DOI 10.15276/imms.v10.no 3-4.197
УДК 004.056.53

Informatics and Mathematical Methods in Simulation
Vol. 10 (2020), No. 3-4, pp. 197-207

РОЗРОБКА СИСТЕМИ БЕЗПЕКИ КЛІЄНТ-СЕРВЕРНОГО ЗАСТОСУНКУ НА БАЗІ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID

А.Ю. Судаков, А.В. Соколов

Одеський національний політехнічний університет,
Україна, Одеса, 65044, пр. Шевченка, 1, e-mail: radiosquid@gmail.com

В сучасному світі неможливо обійтися без використання різного роду систем комунікації. Вони стали невід'ємною частиною нашого життя. Майже кожна сучасна людина користується тою чи іншою системою обміну інформації з іншими людьми. Частіше всього для обміну інформацією використовуються мобільні платформи. В переважній більшості це такі платформи як Android, iOS та Windows Phone. Інформація, яка передається від одного користувача до іншого, часто може бути конфіденційною або з обмеженим доступом. Для цього мають використовуватися відповідні методи захисту інформації. Сьогодні увага дослідників інформаційної безпеки зосереджена на покращенні роботи криптографічних алгоритмів та примітивів, на яких вони засновані. Тим не менш, питання побудови комплексних систем захисту в системах обміну миттєвими повідомленнями залишаються розглянутими недостатньо. Дана стаття присвячена розробці комплексної системи захисту застосунку для обміну миттєвими повідомленнями. Будь який застосунок для роботи з миттєвими повідомленнями складається з трьох основних частин. Першою складовою частиною є програмний код, виконуваний на мобільній платформі. Другою частиною є канал зв'язку. Останньою частиною є програмний код, виконуваний на стороні серверу. В даній статті розглянуто кожну з основних частин для побудови комплексної системи захисту застосунку для обміну миттєвими повідомленнями. Обґрунтовано вибір тих чи інших компонентів для побудови системи захисту, приведено схеми роботи програмного коду виконуваного на мобільній платформі Android, обґрунтовано вибір форм передачі користувацької інформації через канал зв'язку. Надано рекомендації по вибору каналу зв'язку та забезпеченню його безпеки. Також приведено схеми роботи серверної сторони застосунку. Обґрунтовано вибір тих чи інших хеш-функцій, функцій перевірок вхідної інформації, підходів до реалізації системи захисту, системи зберігання і обробки користувацької інформації на стороні серверу. У застосунку також продемонстровано деякі недоліки поточної версії мобільної платформи Android з точки зору інформаційної безпеки. Показано, як зловмисники можуть використати ці недоліки, та надано поради по уникненню їх експлуатації зловмисником. Результати роботи можуть бути використані для захисту інформації у подібних застосунках.

Ключові слова: системи обміну миттєвими повідомленнями, захист інформації, мобільний застосунок.

Вступ і постановка задачі

З кожним днем кількість користувачів мобільних пристроїв зростає. Більшість пристроїв, які використовують користувачі, — це смартфони та планшети на базі операційної системи Android та iOS. Нові способи взаємодії з пристроєм та його мобільність — це нові фактори, через які з'являються нові проблеми з інформаційною безпекою користувача. При розробці мобільного застосунку необхідно брати до уваги те, що дані, які використовуються застосунком, можуть представляти певну цінність для зловмисників. Те, наскільки цінною буде інформація, залежить від багатьох факторів, проте навіть найпростіша інформація, наприклад, особистий пароль від облікового запису користувача потребує опрацювання його за допомогою системи захисту. На сьогоднішній день мобільні застосунки зайняли місце у всіх сферах електронних послуг, включаючи фінансові, банківські операції, зберігання і передачу

особистих даних, що робить завдання підвищення їх безпеки особливо важливим [1]. Сьогодні в літературі чимало уваги дослідники приділяють основним блокам для побудови систем захисту інформації — криптографічним алгоритмам, хеш-функціям, тощо [2, 3]. Однак питання побудови комплексних систем захисту інформації для конкретних програмних продуктів, зокрема, систем обміну миттєвими повідомленнями освітлені в літературі недостатньо. Існуючі представлені системи [4…6] є специфічними і при цьому деталі технічної реалізації їх систем захисту інформації на клієнтській та серверній частині розкрито не повністю. При цьому системи безпеки реально працюючих програмних продуктів зазвичай, в більшій своїй частині, залишаються комерційною таємницею компаній, що розроблюють та підтримують свої програмні продукти, та є недоступними для світової наукової спільноти. У цій роботі розглядається питання побудови комплексної системи безпеки реального застосунку для обміну миттєвими повідомленнями на базі операційної системи Android.

Метою роботи є розробка системи захисту інформації застосунку для обміну миттєвими повідомленнями.

Основна частина

Більшість сучасних застосунків є клієнт-серверними, тобто частина логіки застосунку, а також частина інформації користувача зберігається та обробляється на стороні серверу [7]. Для отримання якісної моделі системи захисту клієнт-серверного Android застосунку, а також для кращого її розуміння розділимо її на три основні частини. Кожна частина системи безпеки буде розглянута окремо для кращого її розуміння та більш детального опису. Розглянемо три основні частини захисту клієнт-серверного застосунку.

Перша частина, це власне сам застосунок, точніше його код, який буде виконуватися на пристрої користувача, всі маніпуляції з інформацією та її захистом на пристрої. Друга частина — канал передачі інформації між пристроєм та сервером. Остання частина системи безпеки — серверна сторона застосунку — код, який обробляє запити та надсилає відповіді на них, а також сам сервер і база даних, в якій зберігається інформація користувачів. Основні частини клієнт-серверного застосунку показані на рис. 1.

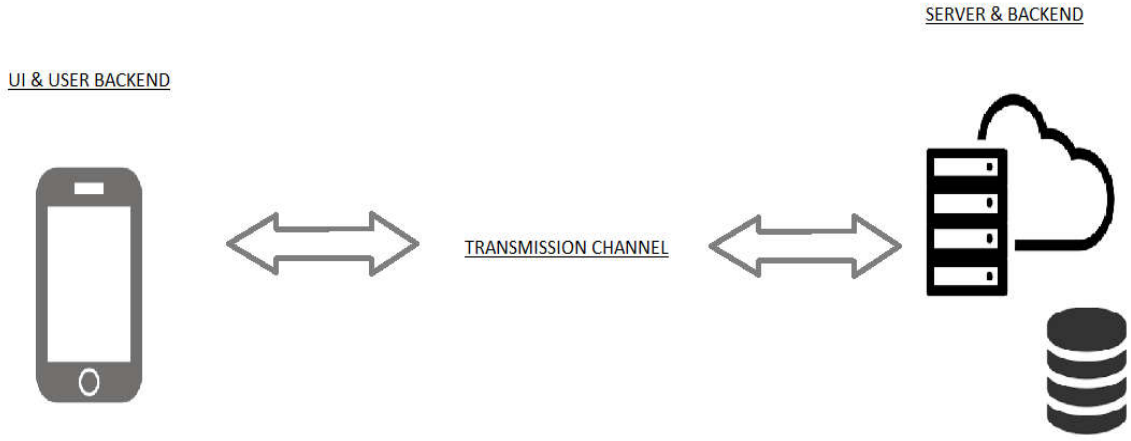

Рис. 1. Основні частини клієнт-серверного застосунку

Далі ми окремо розглядаємо аспекти системи захисту інформації у кожній частині клієнт-серверного застосунку.

1. Захист на стороні клієнта

Розглянемо більш детально клієнтську частину системи захисту клієнт-серверного застосунку. Вона складається з двох основних частин. Перша частина — те, що бачить користувач на дисплеї свого екрану — UI (UserInterface), друга частина — код, який виконується на пристрої — backend, тобто сторона застосунку, яка прихована від користувача.

Кожен користувач клієнт-серверного застосунку має бути авторизованим [8]. Частіше всього в якості ідентифікатора в сучасних застосунках виступає обліковий запис користувача певного пристрою. Проте є застосунки, які надають доступ до інформації тільки після вводу певних даних для авторизації і знищують всю інформацію після виходу з облікового запису. Саме такими є два основних підходи до створення систем захисту. Перевагою першого є те, що користувачу достатньо знати пароль та логін і, маючи ці дані, він в будь-який час може отримати доступ до своєї інформації. З точки зору користувача, така система є доволі комфортною. Проте в ній є певні недоліки. Наприклад, небажано, щоб хтось інший міг отримати доступ до інформації користувача. Уявимо ситуацію, коли така інформація не видаляється, а зберігається в певному місці. Дані для входу в обліковий запис можуть бути викрадені і зловмисник отримає доступ до інформації користувача. Проте, якби така інформація була видалена відразу після виходу із облікового запису, навіть отримавши доступ до нього, було б неможливим отримання такої інформації, адже вона була видалена. В свою чергу, недоліком другого підходу є те, що користувачеві необхідно кожного разу вводити одну і ту ж саму інформацію, що викликає дискомфорт у користувачів. Також без збереження даних користувачів складно відрізнити одного користувача від іншого, або і зовсім неможливо. Це несе ризик того, що один користувач зможе використовувати ресурси, розподілені на інших користувачів і як наслідок, перенавантажувати систему.

Для системи обміну миттєвими повідомленнями найкращим вибором системи авторизації є перший підхід. Даними для отримання доступу до інформації виступає логін та пароль. Розглянемо систему захисту застосунку на етапі входу в обліковий запис.

Тут слід наголосити, що вся інформація в застосунку може бути змінена користувачем. Завдання захисту інформації в застосунку в частині UI & USER BACKEND — захистити персональну інформацію користувача від інших застосунків та від витоку на сторонні сервісу, запобігти передачі некоректної інформації на сервер, для запобігання виникнення помилок в застосунку та позитивного досвіду користування у користувача. Слід розуміти, що вся інформація, яку користувач передає на сервер, повинна перевірятися на сервері. Інформація, яку застосунок передає на сервер може бути змінена як системою так і іншими застосунками, тому в самому застосунку перевірка інформації більше потрібна для зручності користувача.

Ще одним важливим моментом є безпека критично важливої інформації, яка зберігається на пристрої. До такої інформації можна віднести паролі, ключі, персональну інформацію.

В табл. 1, ми наводимо перелік основних вразливостей критично важливої інформації безпосередньо в самому застосунку, характеризуючи їх рівень небезпеки, а також наводимо методи захисту інформації, які було імплементовано у розробленій системі для протидії наведеній вразливості.

Почнемо з запуску застосунку. Під час запуску, система повинна перевірити, чи пройшов користувач процедуру входу в обліковий запис, а також перевіряти статус входу під час кожної дії користувача. Для цього використовується функція *is Authed ()*. Вона викликається застосунком при кожній дії користувача. Така функція звертається до внутрішньої бази даних для перевірки наявності даних для авторизації користувача на сервері. Якщо така інформація наявна, функція повертає true, інакше — false.

Якщо функція повернула true — виконується запуск activity, яка містить основну частину застосунку, проте, якщо функція повернула false — виконується запуск activity, яка містить поля для вводу інформації для авторизації на сервері. В нашому випадку, це поля *e-mail* та *password*.

Далі, інформація відправляється на сервер через протокол https. Коли відповідь від сервера отримана, створюється таблиця в локальній БД (Базі Даних), в якій зберігається хеш-значення паролю та e-mail, які будуть відправлятися на сервер з кожним запитом. В нашому випадку використовується хеш-функція sha-256, яка є досить стійкою на даний момент. Не рекомендується використовувати хеш-функції, довжина дайджесту яких менше ніж 256 символів. Такою застарілою функцією є md5. В найкращому випадку рекомендується використовувати хеш-функцію з третього покоління sha. Такою функцією є sha-512. Може здатися дивним те, що хеш-функція з довжиною дайджесту в 512 символів працює швидше ніж та ж сама функція, з довжиною дайджесту в 256 символів. Проте це доведений факт. В нашому застосунку використовується sha-256 для кращого сприйняття, адже вона давно відома і широко використовується.

Таблиця 1.

Перелік основних вразливостей критично важливої інформації безпосередньо в самому застосунку

№	Вразливість	Рівень ризику	Використане рішення захисту
1	Використання незахищених локальних сховищ	Дуже висока	Зберігати критично важливу інформацію тільки в захищених сховищах платформи Android
2	Зберігання інформації в захищених сховищах, але у відкритому вигляді	Середня	Критично важлива інформація не повинна використовуватися в застосунку без додаткового шифрування. Як тільки потреба в такій інформації відпала — вона негайно повинна бути або зашифрована, або знищена
3	Зберігання критично важливої інформації в коді	Висока	Не зберігати критично важливу інформацію в коді або ресурсах застосунку
4	Використання асиметричного алгоритму з приватним ключем, відомим для сервера	Залежить від ступеня захищеності сервера	Не зберігати приватний ключ на сервері, він повинен бути доступний тільки користувачу
5	Використання самописних алгоритмів шифрування та захисту	Середня	Слід підбирати відповідний алгоритм тільки з налагоджених і актуальних загальновідомих криптографічних алгоритмів
6	Передача критично важливої інформації у відкритому вигляді	Середня	Будь-яка інформація, перед виходом за межі застосунку повинна бути зашифрована
7	Ігнорування пристроїв з правами суперкористувача	Середня	Перевіряти на права суперкористувача систему і в разі їх наявності інформувати користувача і, по можливості, не продовжувати роботу застосунку без інформування користувача

Пароль обов'язково хешується ще до відправки на сервер. По каналу зв'язку передається тільки хеш-значення цього паролю. В локальній БД також зберігається тільки хеш-значення пароля. Як було вказано раніше в табл. 1 — не можна зберігати критично важливу інформацію в захищених сховищах у відкритому вигляді. При цьому e-mail зберігається у відкритому вигляді, оскільки в даному контексті він не відноситься до критично важливої інформації.

Коли застосунок отримав інформацію про успішний вхід до облікового запису і створив таблиці з даними користувача в локальній БД, ініціюється запуск основної activity в якій реалізовано основну частину логіки застосунку.

Схожим чином реалізується і «відновлення паролю». Після натискання на «відновити пароль» завантажується activity в якій є поля для вводу логіна та кнопка підтвердження дії. Перевірка логіна (e-mail) працює таким же чином як і при «вході» в застосунок. Для кращого розуміння на рис. 2 представлена схема логіки UI &BACKEND застосунку.

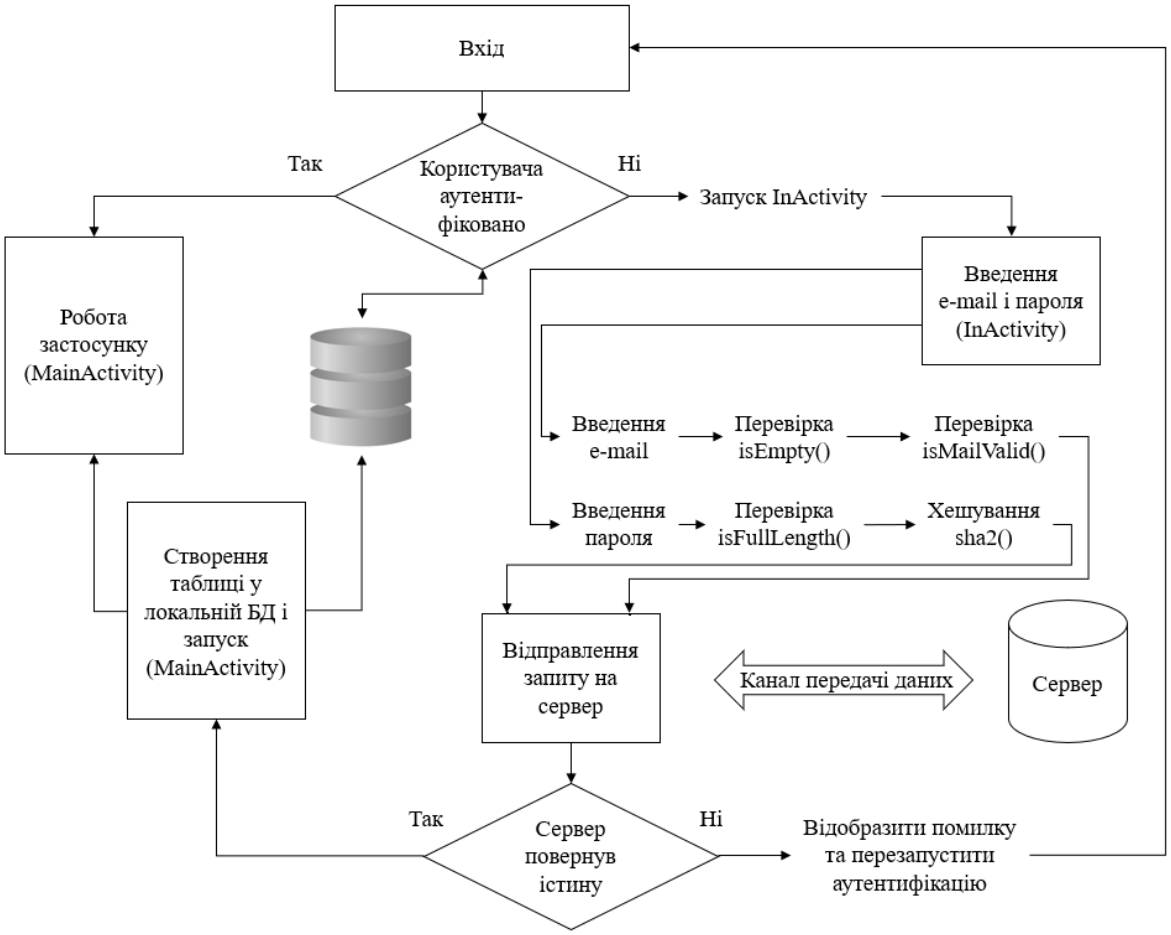

Рис. 2. Схема логіки входу та перетворення даних в UI &BACKEND частині

2. Захист інформації при її передаванні на сервер

Перейдемо до частини «TRANSMISSION CHANNEL». В застосунку використовується сучасний захищений протокол передачі https, який забезпечує безпеку інформації при її передаванні. Також для передачі даних на сервер використовується неявна передача методом POST замість стандартного GET, адже критично важливу інформацію слід передавати неявним методом. Інформація передається в вигляді «key –>value» без кодування JSON. Файли передаються як multipart/form-data.

3. Захист інформації на серверній частині

Перейдемо до останньої частини — Server & BACKEND. Розглянемо, як сервер опрацьовує запити та відповідає на них. Для кращого розуміння розглянемо можливість реєстрації нових користувачів у застосунку. Реєстрація на пристрої працює схожим способом як і «Вхід», дані обробляються тими ж функціями і так само відправляються на сервер. Різниця між «входом» і «реєстрацією» більше всього відчувається на стороні Server & BACKEND. Розглянемо більш детально.

Коли дані користувача потрапляють до серверу їх потрібно обробити. Не можна забувати, що дані могли прийти не від користувача і що в них може міститися шкідливий код або щось інше. Основні атаки на сервер — це DDoS, SQL-injection, XSS та OWASP [9]. Оскільки розглядається саме сторона захисту в коді, який опрацьовує запити (файл, який виконується після перевірки на безпеку запиту системою захисту сервера) то найактуальнішою, в даному контексті, атакою є атака типу injection та інші способи викликати помилку на сервері.

Для найкращого захисту від різних атак на сервері потрібно вимкнути відображення помилок користувачу і відображати мінімум інформації. Для цього, у випадку невірних параметрів або їх відсутності просто потрібно повернути помилку. Наприклад: `{"err":"wrongparam(s)"}`. Такий результат дає мало інформації і зловмисник не має можливості перевірити чи спрацював його метод, чи ні. Наступним кроком захисту є перевірка наявності всіх необхідних параметрів. Якщо кількість або імена параметрів не збігаються з потрібними — повертаємо помилку. Щоб перевірити наявність параметрів частіше всього використовують php код, наприклад, такий:

```
if(isset($_POST['app']) &&isset($_POST['mail']) &&
isset($_POST['userName']){doSomething();}else{exitWithStatus("wrong
param(s)");}
```

Такий спосіб перевірки не дасть зловмиснику вибрати інший метод передачі даних або додати свої параметри. В цьому випадку зловмиснику залишається тільки намагатися підмінити значення цих параметрів. Щоб захиститися від цього використовуються наступні функції:

– trim(data) — зрізає пробіли на початку і в кінці рядка, запобігає внесеннювідправки пустого поля;

– stripslashes(data) — видаляє екрануючіслеши, які можуть привести до виконання переданого коду;

– htmlspecialchars(data) — перетворює спеціальні символи в html сутності, таким чином можна запобігти будь-яким спробам виконати код.

Таким чином перевіряються всі параметри. Далі потрібно перевірити, що передано і яка в цього значення довжина. Це робиться за допомогою наступного коду:

```
if(strlen($data) <someLength){
exitWithStatus("err: wrongparam(s)");}
```

Так як ми розглядаємо процедуру реєстрації, після вказаних вище перевірок, необхідно перевірити, чи наданий e-mail вже зареєстровано. Якщо такий e-mail вже зареєстрований, але не підтверджений, потрібно видалити всю інформацію пов'язану з ним і надіслати код підтвердження на e-mail. Якщо користувач все ж таки не зареєстрований, потрібно згенерувати пароль, який буде зберігатись в БД. Перед тим, як вносити хеш-значення пароля в БД необхідно згенерувати для нього модифікатор входу хеш-функції. Модифікатор входу хеш-функції — певний рядок інформації, який передається хеш-функції разом з вхідним масивом інформації для вираховування хеш-значення. Модифікатор входу хеш-функції використовується для ускладнення визначення прообразу хеш-функції методом перебору по словнику можливих вхідних значень, включаючи атаки з використанням райдужних таблиць. Головною функцією є приховання факту використання однакових вхідних даних, наприклад, — паролів. Тобто, однакові паролі, після обробки хеш-функцією з додаванням модифікатора входу хеш-функції будуть виглядати по різному.

Модифікатор входу хеш-функції повинен бути максимально випадковим. Тому нами був обраний наступний підхід до формування модифікатора входу хеш-функції: для її генерації використовується хеш-значення пароля від користувача, до нього додається хеш-значення часу запиту з точністю до мілісекунди, ім'я користувача, а також хеш-значення e-mail адреси, яка була використана при реєстрації.

Отриманий рядок обробляється алгоритмом sha2(data) [10]. Отриманий модифікатор входу хеш-функції змішується з хеш-значенням пароля користувача і вноситься в БД. Тобто в розробленій нами системі у БД зберігається модифікатор входу хеш-функції і хеш-значення пароля користувача, яке було змішане з модифікатором входу хеш-функції. Таким чином, навіть однакові паролі в БД виглядають абсолютно по різному.

Одним з ризиків підсистеми реєстрації у системах парольного захисту інформації є атака за допомогою множинної реєстрації фейкових користувачів. Для цього нами були прийняті наступні засоби захисту: для підтвердження реєстрації генерується 4-х значний код, який відправляється на e-mail адресу, яка вказана при реєстрації. Для його генерації використовується функція «стандартний генератор псевдовипадкових чисел» за рівномірним законом у діапазоні від 1000 до 9999, після чого код зберігається в БД. Разом з ним генерується timeHash, який є унікальним ідентифікатором коду. Також, в БД зберігається кількість спроб ввести код. Якщо кількість спроб більше 3, код генерується знову, відправляється на e-mail.

Після того як код буде підтверджено, вноситься запис в БД про те, що користувач підтвердив e-mail і йому може бути надано доступ до інформації та можливостей. Нижче показана схема взаємодії даних від користувача та сервера (рис. 3).

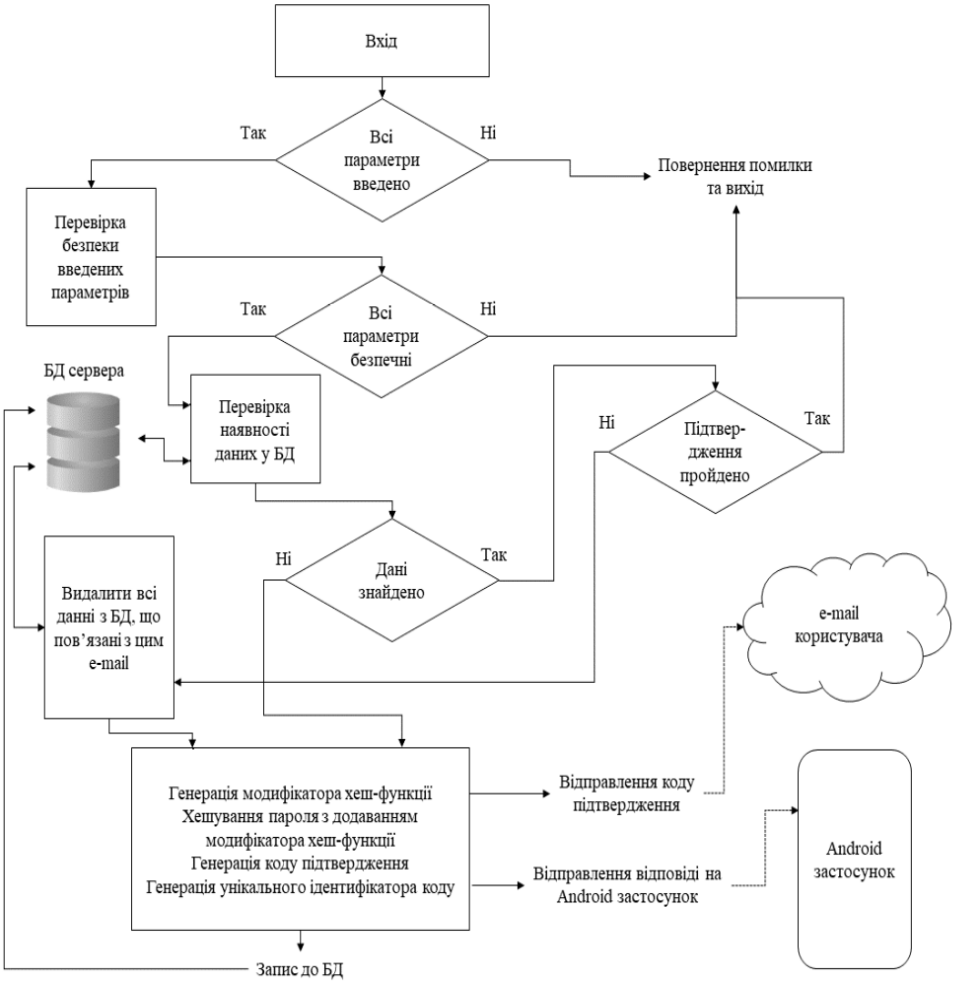

Рис. 3. Схема серверної логіки можливості «Реєстрація»

Після відправки коду підтвердження, сервер повідомляє про це застосунок. Застосунок, отримавши таку відповідь, завантажує activity для передачі коду підтвердження на сервер із застосунку. Система перевірки інформації та її захист виконується подібно до можливості «Реєстрація», як на сервері так і в самому застосунку.

Зовнішній вигляд застосунку представлено на рис.4.

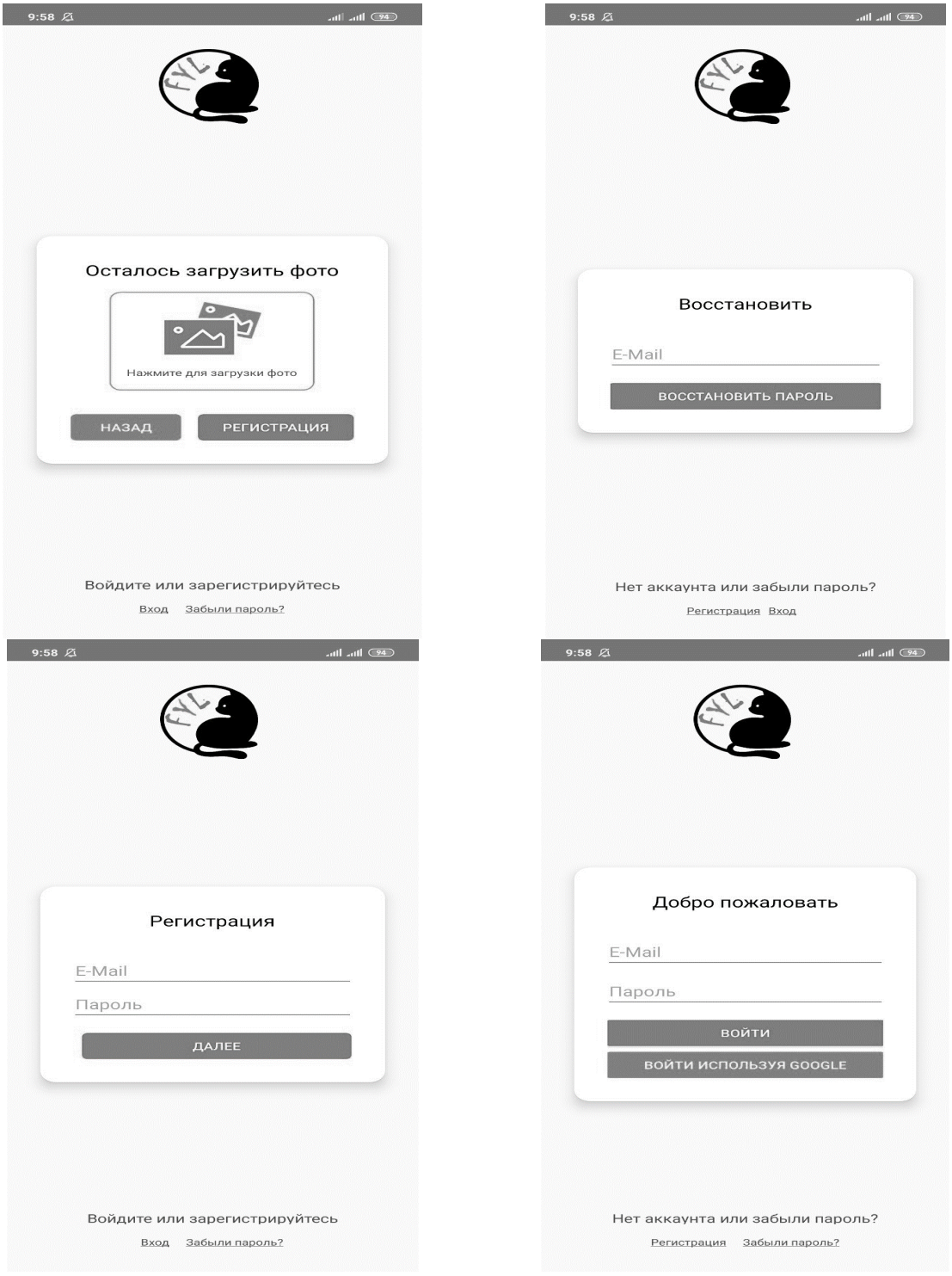

Рис. 4. Зовнішній вигляд розробленого застосунку

Код застосунку можна переглянути на GitHub за наступним посиланням [11].

Висновки

Відзначимо основні результати проведеного синтезу системи безпеки клієнт-серверного застосунку для обміну миттєвими повідомленнями:

1. При розробці клієнт-серверного застосунку мають окремо розглядатися заходи захисту інформації у трьох його основних частинах: на стороні клієнта, у каналі передачі даних та на серверній стороні. При цьому, зважаючи на існування надійного на цей час протоколу https, найголовнішими завданнями є розробка системи захисту у клієнтській та серверній частині.

2. В клієнтській частині застосунку було розроблено систему захисту, яка перевіряє всі введені користувачем дані. Перевіряється довжина введеної інформації в поля вводу, її формат, виконується перевірка на пустоту та допустимі символи. Також вся інформація в клієнтській частині зберігається в захищеному внутрішньому сховищі платформи у вигляді бази даних. Внутрішнє захищене сховище використовується для зберігання користувацьких фото. Критичні користувацькі дані, такі як пароль, не зберігаються у відкритому вигляді. Дані користувача про його місцезнаходження є менш критичними, тому зберігаються у відкритому вигляді, проте всього на протязі 24-х годин.

3. Серверна частина застосунку є більш захищеною ніж клієнтська сторона. Це зумовлено тим, що на серверній стороні обслуговуються запити від багатьох користувачів і перевірка на стороні сервера повинна бути якомога більш жорсткою та надійною. На стороні сервера, крім тих же перевірок, що і на клієнтській стороні, також виконується перевірка на кількість вхідних параметрів, екранується вхідна інформація для запобігання різного роду ін’єкцій в передану інформацію. В базі даних зберігаються певні мітки про користувачів, завдяки яким можна надати чи не надати доступ до ресурсу. Також зберігається кількість спроб вводу коду підтвердження та виконується ще ряд додаткових перевірок користувацької інформації. При виникненні помилки обробки запиту, сервер повертає відповідь з мінімальною кількістю інформації для запобігання вивчення системи безпеки на серверній стороні.

Було запропоновано нові підходи до реалізації:

— вводу та зберігання пароля в локальному сховищі;

— передачі пароля через канал зв’язку;

— генерації унікального хеш-значення з використанням унікальної користувацької інформації;

— перезавантаження сервісів після їх знищення системою;

— використання пасивних менеджерів для отримання геолокації з інших застосунків;

— пошукових запитів, які відображають збіги в користувацькому місцезнаходженні по даним геолокації та часу;

— зберігання користувацької інформації обмежений час, як в самому застосунку так і на сервері.

Таким чином, була розроблена система безпеки клієнт-серверного застосунку, яка показує як потрібно перетворювати, зберігати, передавати та обробляти інформацію користувача на пристрої, в каналі зв’язку та на сервері. Показано основні методи захисту від вводу некоректної інформації на пристрої, а також коректної обробки такої інформації на сервері. Розроблено конкретні схеми комунікації Androidзастосунку з сервером.

Список літератури

1. Внукова З.А., Буханцов А.Д., Путивцева Н.П., Кулешов С.И. Оценка безопасности систем мгновенного обмела сообщениями методом анализа иерархий. *Научные ведомости. Серия Экономика. Информатика*. 2016. №23(244). Вып. 40. С. 165–169.
2. Zhdanov O.N., Sokolov A.V. Block symmetric cryptographic algorithm based on principles of variable block length and many-valued logic. *Far East Journal of Electronics and Communications*. 2016. V.16, No. 3. P. 573–589.
3. Deng S., Li Y., Xiao D. Analysis and improvement of a chaos-based Hash function construction. *Communications in Nonlinear Science and Numerical Simulation*, 2010. V. 15, No. 5. P. 1338–1347.
4. Yang C.H., Kuo T.Y., Ahn T.N. et al. Design and Implementation of a Secure Instant Messaging Service based on Elliptic-Curve Cryptography. *International Symposium on Computer Science and Computational Technology*, 2008, 20–22 December 2008, Shanghai, China. P. 31–38.
5. Z. Wang, Z. Ma, S. Luo et. al. Enhanced Instant Message Security and Privacy Protection Schemed or Mobile Social Network Systems. *IEEE Access*. 2018. V.6. P. 13706–13715.
6. Bonesa E., Hasvolda P., Henriksena E. et al. Risk analysis of information security in a mobile in stant messaging and presence system for healthcare. *International Journal of Medical Informatics*. 2006. No. 76(9). P.677–687.
7. Чернобровкин С.В., Мялкин М.П. Архитектура клиент-серверного взаимодействия с мобильным приложением. *Молодежный научно-технический вестник*. 2015. №. 8.С. 25.
8. Шибанов С.В., Карпушин Д.А. Сравнительный аналіз современных методов аутентификации пользователя. *Информационное обеспечение систем*. 2015. №1(6). С. 33–37.
9. Al-Khurafi O.B., Al-Ahmad M.A. Survey of web application vulnerability attacks. *4th International Conference on Advanced Computer Science Applications and Technologies*.2015. P. 154–158.
10. Mendel F., Nad T., Schaffer M. Finding SHA-2 characteristics: searching through a minefield of contradictions. *International Conference on the Theory and Application of Cryptology and Information Security*., Berlin, Heidelberg : Springer, 2011. P. 288–307.
11. Sudakov A.Yu. Код застосунку FYL URL: https://github.com/ArtemSudakov/FylApp.

DEVELOPMENT OF CLIENT-SERVER APPLICATION SECURITY SYSTEM BASED ON ANDROID OPERATING SYSTEM

A.Yu. Sudakov, A.V. Sokolov

Odesa National Polytechnic University,
Ukraine, Odessa, 65044, 1 Shevchenko Ave. radiosquid@gmail.com

In today's world it is impossible to do without the use of various communication systems. They are now the integral part of our lives. Almost every modern person uses one or another system of information exchange with other people. Mobile platforms are most often used for information exchange. The vast majority are platforms such as Android, iOS and Windows Phone. Information transmitted from one user to another can often be confidential or restricted. So, the appropriate methods of information protection should be used for this purpose. Today, information security researchers focus on improving the performance of cryptographic algorithms and the primitives on which they are based. Nevertheless, the issues of building of integrated security systems for instant messaging systems remain insufficiently considered. This paper is devoted to the development of a comprehensive application protection system for instant messaging. Any IM application consists of three main parts. The first component is the program code executed on a mobile platform. The second part is the communication channel. The last part is the program code executed on the server side. This paper discusses each of the main parts for building a comprehensive application protection system for instant messaging. The choice of certain components for building a security system is substantiated, the schemes of operation of the program code executed on the Android mobile platform are given, the choice of forms of transmission of user information through the communication channel is substantiated. Recommendations for choosing a communication channel and ensuring its security are given. The schemes of the server side of the application are also given. The choice of certain hashes functions, functions of input information checks, approaches to the implementation of the protection system, storage system and processing of user information on the server side is substantiated. The developed application also shows some shortcomings of the mobile platform Android in terms of information security. It is shown how attackers can take advantage of these shortcomings, and suggestions to prevent their use are provided. The results of the work can be used to protect information in practical systems.

Keywords: instant messaging systems, information security, mobile application.