ივანე ჯავახიშვილის სახელობის თბილისის
სახელმწიფო უნივერსიტეტი

საბაკალავრო ნაშრომი თემაზე:

დაცული ონლაინ გადახდების სისტემა

ნიკა გარუჩავა

ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი
კომპიუტერული მეცნიერების დეპარტამენტი

ხელმძღვანელი: ასოცირებული პროფესორი მაგდა ცინცაძე

თბილისი 2017

აბსტრაქტი

ონლაინ გადახდების სისტემა მომხმარებელს საშუალებას აძლევს გააკეთოს გადახდა ონლაინ მერჩანტთან ან სერვის პროვაიდერთან. გადახდების gateway-ები, არხი მომხმარებელსა და გადახდების პროცესორს შორის, იყენებენ სხვადასხვა სახის დაცვის მექანიზმებს, რათა უზრუნველყონ მომხმარებლის გადახდის ინფორმაცია (საკრედიტო ან სადებეტო ბარათის ინფორმაცია) ონლაინ გადახდისას. მაგრამ gateway-ს უსაფრთხოება არაა ხშირად საკმარისი, რადგან მერჩანტს აქვს შესაძლებლობა მოიპოვოს საბარათე ინფორმაცია გადახდისას რაიმე ფორმით. გარდა ამისა, ზოგიერთი მერჩანტი არ სთავაზობს გადახდის უსაფრთხო გარემოს მომხმარებელს. შესაბამისად, ეს ყველაფერი ქმნის საშიშროებას, რომ მომხმარებლის საბარათე მონაცემები იყოს არასწორად გამოყენებული მერჩანტის მიერ, ან სულაც ჰაკერებმა მოიპოვონ მასზე წვდომა.

ნაშრომში გთავაზობთ ახალ მიდგომას გადახდების სისტემებისადმი, რომლის მეშვეობითაც კლიენტის საბარათე ინფორმაცია ხელმიუწვდომელია მერჩანტის მიერ. მომხმარებელი თავის საბარათე მონაცემებს აგზავნის პირდაპირ გადახდების gateway-ში და gateway, მონაცემების ვერიფიცირების შემდეგ აგზავნის გადახდას მერჩანტთან, ხოლო საბარათე ინფორმაციას ნებისმიერი ინტერნეტ-საშიშროებებისგან ვიცავთ Payment Card Industry Data Security Standard (PCI DSS)-ის მითითებების გამოყენებით.

Abstract

Online payment system is allowing customers to make payment with online merchant or service provider. Payment gateway is a bridge between costumer and payment processor, which uses a variety of safety mechanisms to safely store costumer's payment information (credit or debit card details) while processing online payment. Most of the times gateway's safety system might not be enough as merchant can get access to card details in some form. In addition, some merchants do not even offer safe payment conditions to the costumers. Consequently, this threatens the protection of card details, and risks it to be misused by merchant himself or stolen by hackers.

The thesis provides with the new approach of how can payment systems work without being accessed by merchants. Costumers will send their information directly to the payment getaway, which after verifying transaction sends a payment to the merchant.
Using Payment Card Industry Data Security Standard (PCI-DSS) we can protect a costumer's card information from internet vulnerabilities.

ჩანართის სია

სარჩევი

თავი I: შესავალი

საქონლის და სერვისების ყიდვა/ გაყიდვა ინტერნეტში ცნობილია როგორც ელექტრონული კომერცია (e-commerce). E-commerce-ის კონცეფცია მოიცავს არა მხოლოდ ყიდვასა და გაყიდვას, არამედ ასევე მარკეტინგს, სერვისებს, მიტანას და გადახდებს პროდუქტებისა და სერვისებისთვის.

E-commerce-ის განვითარებასთან ერთად, გადახდების სისტემები და პროტოკოლები განვითარდა. უმრავლესობა დღევანდელი გადახდების სისტემებისა მოიცავს მომხმარებლებს, მერჩანტებს და გადახდების gateway-ებს შემდეგი სახით: მერჩანტი აგზავნის მომხმარებლის საბარათე მონაცემებს გადახდის gateway-ში გადახდის შესასრულებლად, რაც რისკის ქვეშ აყენებს კლიენტის საბარათე მონაცემებს, რადგან მერჩანტმა შეიძლება შეინახოს საბარათე მონაცემები ღია ან დაშიფრული სახით და შემდეგ გამოიყენოს ის. ასევე შეიძლება მერჩანტის სერვერი საიდანაც გადახდის gateway-ს ეგზავნება მომხმარების საბარათე ინფორმაცია, კომპრომეტირებული იყოს და მერჩანტმა საერთოდ არ იცოდეს ამის შესახებ.

2006 წელს ხუთმა დიდმა გადახდების ბრენდმა, VISA, MasterCard, Amex, JCB, International and Discover Financial Service-მა დაარსა უსაფრთხოების სტანდარტების საბჭო რომელსაც დაარქვეს Payment Card Industry Data Security Standard (PCI DSS). PCI DSS არის ღია გლობალური ფორუმი რომელიც ანვითარებს და მართავს PCI უსაფრთხოების სტანდარტებს და ასევე ინფორმაციას ავრცელებს მერჩანტებში და მომხმარებლებში ამ სტანდარტების შესახებ. ისინი ასევე მუშაობენ Payment Application Data Security Standard (PA-DSS) და Pin Transaction Security (PTS) მოთხოვნილებებზე. ონლაინ მერჩანტი, რომელიც იყენებს ამ ხუთ გადახდების ბრენდს გადახდებისთვის, აუცილებლად უნდა მისდიონ PCI DSS-ის პოლიტიკას, წინააღმდეგ შემთხვევაში მათ აქვთ ვაჭრობის უფლება ჩამოართვან მერჩანტს, მაგრამ ონლაინ მერჩანტმა შეიძლება არ მისდიოს PCI DSS-ის ყველა წესს. ამას გარდა წესების დარღვევას ხშირად ააშკარავაბენ, როდესაც ვინმე უჩივლებს მერჩანტს ან რაიმე სახით მერჩანტის სისტემას გატეხავენ.

კლიენტის საგადასახადო ინფორმაცია ონლაინ გადახდისას უმეტესწილად არის საბარათე ინფორმაცია (PAN, ვადის გასვლის თარიღი, CVV), რომლის გადაცემა მერჩანტისთვის ელექტრონული გადახდისას რისკის ქვეშ აყენებს ადვილად გამოყენებად ფინანსურ ინფორმაციას. ასეთი რისკებთაგანია მონაცემთა ფალსიფიცირება (data tampering), საბარათე ინფორმაციის მოპარვა ან/და საკრედიტო ბარათებით თაღლითობა. მერჩანტი შეიძლება ცუდად არ იყენებდეს კლიენტის ინფორმაციას, მაგრამ შეიძლება დაუცველად ინახავდეს მას და სერვერი ან სისტემა არ ჰქონდეს სათანადოდ დაცული რომ თავიდან აირიდოს სხვადასხვა ჰაკერული შეტევა (spyware, malware და ა.შ).

ნაშრომში შემოთავაზებულია დაცულ გადახდების სისტემის მოდელი, სადაც მომხმარებლის ფინანსური ინფორმაცია პირდაპირ იგზავნება გადახდების gateway-ში და მერჩანტი დაშიფრული/დაჰეშილი სახითაც ვერ მოიპოვებს სრულ საბარათე ინფორმაციას. ბარათის ციფრების დამალული სახით გადაცემა მერჩნატისთვის დაშვებულია, თუ ჩანს მხოლოდ ბოლო 4 ციფრი და წინა ციფრები შეცვლილია სხვა სიმბოლოებით, მაგალითად “*”-ით (**********1111).
გადახდების gateway-ები არიან სანდოები, რადგან ისინი იყენებენ მონაცემთა სტანდარტული დაცვის მეთოდებს და ურთიერთობენ ბანკებთან და საკრედიტო ბარათების კომპანიებთან ყველაზე დაცული მეთოდებითა და ტექნოლოგიებით.

1.1 პრობლემის აღწერა

მიუხედავად უსაფრთხოების სტანდარტებისა, მომხმარებლის ფინანსური ინფორმაციის გაგზავნა მერჩანტის გავლით არის ერთ-ერთი ძირითადი მიზეზი მომხმარებლის ინფორმაციის მოპარვის დღევანდელ e-commerce სისტემებში. როდესაც მომხმარებელის საბარათე მონაცემები სრული სახით იგზავნება ონლაინ მერჩანტს, მერჩანტს აქვს შესაძლებლობა ამოიღოს მომხმარებლის საბარათე ინფორმაცია როგორიცაა: ბარათის ნომერი, ბარათის გამცემი (credit card issuer), ბარათის ვადა, ბარათის მფლობელის სახელი და

CVV/CVC. დაშიფრული/დაჰეშილი სახით გაგზავნისას მერჩანტმა შეიძლება შეინახოს ეს ინფორმაცია და შემდეგ შეუდგეს სხვადასხვა მეთოდებით მის გაშიფვრას.

ძირითადი თაღლითობები, რომლებიც დღეს ინტერნეტში ხდება, შესრულებულია ჰაკერების, თაღლითი მერჩანტების, სპამერების, ფიშერების, malware, spyware და მონაცემთა ქურდების მიერ რომლებიც ახორციელებენ შეტევებს ქსელებზე და პერსონალურ კომპიუტერებზე რათა მოიპარონ ან დააზიანონ ინფორმაცია. ამის თავიდან ასარიდებლად სასურველია საერთოდ არ გავაგზავნოთ კლიენტის ფინანსური ინფორმაცია მერჩანტთან, რადგან ეს ქმნის სისტემის გატეხვისა და ინფორმაციის გამოჟონვის შესაძლებლობას მერჩანტის მხრიდან.

1.2 თეზისის მიმოხილვა

ამ თეზისში, გთავაზობთ ონლაინ გადახდების სისტემას რომელშიც მომხმარებლის საბარათე ინფორმაცია მერჩანტის გვერდის ავლით იგზავნება გადახდის gateway-ში. ეს მიდგომა იცავს მომხმარებლის საბარათე ინფორმაციას არასანქცირებული გამოყენებისგან. შემოთავაზებულ აგებულია PCI DSS-ის უსაფრთხოების პოლიტიკაზე დაყრდნობით.

თავი II: დაკავშირებული ნამუშევრები

ერთ-ერთი ყველაზე სახელგანთქმული გადახდების სისტემა არის Secure Electronic Transaction (SET), რომელიც შექმნა კომპანია SETco-მ Visa და MasterCard-ის თაოსნობით 1996 წელს. SET იყო არამხოლოდ გადახდის სისტემა, ეს იყო სტანდარტული პროტოკოლი საკრედიტო ბარათების ტრანზაქციების დაცვისა ინტერნეტში. სამწუხაროდ პროექტი ჩავარდა სხვადასხვა მიზეზის გამო, როგორიცაა კლიენტის მხარეზე პროგრამის ინსტალაცია, ფასი და სირთულე.

SET-ის ჩავარდნის შემდეგ Secure Sockets Layer (SSL) and Transport Layer Security (TLS) გამოიყენებოდა უსაფრთხო ელექტრონული კომუნიკაციისა და კომერციისთვის. Visa-მ და MasterCard-მა შექმნა საკუთარი პროტოკოლი დაცული ელექტრონული გადახდებისათვის, სახელად, 3-D Secure როგორც Verified by Visa და MasterCard SecureCode. ასევე American Express იყენებს SafeKey-ს და JCB (Japan Credit Bureau) - J/Secure-ს. გადახდის უსაფრთხოებისთვის ეს პროვაიდერები აერთიანებენ ფინანსურ ავტორიზაციას ონლაინ აუტენთიფიკაციასთან, რომელშიც ონლაინ აუტენთიფიკაცია აერთიანებს როგორც კლიენტის ასევე სერვერის აუტენთიფიკაციას და ფინანსური ავტორიზაცია ნიშნავს გადახდის ინფორმაციის ავტორიზაციას საიდუმლო პაროლის გამოყენებით, რომელიც მიბმულია სადებეტო ან საკრედიტო ბარათზე.

2.1 SET

ინტერნეტის გამოგონებამ და e-commerce ზრდამ და განვითარებამ გამოიწვია უსაფრთხო გადახდების მოთხოვნა. ამ მიზეზით, Visa-მ და MasterCard-მა დაიწყეს SET-ის შექმნა სხვა კომპანიებთან ერთად როგორებიცაა GTE, IBM, Microsoft, Netscape, RSA, Safelayer და VeriSign. SET გათვლილი იყო სტადნარტული პროტოკოლი ონლაინ გადახდების სისტემებისთვის,

მაგრამ ხარჯების და სირთულის გამო პროექტი ჩაიშალა. მიუხედავად ჩაშლისა SET-მა სიახლე შემოიტანა კრიპტოგრაფიაში, ორმაგი ხელმოწერის სახით.

2.2 Verified by Visa & SecureCode

SET-ის ჩავარდნასთან ერთად, e-commerce კომპანიებს უწევდათ SSL-ის გამოყენება დაცული ინფორმაციის გადასაცემად ინტერნეტში. SSL ქმნის უნიკალურად დაშიფრულ არხს დახურული კომუნიკაციისთვის ღია არხებში და ამოწმებს სერვერის სერტიფიკატს ინფორმაციის გაგზავნამდე. სენსიტიური ინფორმაციის გაგზავნამ SSL-ის მეშვეობით შექმნა უსაფრთხოება, მაგრამ კომპლექსური ტექნოლოგიების გამოყენების გამო, მალე გამოჩნდა მისი მემკვიდრე TLS. ის გამოიყენება გადაცემის ისეთი პროტოკოლების ენკაფსულაციისთვის, როგორიცაა HTTP, FTP, SMTP, NNTP და XMPP. TLS-ის ცნობილი გამოყენების მაგალითია HTTPS.

მიუხედავად იმისა, რომ SSL და TLS ინფორმაციის დაცულად გასაგზავნად გამოიყენება, e-commerce და საკრედიტო ბარათების კომპანიებს ჭირდებოდათ უფრო საიმედო გადახდის მეთოდი ან სისტემა, რათა დაეცვათ თავიანთი მომხმარებლების ფინანსური ინფორმაცია. ამ მიზეზით Visa-მ შექმნა თავისი დაცული პროტოკოლი Verified by Visa და მასტერქარდმა - SecureCode. ორივე მათგანი არის დამატებითი უსაფრთხოების დონე თაღლითური ტრანზაქციებისგან დასაცავად. ისინი ადასტურებენ ბარათის მფლობელის ნამდვილობას პაროლზე დაფუძნებული მეთოდით. ორივე მეთოდი დღეს გამოყენებაშია და სანდოა. მაგრამ ორივე შემთხვევაში მომხმარებლის საგასახადო ინფორმაცია იგზავნება გადახდების gateway-ში მერჩანტის მიერ.

ერთადერთი განსხვავება არის უსაფრთხოების იმპლემენტაცია ბარათის მფლობელის აუთენთიფიკაციის ველის (Cardholder Authentication Verification Value (CAVV)) დაგენერირებაში. MasterCard იყენებს Accountholder Authentication Value (AAV)-ს და Visa Cardholder Authentication Verification Value (CAVV)-ს. ორივე მათგანი არის 3 ციფრიანი უსაფრთხოების კოდი, რომლითაც მომხმარებელი გადის აუთენთიფიკაციას. ეს 3 ციფრიანი კოდი არის დამატებითი დაცვის ფენა შემოღებული ბარათების მწარმოებლების მიერ იმის დასადასტურებლად აქვს თუ არა ბარათი შესაბამის კლიენტს.

ორივე მეთოდის დროს, ტრანზაქციის დროს რედირექტი ხდება მერჩანტის გვერდიდან ბარათის გამცემი ბანკის გვერდზე ტრანზაქციის ავტორიზაციისთვის. მართალია ინფორმაციის უსაფრთხოებას უზრუნველყოფს ბანკი, მაგრამ არის საშიშროება man-in-the-middle შეტევის, თუ კლიენტის ბროუსერს არ შეუძლია ვერიფიცირება გაუკეთოს სერვერის SSL სერტიფიკატს ამ გვერდზე.

2.3 გადახდის სისტემები და გადახდის gateway-ები

ონლაინ გაყიდვებზე და უსაფრთხოებაზე მოთხოვნამ ხელი შეუწყო სხვადასხვა გადახდის gateway-ების და სისტემების ჩამოყალიბებას. მათ შორის, ზოგიერთი დაცული გადახდების სისტემა, მაგალითად Skipjack, აძლევს შედარებით მცირე ინფორმაციას მერჩანტებს მომხმარებლის საგადასახადო ინფორმაციაზე (სადებეტო/საკრედიტო ბარათის ინფორმაცია).

Skipjack იყენებს ორ მეთოდს ონლაინ გადახდისათვის: მერჩანტის წამოწყებულ და კლიენტის წამოწყებულს. მერჩანტის წამოწყებული გადახდა არის ახლანდელი მიდოგმა გადახდისა, რომელშიც მერჩანტი იღებს კლიენტისგან საბარათე მონაცემებს და აგზავნის გადახდების gateway-ში. როცა მერჩანტი იყნებეს Skipjack-ის მერჩანტისგან წამოწყებულ მეთოდს, Skipjack შიფრავს მომხმარებლის საბარათე მონაცემებს მერჩანტის გადახდების ფორმიდან და აგზავნის ბარათის გამცემ ბანკში ავტორიზაციისთვის. Skipjack-ის მომხმარებლისგან წამოწყებულ მეთოდში, მერჩანტი ქმნის დაცულ გადახდების გვერდს Skipjack-ში და სვამს თავის ვებ გვერდში გადახდისთვის. როცა მომხმარებელი აკეთებს გადახდას, მას საბარათე

ინფორმაცია შეყავს პირდაპირ Skipjack-ის დაცულ ფორმაში. ამ მეთოდისას, როგორც ჩემს მიერ შემოთავაზებულ მიდგომისას გადახდისთვის, საბარათე მონაცემები იგზავნება პირდაპირ გადახდების gateway-ში. მაგრამ Skipjack აძლევს მერჩანტს მცირე ინფორმაციას მომხმარებლის გადახდაზე, მაგალითად ბარათის ბოლო ოთხ ციფრს, და ასევე აძლევს მერჩანტს საშუალებას მომხმარებელს არ შეაყვანინოს CCV. მაგრამ ეს არ ეწინააღმდეგება PCI DSS-ის სტანდარტებს.

CCV არის ბარათის დაცვის მექანიზმი და ის გვაძლევს ინფორმაციას იმაზე აქვს თუ არა მომხმარებელს ეს ბარათი თან. ის ასევე ეხმარება ბარათის გამცემს ვერიფიცირება გაუკეთოს ბარათის მონაცემებს და ბარათის მფლობელის ვინაობას.

2.4 გადახდის სისტემის მუშაობის ბიჯები

გადახდის სისტემის მუშაობის სქემა შემდეგია:

1. მომხმარებელი შედის მერჩანტის ვებ გვერდზე, ირჩევს სერვისებს გადასახდელად და ამატებს მათ კალათაში.

2. როცა მომხმარებელი მზადაა გადასახდელად, მერჩანტს გადაყავს მომხმარებელი გადახდების gateway-ში გადახდის ავტორიზაციისთვის. მას იქ დაცულ ფორმაში შეყავს საგადახდო ინფორმაციას. საგადახდო ინფორმაცია მოიცავს მომხმარებლის სადებეტო ან საკრედიტო ბარათს.

3. გადახდების gateway ამოწმებს მომხმარებლის საგადახდო ინფორმაციას და ნამდვილობის შემთხვევაში ავტორიზირებას უკეთებს გადახდას. შემდეგ იგი უგზავნის გადახდის ტოკენს მერჩანტს. გადახდის ტოკენი არის ინდიკატორი გადახდის ავტორიზირებისა. მერჩანტს ჭირდება გადახდის ტოკენის ინფორმაცია როდესაც ითხოვს გადახდას გადახდების gateway-სგან.

4. ტოკენის მიღების შემდეგ მერჩანტი უგზავნის მომხმარებელს ინფორმაციას გადახდის ავტორიზაციის შესახებ და ასევე სერვის პროვაიდერს უგზავნის ინფორმაციას გადახდის შესახებ.

5. მერჩანტი სერვის პროვაიდერისგან გადახდის მოთხოვნაზე დადებითი პასუხის შემდეგ ურიცხავს მას სერვისის საფასურს.

6. მერჩანტი აგზავნის გადახდების gateway-ში გადახდის დასრულების მოთხოვნას და gateway რიცხავს სერვისის საფასურს მერჩანტის ანგარიშზე.

თავი 3: უსაფრთხო გადახდების სისტემა

3.1 მუშაობის პრინციპის მიმოხილვა

ჩვენს მიერ შემუშავებული უსაფრთხო ონლაინ გადახდების სისტემის მუშაობის პრინციპი შემდეგია:

როდესაც მომხმარებელს უნდა შეასრულოს ონლაინ გადახდა მერჩანტისგან, ის შედის მერჩანტის ვებ გვერდზე, ირჩევს სასურველ პროდუქტს და ამატებს მათ კალათაში (მერჩანტის ვებ გვერდზე). როდესაც მომხმარებელი მზადაა გადაიხადოს სერვისების საფასური, მერჩანტი არეგისტრირებს გადახდას გადახდის gateway-ში და მოხმარებელი გადაყავს gateway-ს მიერ დაგენერირებულ ლინკზე საბარათე ინფორმაციის შესაყვანად. gateway არის PCI DSS-ის მიერ სერტიფიცირებული და აქვს უფლება აწარმოოს გადახდები სხვადასხვა ტიპის ბარათებით.

საბარათე ინფორმაციის შეყვანის შემდეგ gateway ავტორიზაციას უკეთებს ბარათს და თანხის არსებობის შემთხვევაში ბლოკავს მას და მომხმარებელი გადაყავს გადახდის სტატუსის გვერდზე მერჩანტის ვებ გვერდზე. მერჩანტი ამ დროს ამოწმებს გადახდის სტატუსს. თუ თანხა დაიბლოკა იგი უგზავნის სერვის პროვაიდერს გადახდის მოთხოვნას და თანხმობის შემთხვევაში ურიცხავს მას თანხას თავისი ანგარიშიდან. ამის შემდეგ მერჩანტი აგზავნის მოთხოვნას gateway-ში გადახდის დასრულების მოთხოვნას და მერჩანტს ერიცხება შესაბამისი თანხა.

თანხის დაბლოკვის შემდეგ რაიმე შეცდომის უარყოფითი პასუხის წარმოშობის შემთხვევაში მერჩანტი, უგზავნის გადახდის gateway-ს მოთხოვნას გადახდის გაუქმებისათვის, გადახდის გეითვეი ამ დროს აკეთებს რევერსალს(reversal) ფინანსური დღის დასრულებამდე, მის შემდეგ კი რეფანდს (refund). სქემა ნაჩვენებია სურათზე.

3.2 უსაფრთხო ონლაინ გადახდის სისტემის არქიტექტურა

შექმნილი გადახდების სისტემის არქიტექტურა შედგება შემდეგი ნაწილებისგან:

1. მომხმარებელი
2. სერვის პროვაიდერები
3. გადახდის gateway

3.2.1 არქიტექტურის აღწერა

მომხმარებელი არის ადამიანი რომელიც იყენებს სისტემას ონლაინ გადახდებისთვის. მას შეუძლია გადაიხადოს სერვისების საფასური. რამდენიმე სერვისის ერთდროულად გადახდისათვის მომხმარებელს შეუძლია სერვისები დაამატოს კალათაში და შემდეგ ერთიანად გადაიხადოს. ასევე მომხმარებელს შეუძლია ისეთი სერვისები რომლებსაც ხშირად იხდის დაამატოს შაბლონებში, რათა აღარ მოუწიოს მათი ძებნა შემდეგი გადახდისას.

ასევე მომხმარებელს შეუძლია თავის ანგარიშში დაამატოს საკრედიტო ან სადებეტო ბარათები, რომლებსაც შეიყვანს გადახდის gateway-ს დაცულ გვერდზე და მერჩანტს დაუბრუნდება ბარათის ბოლო ოთხი ციფრი, ბარათის ვადის გასვლის თარიღი და ბარათის ტიპი (VISA, MasterCard და ა.შ), ასევე gateway-სთან შენახულ ბარათთან ბმის გასაკეთებლად gateway-ს სისტემაში შენახული ბარათის ID. ამ მხრივ მომხმარებელს აქვს კომფორტი ბარათების დამახსოვრებისა ისე, რომ მერჩანტის სისტემამ არ იცის ბარათის სრული ინფორმაცია, დამახსოვრებული ბარათით გადახდისას მერჩანტი აგზავნის მხოლოდ gateway-ში შენახული ბარათის ID-ს.

ყოველ მომხმარებლის ანგარიშს აქვს უნიკალური იდენტიფიკატორი, უნიკალური ელ. ფოსტის მისამართი და პაროლი, რომელიც მონაცემთა ბაზაში დაშიფრულია PBKDF2 შიფრაციის მეთოდით. ამ მხრივ მომხმარებლის ანგარიში დაცულია და ბაზის გატეხვის შემთხვევაში მომხმარებლის ინფორმაცია დაცულია.

სერვის პროვაიდერები თავაზობენ მომხმარებლებს სხვადასხვა სერვისს. მომხმარებელი ირჩევს სასურველ სერვისს, წერს თავის იდენტიფიკატორს სერვისის ფორმაში და მერჩანტი აგზავნის მოთხოვნას სერვის პროვაიდერთან მომხმარებლის ვალიდაციისთვის. თუ ასეთი მოხმარებელი არსებობს, მაშინ სერვის პროვაიდერმა შეიძლება დააბრუნოს სხვადასხვა ინფორმაცია მასზე, მაგალითად მისამართი, სახელი და გვარი, სერვისის ღირებულება და ა.შ. გამოგზავნილი ინფორმაცია იტვირთება მომხმარებელთან. ამის შემდეგ მომხმარებელს შეყავს სასურველი თანხა და ამატებს გადახდას კალათაში ან, იმ შემთხვევაში თუ მას აქვს დამატებული ბარათი, ირჩევს იმ ბარათს რომლითაც სურს გადახდა და იხდის მას. ამ დროს gateway-ს ეგზავნება მოთხოვნა გადახდის დასარეგისტრირებლად, მერჩანტის სერვერს უბრუნდება gateway-ს მიერ დაგენერირებული ლინკი, რომელზეც გადადის მომხმარებელი და შეყავს საბარათე ინფორმაცია, ან დამახსოვრებული ბარათით გადახდის შემთხვევაში მხოლოდ CVV და იხდის გადასახადს.

ამის შემდეგ მომხმარებელი ბრუდება მერჩანტის ვებ გვერდზე გადახდის სტატუსების გვერდზე და სადაც შეუძლია ნახოს გადახდის საბოლოო სტატუსი. მომხმარებლის მერჩანტის გვერდზე გადმოსვლისთანავე, მერჩანტი ამოწმებს გადახდის სტატუსს gateway-ში, თუ პასუხი დადებითია, აგზავნის მოთხოვნას სერვის პროვაიდერთან გადახდის შესრულებაზე და დადებითი პასუხის შემთხვევაში ურიცხავს მას ფულს. ამის შემდეგ მერჩანტი აგზავნის მოთხოვნას gateway-ში გადახდის დასრულების მოთხოვნას და მერჩანტს ერიცხება შესაბამისი თანხა. კოდის ნაწილი რომელიც უზრუნველყოფს გადახდის შესრულებას:

```
def process_payment(payment):
    payment = emm_check_payment(payment)
    if payment.emm_check_payment_code == 0 and payment.emm_check_payment_status_code == 5:
        # aq midis EPBS payment
        payment = epbs_payment(payment)
        if payment is not None and payment.epbs_payment_code == 0:
            # aq midis EMM confirm payment


            emm_confirm_payment(payment)


        elif payment is not None:
            with transaction.atomic():
                payment = cancel_payment(payment)
                payment.save()
            emm_cancel_payment(payment)
    elif payment is not None:
        with transaction.atomic():
            payment = cancel_payment(payment)
            payment.save()



def cancel_payment(payment):
    payment.status = 'DECLINED'
    payment.is_completed = True
    return payment



def emm_check_payment(payment):
    payload = {
        "emmTransactionId": payment.emm_transaction_id
```

```
    }
    start_time = datetime.datetime.now()

    while True:
        check_payment_response = emm_protocol(payload, 'check_payment')

        code = check_payment_response.get('code', None)
        if code is not None:
            with transaction.atomic():
                payment.emm_check_payment_status_code =
check_payment_response.get('paymentStatusCode', None)
                payment.emm_check_payment_code = check_payment_response['code']
                payment.save()

            return payment
        if (datetime.datetime.now() - start_time) > datetime.timedelta(1):
            return None


def epbs_payment(payment):
    start_time = datetime.datetime.now()
    amount = int(round(payment.total_amount * 100))
    payload = {
        "serviceId": 1000000,
        "terminalTransactionId": payment.id,
        "terminalTransactionTime": datetime.datetime.now().strftime('%Y-%m-%dT%H:%M:%S'),
        "amount": amount,
        "ccy": "GEL",
        "customerId": payment.client_no,
        "agentCommission": int(round(payment.client_fee * 100)),
```

```
    }
    with transaction.atomic():
        payment.epbs_payment_sent = True
        payment.save()
    while True:
        if (datetime.datetime.now() - start_time) > datetime.timedelta(1):
            return None
        response = epbs_protocol(payload, 'payment')
        epbs_response_code = response.get('code', None)
        if epbs_response_code is None:
            continue
        with transaction.atomic():
            payment.epbs_payment_code = epbs_response_code
            payment.date_epbs_payment_response = datetime.datetime.now()
            payment.save()
        if epbs_response_code == 0:
            with transaction.atomic():
                payment.epbs_transaction_id = str(response['epbsTransactionId'])
                payment.save()
            return payment
        elif epbs_response_code == 1 or epbs_response_code == 202 or epbs_response_code == 203 \
                or epbs_response_code == 999 or epbs_response_code == 998 or epbs_response_code == 106:
            continue
        else:
            return payment


def emm_confirm_payment(payment):
    start_time = datetime.datetime.now()
```

```
payload = {
    "emmTransactionId": payment.emm_transaction_id
}
with transaction.atomic():
    payment.emm_payment_confirm_sent = True
    payment.save()
while True:
    if (datetime.datetime.now() - start_time) > datetime.timedelta(1):
        return None
    response = emm_protocol(payload, 'confirm_payment')
    emm_confirm_payment_code = response.get('code', None)
    if emm_confirm_payment_code is None:
        continue
    if emm_confirm_payment_code == 0:
        with transaction.atomic():
            payment.emm_payment_confirm_code = emm_confirm_payment_code
            payment.status = 'FINISHED'
            payment.is_completed = True
            payment.date_emm_operation_end_status = datetime.datetime.now()
            payment.save()
        return payment
    elif emm_confirm_payment_code == 998:
        with transaction.atomic():
            payment.emm_payment_confirm_code = emm_confirm_payment_code
            payment.save()


    else:
        with transaction.atomic():
            payment.emm_payment_confirm_code = emm_confirm_payment_code
            payment.date_emm_operation_end_status = datetime.datetime.now()
```

```
            payment.save()
        return payment


def emm_cancel_payment(payment):
    start_time = datetime.datetime.now()
    payload = {
        "emmTransactionId": payment.emm_transaction_id
    }
    with transaction.atomic():
        payment.emm_cancel_sent = True
        payment.save()
    while True:
        if (datetime.datetime.now() - start_time) > datetime.timedelta(1):
            return None
        response = emm_protocol(payload, 'cancel_payment')
        emm_cancel_payment_response_code = response.get('code', None)
        if emm_cancel_payment_response_code is None:
            continue
        if emm_cancel_payment_response_code == 0:
            with transaction.atomic():
                payment.emm_cancel_code = emm_cancel_payment_response_code
                payment.date_emm_operation_end_status = datetime.datetime.now()
                payment.save()
            return payment
        elif emm_cancel_payment_response_code == 998:
            with transaction.atomic():
                payment.emm_cancel_code = emm_cancel_payment_response_code
                payment.save()
```

```
    else:
        with transaction.atomic():
            payment.emm_cancel_code = emm_cancel_payment_response_code
            payment.date_emm_operation_end_status = datetime.datetime.now()
            payment.save()
        return payment
```

ჩანართი 1: ტრანზაქციის გატარება

ეს ფუნქცია ყოველი გადახდისთვის ცალკე thread-ში ეშვება, არაფატალური პასუხის კოდის დაბრუნების შემთხვევაში აგზავნის მოთხოვნას თავიდან, ფატალურის შემთხვევაში გადახდას აუქმებს, ხოლო დადებითის შემთხვევაში ასრულებს მას.

3.3 სისტემებს შორის ურთიერთობის აღწერა

მერჩანტი სერვის პროვაიდერებთან და გადახდების gateway-სთან ურთიერთობს REST პროტოკოლის მეშვეობით. თითოეული მოთხოვნის გაგზავნა ხდება POST მეთოდით. ქსელის დაცულობას უზრუნველყოფს SSL სერტიფიკატები. რაც უზრუნველყოფს ინფორმაციის დაცვას man.in.the.middle შეტევისგან.

ყოველი მოთხოვნის ტანი იჰეშება HMAC-ის ალგორითმით, წინასწარ შეთანხმებული გასაღებით. ჰეში იგზავნება ყოველი მოთხოვნის ჰედერში, და მიმღები ამოწმებს ტანის ვალიდურობას ჰეშის თავიდან დათვლით და შედარებით. ეს გამორიცხავს ინფორმაციის დაზიანებას.
მოთხოვნის გაგზავნის პროტოკოლს აქვს შემდეგი სახე:

```
def epbs_protocol(request_body, request_type):
    try:
        logging.info('<-------EPBS REQUEST TYPE: %s---------->' % request_type)
        logging.info('<-------REQUEST BODY start ---------->')
        logging.info(request_body)
        logging.info('<-------REQUEST BODY end ---------->')
        if isinstance(request_body, dict):
            json_payload = json.dumps(request_body)
            hmac_signature = base64.b64encode(
                hmac.new(settings.EPBS_SECRET_KEY, json_payload.encode('utf-8'),
digestmod=hashlib.sha256).digest())
            headers = {
                'terminalId': settings.EPBS_TERMINAL_ID,
                'epbs-mac': hmac_signature,
                'Content-Type': 'application/json'
            }

            if request_type == 'get_info':
                response = epbs_get_info(json_payload, headers)
            elif request_type == 'payment':
                response = epbs_payment(json_payload, headers)
            elif request_type == 'payment_status':
                response = epbs_get_payments_status(json_payload, headers)
            elif request_type == 'balance':
                response = epbs_get_balance(json_payload, headers)
            logging.info('<-------RESPONSE start ---------->')
            logging.info(response)
            logging.info('<-------RESPONSE end ---------->')
```

```
        return response
```

ჩანართი 2: სერვის პროვაიდერის პროტოკოლი

პროტოკოლს გადაეცემა მოთხოვნის ტიპი და გასაგზავნი მონაცემები JSON-ის ფორმატით. მოთხოვნა იგზავნება POST მეთოდით, და თითოეული მოთხოვნის ტიპისთვის გამოიძახება შესაბამისი ფუნქცია, მაგალითად:

```
def epbs_get_info(json_payload, headers):
    try:
        with session() as s:
            response_json = s.post(settings.EPBS_GET_INFO_URL, json_payload, headers=headers)
            response = json.loads(response_json.content.decode("utf-8"))
            return response
    except ConnectionError as e:
        logging.error('<----------------------EPBS DOWN-------------------------->')
        logging.error('REQUEST_TYPE: GET_INFO')
        logging.error(e)
        logging.error('<----------------------EPBS DOWN-------------------------->')
        return {'error': 'EPBS DOWN'}
```

ჩანართი 3: მოთხოვნის გაგზავნის მეთოდი

ამ ფუნქციას გადაეცემა მოთხოვნის ტანი და ჰედერი. იგი აგზავნის მოთხოვნას შესაბამის მისამართზე და იღებს პასუხს და აბრუნებს მას, ConnectionError შეცდომის (exception-ის) შემთხვევაში იგი აბრუნებს შესაბამის პასუხს.

ყველა სხვა ფუნქცია რომელსაც აქვს კავშირი სხვა სისტემასთან მუშაობს მსგავსი პრონციპებით.

მოთხოვნის ტანისა და პასუხების აღწერა იხილეთ დანართში.

3.4 მომხმარებლის ინტერფეისი

ვინაიდან ეს პროექტი კომერციულია და ყავს კონკურენტები, ამიტომ საჭიროა არამხოლოდ სისტემის უსაფრთხოება, არამედ სისტემის მომხმარებლის მოთხოვნებზე მორგება. ამისათვის საიტი მომხმარებელს აძლევს საშუალებას დაამატოს ბარათები, შექმნას გადახდის შაბლონები, დაამატოს გადახდები კალათაში და გააკეთოს ავტომატური გადახდები.

3.4.1 ბარათები

ბარათის სისტემაში დამახსოვრება შეუძლია ავტორიზირებულ მომხმარებელს. მომხმარებელი გადადის გადახდის gateway-ს გვერდზე, სადაც დაცულ ფორმაში შეყავს ბარათის ნომერი და ვადის გასვლის თარიღი. ამის შემდეგ მომხმარებელს თავის ბარათების გვერდზე გამოუჩნდება ბარათის ტიპი (VISA, MASTERCARD და ა.შ), რომელსაც gateway ბარათის ნომრის მიხედვით ადგენს, ბარათის ბოლო ოთხი ციფრი. ასევე მომხმარებელს შეუძლია წაშალოს ბარათი სისტემიდან.

3.5 HMAC-ის ჰეშირების ალგორითმი

HMAC-ის ჰეშის დათვლა სისტემაში ხდება HMAC SHA 256 ფუნქციით, რომელსაც ჭირდება გასაღები და დასაჰეში მესიჯი პარამეტრებად. HMAC-ს დათვლისას ბევრი კრიპტოგრაფიული ჰეშირების ალგორითმი შეიძლება გამოვიყენოთ, მაგალითად MD5 ან SHA-1, მიღებულ ალგორითმს კი ერქმევა HMAC-MD5 ან HMAC-SHA1.

მესიჯი ამ სისტემის შემთხვევაში არის JSON ენკოდირებული UTF-8-ით. გზავნილის ჰეში ყოველთვის იგზავნება მოთხოვნის ჰედერში, რაც უზრუნველყოფს მონაცემთა სისრულეს.

მიმღებს აუცილებლად აქვს იგივე გასაღები და ამ გასაღებით დავიდან ჰეშავს მესიჯს, და თუ მიღებული და დათვლილი ჰეში ემთხვევა ერთმანეთს ესე იგი გზავნილი ავთენტურია.

HMAC-ის დაცულობა დამოკიდებულია გასაღების ზომაზე. ყველაზე გავრცელებული შეტევა ამ ალგორითმზე არის ძალისმიერი მეთოდი. მაგრამ HMAC-MD5-ს არ აქვს ისეთი სისუსტეები როგორიც მაგალითად MD5-ს, რომელიც ლექსიკონით საკმაოდ მარტივად ტყდება.

3.6 PCI-DSS

PCI-DSS არის მიღებული ინფორციის დაცვის სტანდარტი ისეთი ორგანიზაციებისთვის, რომლებიც იყენებენ ბრენდირებულ საკრედიტო ბარათებს დიდი საბარათე სქემებიდან. საბარათე სქემა არის გადახდების ქსელი დაკავშირებული საგადახდო ბარათებზე, როგორიცაა საკრედიტო და სადებეტო ბარათები, რომლის წევრიც შეუძლია გახდეს ბანკს ან სხვა შესაბამის ფინანსურ ინსტიტუტს. გაწევრიანების შემდეგ, წევრს შეუძლია გაცეს ან გაატაროს ბარათები, რომლებიც ჩართულია ამ ქსელში.

PCI სტანდარტები შექმნილია ბარათების ბრენდების მიერ და მას მართავს საგადახდო ბარათების ინდუსტრიის დაცვის სტანდარტების საბჭო. სტანდარტი შეიქმნა იმისათვის რომ დაეცვა ბარათის მფლობელის ფინანსური ინფორმაცია სხვადასხვა თაღლითობებისგან. სერტიფიკატს გაცემს კვალიფიციური უსაფრთხოების ინსპექტორი (QSA). ინსპექცია ტარდება ყოველწლიურად.

3.6.1 PCI-DSS მოთხოვნები

PCI-DSS მოიცავს თორმეტ მოთხოვნას, ექვს ლოგიკურად დაკავშირებულ ჯგუფად ორგანიზებულს. მათ საკონტროლო მიზნები ეწოდება.

PCI-DSS-ის ყოველი ვერსია ამ თორმეტ მოთხოვნას სხვადასხვანაირად ყოფდა ქვემოთხოვნებად, მაგრამ ეს თორმეტი მოთხოვნა ყოველთვის შეუცვლელი რჩებოდა.

მაგალითად მეექვსე მოთხოვნის მეექვსე ქვემოთხოვნა კოდის განხილვა და ფაირვოლების შემოწმებაა, მეთერთმეტე მოთხოვნის მესამე ქვემოთხოვნა კი - შეღწევადობის ტესტი (Penetration Testing).

თავი 4: დასკვნა

ამ თეზისში, ჩვენ განვიხილეთ ამჟამად არსებული გადახდის სისტემები, ძირითადად VISA და MasterCard. ვისაუბრეთ მათ მექანიზმებზე, რომლებიც თითქმის ერთნაირია, გარდა UCAF-ის გენერაციისა, რომელიც იცავს მომხმარებლის სადებეტო ან საკრედიტო ბარათის ინფორმაციას ინტერნეტში გამოყენებისას. ამას გარდა განვიხილეთ ასევე SET და მისი როლი გადახდის სისტემების ჩამოყალიბებაში და კრიპტოგრაფიაში.

ახლანდელ გადახდის სისტემებში, მომხმარებლის საგადახდო ინფორმაცია იგზავნება gateway-ში მერჩანტის მიერ. ეს გადახდას სისტემას ხდის დაუცველს, რამაც შეიძლება გამოიწვიოს ინფორმაციის გაჟონვა, მონაცემების მოპარვა და არასანქცირებული ტრანზაქციები. მომხმარებლის ფინანსური ინფორმაციის დაცვისათვის ჩამოვაყალიბეთ ახალი მიდგომა ონლაინ გადახდების სისტემებისთვის, რომელშიც მომხმარებლის საბარათე ინფორმაცია მომხმარებელს პირდაპირ შეყავს გადახდების gateway-ში. ეს მეთოდი ნებადართულია VISA, MasterCard და სხვა ბარათების მწარმოებლებისგან და მერჩანტს აძლევს საშუალებას დაამტკიცოს ტრანზაქციის ლეგიტიმურობა ეჭვის გამოთქმის შემთხვევაში.

ნაშრომში აღწერილი ახალი მიდგომა ელექტრონული გადახდების სფეროში, რეალიზებულია და წარმოადგენს შპს იუპიესის საკუთრებას. პროგრამას ჩაუტარდა შეღწევადობის ტესტები, რომლებიც წარმატებით გაიარა. იგი დაახლოებით 500-1000მდე ტრანზაქციას ატარებს დღეში. მისი ნახვა შეგიძლიათ მისამართზე http://gadaixade.com.ge.

ზემოაღნიშნულიდან გამომდინარე ვთვლით რომ ეს სისტემა უსაფრთხოა და იცავს მომხმარებლის საბარათე და პირად ინფორმაციას ჰაკერებისგან.

დანართი

მომხმარებლის ნომრის შემოწმება

ჰედერი

```
{
'terminalId': '13',
'epbs-mac': b'P4ZV2RRQFNmT8wBYYoC+swMdJgk2q4i9uXsYNMaV+rk=',
 'Content-Type': 'application/json'
}
```

მოთხოვნა

```
{
        'serviceId': 19,
        'customerId': ‘1233455464’
}
```

პასუხი

```
{
'terminalId': 13,
'customerId': '599508672',
'params': {
        'OWNER': ' ',
         'Description':
         'Phone Exists'
},
'code': 0,
'message': 'OK',
'serviceId': 19
}
```

გადახდის რეგისტრაცია გადახდების gateway-ში

ჰედერი

```
{
'terminalId': '13',
'epbs-mac': b'P4ZV2RRQFNmT8wBYYoC+swMdJgk2q4i9uXsYNMaV+rk=',
'Content-Type': 'application/json'
}
```

მოთხოვნა

```
{
“transactions”: [
{
“serviceGroupId”:”1000000”,
“transactionId”:12345677,
“amount”: 100
}
],
“cardId”: “126”,
“currency”: “GEL”
“backURL”: “http://merchant.ge?transactionId=5“
}
```

პასუხი

```
{
“code”: 0,
“message”: “OK”,
“transactions”: [
{
“transactionId”:12345677,
“emmTransactionId”: 1
“transactionTime”: “2016-07-27 11:12:08”,
“merchantFee”: 2,
}
]
“redirectId”: 17,
}
```

გადახდის სტატუსის შემოწმება gateway-ში

მოთხოვნა

```
{
“emmTransactionId”:”1”
}
```

პასუხი

```
{
“code”: 0,
“message”: “OK”,
“paymentStatusCode”: 4,
“paymentStatusMessage”: “bank confirmed”

}
```

სერვის პროვაიდერთან გადახდის მოთხოვნა

მოთხოვნა


```
{
"serviceId":1000000,
"terminalTransactionId":456,
"terminalTransactionTime":"2015-
04-14T18:25:43",
"amount":5000,
"agentCommission":50,
"ccy":"GEL",
"customerId":"abc123",
"params":{
"param1":"val1",
"param2":"val2"
}
}
```


პასუხი


```
{
"code": 0,
"message": "OK",
"serviceId": 1000000,
"terminalId": 13,
"terminalTransactionId": "456",
"epbsTransactionId": 165475,
"time": "2015-12-28T13:51:01",
"params": {
"comment": "OK"
}
}
```

გადახდის დასრულების მოთხოვნა gateway-სთან

მოთხოვნა

{
“emmTransactionId”:”1”
{

პასუხი

{
“code”: 0,
“message”: “OK”
}

გადახდის გაუქმება gateway-სთან

მოთხოვნა

{
“emmTransactionId”:”1”
}

პასუხი

{
“code”: 0,
“message”: “OK”
}

ბიბლიოგრაფია

3D-Secure, verified by Visa and SecureCode. (n.d.). Retrieved from http://www.mastercard.com/gateway/implementation_guides/3D-Secure.html

ecommerce. (n.d.). Retrieved from investopedia: http://www.investopedia.com/terms/e/ecommerce.asp

http://www.bestpaymentgateways.com/payment-processing/what-is-a-payment-gateway/. (n.d.).

http://www.postgresql.org/. (n.d.).

https://www.djangoproject.com/. (n.d.).

Payment Card Industry Data Security Standard. (n.d.). Retrieved from pcisecuritystandards: https://www.pcisecuritystandards.org/pci_security/

Secure Electronic Transaction. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Secure_Electronic_Transaction

Transport Layer Security. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Transport_Layer_Security