponseV-
erif_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 10: Counter example traces on verifying a weak version of ZFree, i.e., removing control-plane keyed hash (TLS Connection)

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 11: Counter example traces on a weak version of ZFree with a weak hash function (TLS Connection)