Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка системи захисту персонального комп'ютера від несанкціонованого доступу»

Студента 4 курсу, 7 групи,
спеціальності 121 «Інженерія програмного забезпечення» _____ Полякова Ярослава Сергійовича
підпис студента

Науковий керівник
кандидат технічних наук,
доцент _____ Савченко Тетяна Віталіївна
підпис керівника

Гарант освітньої програми
кандидат технічних наук,
доцент _____ Цензура Микола Олександрович
підпис керівника

КИЇВ – 2020

ЗМІСТ

ВСТУП

У банківській, а також у багатьох інших сферах часто потрібні спеціальні програмні рішення для забезпечення надійної та безпечної роботи з конфіденційними даними. Змоделюємо ситуацію, коли на робочих комп'ютерах готелю встановлено софт, який відповідає темі даного проекту. Нехай місцем дії нашої ситуації буде готель. Готель – це місце, де люди орендують номери з ціллю тимчасового проживання. Для реєстрації гості мають залишити контактну інформацію, а також паспортні дані, що допоможуть ідентифікувати їх. Виходячи з цього, необхідно прийняти заходи щодо захисту даних, що накопичує підприємство. Зазвичай, доступ до внутрішніх даних підприємства можна отримати з робочого комп'ютера адміністратора готелю. А отже необхідно ввести особливий режим доступу до всіх робочих комп'ютерів у готелі, щоб не допустити несанкціонований доступ до конфіденційної інформації. Для захисту від несанкціонованого доступу на комп'ютерах встановлено спеціальний програмний додаток, що перетворює звичайну флешку у ключ доступу до персонального комп'ютера.

Програма запам'ятовує флешку співробітника, і повністю блокує комп'ютер, коли довірений носій не підключено у usb порт. Задумка полягає у тому, щоб спростити процес доступу до робочого персонального комп'ютера, шляхом простого підключення флеш-носія. Без наявності підключеного довіреного носія використання комп'ютера неможливе. Кожен раз, коли працівник залишає своє робоче місце, він має забирати носій із собою. При цьому відбувається повне блокування робочого комп'ютера, і потенційний зловмисник залишиться ні з чим. Співробітники багатьох компаній, зокрема ті, що працюють у банках мають отримувати максимально якісний софт для роботи з даними клієнтів, адже від такої важливої сфери залежить не лише репутація фінансової установи, а й економічний розвиток

держави, не кажучи про спокій клієнтів. Також варто зазначити, що у банках завжди використовують особливий режим доступу до всіх комп'ютерів, серверів, терміналів, світчів, роутерів. Завдяки цьому всі співробітники поділяються на набір ролей із різним рівнем допуску, і уся ця система реалізована на рівні програмного забезпечення. Все банківське програмне забезпечення повинно відповідати наступим вимогам: надійність роботи, простота використання, лаконічність програмного коду, можливість підтримки та супроводу розробником, а також високий рівень безпеки. Варто пам'ятати, що із таким програмним забезпеченням будуть працювати не лише програмісти, а й звичайні користувачі. Отже кінцевий програмний продукт має бути візуально простим та інтуїтивно зрозумілим, у ньому варто використовувати прості форми та кольори, що не перевантажують зір користувача. Враховуючи вимоги, кваліфікація розробника має бути досить високою. Програмний додаток для захисту персональних комп'ютерів від несанкціонованого доступу планується розробляти у середовищі Visual Studio, мовою C#, відповідно госту 24.201-79, що забезпечить зручність розробки, високий коефіцієнт дебагінгу (debugging), а також високу якість кінцевого продукту.

У проекті буде реалізовано безпечне зберігання даних для авторизації користувача, а також програмний додаток для доступу і роботи із ними. Даний проект може бути корисним не тільки у банківській сфері, а й у будь-яких інших підприємствах, для фізичних та приватних осіб а також для задоволення особистих цілей розробника.

Метою дослідження випускної кваліфікаційної роботи є розробка програмного додатку, користувацького інтерфейсу а також механізму для безпечного та надійного зберігання даних про користувача, що в сумі забезпечить захист персонального комп'ютера від несанкціонованого доступу та безпеку інформації.

Об'єктом дослідження є безпека доступу до персональних комп'ютерів.

Предметом – програмний додаток для забезпечення захисту персонального комп'ютера від несанкціонованого доступу.

Завдання дослідження:

- Розробити програмний додаток для захисту персональних комп'ютерів від несанкціонованого доступу;
- Розібратися у предметній області захисту інформації;
- Підвищити кваліфікацію автора у області кібербезпеки;

Методи дослідження: теоретичний, емпіричний.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ВІДОМОСТІ ЩОДО ЗАХИСТУ ПЕРСОНАЛЬНИХ КОМП'ЮТЕРІВ ВІД НЕСАНКЦІОНОВАНОГО ДОСТУПУ

1.1. Аналіз сфери діяльності захисту інформації

Захист інформації – це діяльність щодо запобігання витоку, розкрадання, втрати, модифікації (підробки), несанкціонованих і ненавмисних дій, що спрямована на захист інформації. За суто технічними і ненавмисними причинами, під це визначення потрапляє також діяльність, пов'язана з підвищенням надійності сервера \ персонального комп'ютера \ хмари через відмови або збої в роботі вінчестерів, недоліків у програмному забезпеченні та інших проблем. Для вирішення проблеми захисту інформації основними засобами, що використовують для створення механізмів захисту прийнято вважати:

Технічні засоби – електричні, електромеханічні, електронні та інші типи пристроїв. Переваги технічних засобів пов'язані з їх надійністю, незалежністю від суб'єктивних факторів, високу стійкість до модифікації. Слабкі сторони - недостатня гнучкість, відносно великі обсяг і маса, висока вартість. Технічні засоби поділяються на:

- **апаратні** – пристрої, що вбудовуються безпосередньо в апаратуру, або пристрої, які сполучаються з апаратурою локальних мереж по стандартному інтерфейсу (схеми контролю інформації по парності, схеми захисту полів пам'яті по ключу, спеціальні регістри);
- **фізичні** – реалізуються у вигляді автономних пристроїв і систем (електронно-механічне обладнання охоронної сигналізації та спостереження, такі як замки на дверях, решітки на вікнах).

Програмні засоби – програми, спеціально призначені для виконання функцій, пов'язаних із захистом інформації. А саме програми для ідентифікації користувачів, контролю доступу, шифрування інформації, видалення залишкової (робочої) інформації типу тимчасових файлів, тестового контролю системи захисту та ін. Переваги програмних засобів - універсальність, гнучкість, надійність, простота установки, здатність до модифікації і розвитку. Недоліки – обмежена функціональність мережі, використання частини ресурсів файл-сервера і робочих станцій, висока чутливість до випадкових або навмисних змін, можлива залежність від типів комп'ютерів (їх апаратних засобів).

Організаційні засоби - організаційно-технічні (підготовка приміщень з комп'ютерами, прокладка кабельної системи з урахуванням вимог обмеження доступу до неї та ін.) І організаційно-правові (національні законодавства і правила роботи, що встановлюються керівництвом конкретного підприємства). Переваги організаційних засобів полягають у тому, що вони дозволяють вирішувати безліч різнорідних проблем, прості в реалізації, швидко реагують на небажані дії в мережі, мають необмежені можливості модифікації і розвитку. Недоліки – висока залежність від суб'єктивних чинників, в тому числі від загальної організації роботи в конкретному підрозділі.

Який засіб є найбільш ефективним? В ході розвитку концепції захисту інформації фахівці прийшли до висновку, що використання будь-якого одного з вище зазначених способів захисту не забезпечує надійного збереження інформації. Необхідний комплексний підхід до використання і розвитку всіх засобів і способів захисту інформації. Оскільки даний проект передбачає розробку лише програмного засобу, автор докладе всіх зусиль задля забезпечення якісної та надійної роботи програмного продукту. Сам результат дипломної роботи краще комбінувати із іншими заходами, що були

описані раніше.

1.2. Опис проблеми захисту інформації

Широке застосування комп'ютерних технологій в автоматизованих системах обробки інформації та управління призвело до загострення проблеми захисту інформації, що циркулює в комп'ютерних системах, від несанкціонованого доступу. Захист інформації в комп'ютерних системах має низку специфічних особливостей, пов'язаних з тим, що інформація не є жорстко пов'язаною з носієм, може легко і швидко копіюватися і передаватися по каналах зв'язку. Відома дуже велика кількість загроз інформації, які можуть бути реалізовані як з боку внутрішніх порушників , так і зовнішніх.

Комп'ютерними злочинами називають дії, пов'язані з втручанням у роботу комп'ютера, з метою шкоди, отримання вигоди, тощо, в результаті яких комп'ютери використовуються як об’єкт чи суб’єкт злочину. Серед причин комп'ютерних злочинів і пов'язаних з ними викрадень інформації головними є такі:

- швидкий перехід від традиційної паперової технології зберігання та передавання інформації до електронної за одночасного відставання технологій захисту інформації, зафіксованої на машинних носіях;
- широке використання локальних обчислювальних мереж, створення глобальних мереж і розширення доступу до інформаційних ресурсів;
- постійне ускладнення програмних засобів, що викликає зменшення їх надійності та збільшення кількості уразливих місць. Сьогодні ніхто не може назвати точну цифру загальних збитків від комп’ютерних злочинів, але експерти погоджуються, що відповідні суми вимірюються мільярдами доларів. Серед основних статей варто виокремити такі:
- збитки, до яких призводить ситуація, коли співробітники організації не можуть виконувати свої обов'язки через непрацездатність системи (мережі);

- вартість викрадених і скомпрометованих даних;
- витрати на відновлення роботи системи, на перевірку її цілісності, на доробку уразливих місць тощо.

Варто також враховувати й морально-психологічні наслідки для користувачів, персоналу і власників. Що ж до порушення безпеки так званих «критичних» додатків у державному і військовому управлінні, атомній енергетиці, медицині, ракетно-космічній галузі та у фінансовій сфері, то воно може призвести до тяжких наслідків для навколишнього середовища, економіки і безпеки держави, здоров'я і навіть для життя людей. Забезпечення безпеки інформаційних технологій являє собою комплексну проблему, яка охоплює правове регулювання використання ІТ, удосконалення технологій їх розробки, розвиток системи сертифікації, забезпечення відповідних організаційно-технічних умов експлуатації. Радикальне вирішення проблем захисту електронної інформації може бути отримано тільки на базі використання криптографічних програмних методів, технічних і організаційних заходів, які дозволяють вирішувати найважливіші проблеми захищеної автоматизованої обробки та передачі даних.

При цьому сучасні швидкісні методи криптографічного перетворення дозволяють зберегти початкову продуктивність автоматизованих систем. Криптографічні перетворення даних є найбільш ефективним засобом забезпечення конфіденційності даних, їхньої цілісності і справжності. Тільки їх використання в сукупності з необхідними технічними та організаційними заходами можуть забезпечити захист від широкого спектру потенційних загроз. Основні проблеми, що виникають з безпекою передачі інформації в комп'ютерних мережах, можна поділити на такі: – Перехоплення інформації - цілісність інформації зберігається, але її конфіденційність порушена; – Модифікація інформації – вихідне повідомлення змінюється або повністю підміняється іншим і надсилається адресату; – Підміна авторства інформації. Дана проблема може мати серйозні наслідки.

Наприклад, хтось може надіслати листа від чужого імені (цей вид обману прийнято називати спуфінгом) або Web-сервер може прикидатися електронним магазином, приймати замовлення, номери кредитних карт, але не висилати ніяких товарів. Потреби сучасної практичної інформатики призвели до виникнення нетрадиційних завдань захисту електронної інформації, однією з яких є аутентифікація електронної інформації в умовах, коли сторони що обмінюються інформацією не довіряють одна одній. Зростання ролі програмних і криптографічних засобів захисту проявляється в тому, що виникають нові проблеми в галузі захисту обчислювальних систем від несанкціонованого доступу, вимагають використання механізмів і протоколів з порівняно високою обчислювальною складністю і можуть бути ефективно вирішені шляхом використання ресурсів ЕОМ. Виникнення глобальних інформаційних мереж типу INTERNET є важливим досягненням комп'ютерних технологій, однак, з INTERNET пов'язана маса комп'ютерних злочинів.

Результатом досвіду застосування мережі INTERNET є слабкість традиційних механізмів захисту інформації та відставання у застосуванні сучасних методів. Криптографія надає можливість забезпечити безпеку інформації в INTERNET і зараз активно ведуться роботи з впровадження необхідних криптографічних механізмів в цю мережу. Не відмова від прогресу в інформатизації, а використання сучасних досягнень криптографії - ось стратегічно правильне рішення. Інформація повинна бути захищена в першу чергу там, де вона створюється, збирається, переробляється і тими організаціями, які несуть шкоди безпосередній при несанкціонованому доступі до даних. Цей принцип є як раціональний так і ефективний: захист інтересів окремих організацій – це складова реалізації захисту інтересів держави в цілому.

1.3. Актуальність розробки системи захисту персонального комп'ютера від несанкціонованого доступу

Розробка систем для захисту інформації є надзвичайно затребуваною не лише для банків і бізнесу, а й для звичайних людей, що хочуть забезпечити підвищену безпеку свого домашнього комп'ютера. Даний проект буде актуальним завжди, доки існують установи, що обробляють та накопичують конфіденційні дані. Також важливим фактом є те, що розробляючи подібні проекти, самі розробники отримують надзвичайно корисні професійні навички, досвід та знання. Поглянемо на ринок подібних програмних рішень станом на 01.04.2020. Основними конкурентами проекту є «USB System Lock», «WinLockr», «Predator». Проаналізуємо характеристики конкурентів, а також порівняємо їх основні характеристики між собою:

	Простота та зрозумілість інтерфейсу	Програмний Persistence	Зберігання ключа на носії	Механізм самовідновлення
Predator	-	+	-	+
USB System Lock	-	-	-	+
WinLockr	-	+	+	-

Рис. 1.1. Аналіз ринку

Виходячи з аналізу, розуміємо, що основні конкуренти мають масу недоліків, а отже автор може виграти там, де конкурентні програми вже зазнали поразки. Перше, на що дивиться користувач – зручність інтерфейсу. Він має бути інтуїтивно зрозумілим, простим у користуванні, та не навантажувати зір користувача. У конкурентів із інтерфейсом явні проблеми. Програмний persistence – механізм, що забезпечує безперебійність роботи програми, шляхом сканування активних процесів і запуску цільової програми

при збої. Не у всіх конкурентів реалізований persistence, а отже у автора є ще один спосіб краще зіграти на маркетинговій компанії. Зберігання ключа на носії – параметр, що вказує на те, чи буде зберігатися файл доступу на носії. Якщо так – при видаленні файлу з ключами чи форматуванні носія розблокування флешкою перестає працювати, а якщо ні – ключі на флешці не зберігаються. Автор планує реалізувани механізм аналізу заліза, а отже форматування носія ніяк не завадить програмі ідентифікувати флешку. Механізм самовідновлення потрібний для відновлення файлів програми при їх видаленні чи пошкодженні. Автор з легкістю реалізує цей простий механізм захисту.

1.4. Технічне завдання

Загальні відомості: Дане технічне завдання є основним документом для створення АСУ, і розроблено на основі ГОСТу 24.201-79. У технічне завдання входить розробка документації відносно програмного додатку для ОС Windows. Об’єкт даного технічного завдання має бути простим у користуванні, і гарно оптимізованим. Також АСУ повинна працювати із хешованими даними, що зберігаються у системному регістрі.

Назва проекту: ”Secure IT”.

Опис проекту: ”Secure IT” – це програмний продукт, що призначений для захисту персональних комп’ютерів від несанкціонованого доступу, шляхом використання USB накопичувача в якості фізичного ключа доступу. При підключенні флеш-ключа додаток проводить перевірку і дозволяє використовувати систему. При відключенні ключа додаток блокує комп’ютер.

Вимоги до системи:

- Планові терміни початку та закінчення робіт: 01.09.2019-01.12.2019

  Головним бенефіціаром є розробник. Потенційними користувачами системи є фізичні, юридичні та приватні особи, яким за тих чи інших

причин потрібно забезпечити підвищену безпеку для своїх персональних комп’ютерів. Право на використання системи мають фізичні, юридичні та приватні особи, що придбали у розробника право на користування системою. Продаж або передача проекту стороннім особам не розглядається.

- Мета та призначення системи: Метою системи є забезпечення контролю доступу до персонального комп’ютера, шляхом аналізу підключених USB девайсів. Система призначається для захисту персональних комп’ютерів від несанкціонованого доступу.
- Вимоги до способів і засобів інформаційного обміну між компонентами системи: Система будується з програмних процедурно-функціональних компонентів, що називаються класами. Інформаційний обмін між компонентами реалізується як передача параметрів між процедурами та функціями, а також через публічні, загальнодоступні поля у класах.
- Вимоги до режимів функціонування системи: Система має всього два режими: режим аналізу – коли довірений флеш-носій не підключено і комп’ютер заблоковано системою, вона накладає хук на підключення USB пристроїв, і при спрацьовуванні хука проводить аналіз пристрою. Якщо результат аналізу еквівалентний true – система знімає блокування. Режим очікування – коли довірений пристрій підключено, система накладає хук на відключення USB пристроїв, і при спрацьовуванні хука перевіряє, який саме пристрій було від'єднано. Якщо це був довірений пристрій – система блокує комп'ютер.
- Вимоги до діагностування системи: Задля забезпечення якісної та ефективної діагностики необхідно створити набір методів для логування всіх подій всередині програми, щоб, у разі збою розробник міг відновити ланцюжок подій і викликів. Завдяки цьому буде простіше

проводити аналіз послідовності виконання процедур та функцій, що призвели до збою.

- Вимоги до режимів управління системою: Управління системою відбувається у пасивному та активному режимах, де активні це ті, коли користувач взаємодіє з користувацьким інтерфейсом додатку, а пасивні – коли програма автоматично виконує всі необхідні дії. Як приклад активних – реєстрація користувачем флеш-ключа. Як приклад пасивних – коли користувач підключає ключ, і програма самостійно ініціює ланцюжок викликів, що ведуть до розблокування.
- Параметри, що характеризують ступінь відповідності системи призначенням: Ергономічність – характеризує загальну зручність та задоволення від використання системи. Досягається шляхом якісного макетування. Надійність – характеризує стабільність роботи системи, її незалежність від зовнішніх факторів. Технічна конкурентоздатність – характеризує економічну величину, що досягається шляхом порівняння додатку із додатками-конкурентами і грошовою вигодою від продажів.
- Вимоги до захисту інформації від несанкціонованого доступу: Програма має безпечно та надійно взаємодіяти із авторизаційними даними, які не мають потрапити до рук зловмисника. У нашому випадку дані для авторизації мають хешуватися, та зберігатися у системному регістрі. Будь-які порівняння та перевірки будуть проводитись лише із хешами.
- Вимоги до інформаційного забезпечення: Інформаційне забезпечення додатку – текстові дані, що збираються із користувацького інтерфейсу, нетекстові дані, що збираються із властивостей заліза, та хешовані дані із системного регістру.
- Вимоги до складу, структури і способів організації даних в системі: дані у програмі мають бути двох типів: процедурні (ті, що будуть

- отримуватися, модифікуватися та використовуватися у процесі роботи програми, незалежно від користувача) і користувацькі (ті, що вводить користувач при реєстрації, зберігаються у вигляді хешу та не змінюються без участі користувача). Перший тип зберігається у оперативній пам'яті, та отримується нальоту. Другий – необхідній для розблокування, необхідно помістити до системного реєстру.
- Вимоги до інформаційного обміну між компонентами системи: доступність кожного компоненту програми (класу\процедури\функції) необхідно розділити модифікаторами доступу, що дозволить збільшити чіткість роботи програми, зменшити кількість помилок що може допустити розробник а також забезпечити захист від доступу з інших програм, що підвищить захисні якості та стійкість від злому. Обмін даними, а також механіку викликів процедур та функцій всередині програми реалізувати в рамках рівня доступності Internal, що забезпечить виконуваність вимог, що вказані вище.

1.5. Висновки до розділу 1

Захист інформації – невід'ємна складова не лише бізнесу, а й звичайних людей, що дбають про конфіденційність своїх даних. Проблема захисту інформації від навмисних та ненавмисних факторів гостро стоїть у повсякденному житті багатьох фізичних та юридичних осіб. Не зважаючи на деяку кількість подібних додатків для захисту персональних комп'ютерів на ринку, що розповсюджуються на платній та безкоштовній основі, виникає бажання створити дещо краще. Простий аналіз сучасної ситуації показує, що ринок подібних продуктів залишає бажати кращого. Це відкриває нові бізнес-можливості для розробника, та допомагає йому вдосконалювати свої професійні навички. Програмування є досить складним і творчим процесом через велику кількість варіантів реалізації кожної конкретної задачі. Глибоко проаналізувавши сучасне становище та свої професійні навички, автор

впевнений у явній перевазі свого продукту над конкурентними додатками.

Перед автором постало непросте та дуже цікаве завдання. Враховуючи вимоги до проекту на виході автор отримає досить ефективний, конкурентоздатний, надійний та приємний у використанні програмний продукт. Автор впевнений, що цей проект буде корисним у сферах бізнесу, захисту інформації, програмування та гарним чином вплине на безпеку бізнесу в цілому.

РОЗДІЛ 2.
ОПИС ТА РОЗРОБКА СИСТЕМИ ЗАХИСТУ ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА ВІД НЕСАНКЦІОНОВАНОГО ДОСТУПУ

2.1 Аналіз предмета дослідження та опис його параметрів та характеристик

Предмет дослідження – програмний додаток для забезпечення захисту персональних комп'ютерів банку від несанкціонованого доступу.

Кінцевий програмний продукт матиме вигляд файлу, формату Portable Executable, у розширенні «.exe». Portable Executable (PE, «мобільний виконуваний») – формат виконуваних файлів, об'єктного коду та динамічних бібліотек, який використовується в 32- і 64-розрядних версіях операційної системи Microsoft Windows. Формат PE є структурою даних, що містить всю інформацію, необхідну PE-завантажувачу для відображення файлу в пам'ять. Виконуваний код включає в себе посилання для зв'язування динамічно завантажуваних бібліотек, таблиці експорту і імпорту API-функцій, дані для управління ресурсами і дані локальної пам'яті потоку. Файл PE складається з декількох заголовків і секцій, які вказують динамічному компонувальнику, як відображати файл у пам'ять. Виконуваний образ складається з декількох різних областей (секцій), кожна з яких вимагає різних прав доступу до пам'яті. таким чином, початок кожної секції повинно бути вирівняно по межі сторінки. Наприклад, зазвичай секція .text, яка містить код програми, відображена як виконувана / прийнятна тільки для читання, а секція .data, що містить глобальні змінні, відображена як невиконувана / прийнятна для читання і запису. Однак, щоб не витрачати даремно простір на жорсткому диску, різні секції на ньому на кордоні сторінки не вирівняні. Частина роботи динамічного компонувальника полягає в тому, щоб відобразити кожну

секцію в пам'ять окремо і привласнити коректні права доступу областям згідно з вказівками, що містяться в заголовках. Розробник постачатиме файл програми без електронного цифрового підпису, оскільки на перших етапах продажів у проекту будуть явні фінансові нестачі. Об’єкт має бути побудований з урахуванням чіткої архітектури, кожен клас, як елемент архітектури має бути ізольованим від зовнішнього середовища, методи мають чітко обмінюватись необхідними даними. Код додатку буде легко читатися, доповнюватись, модифікуватися, та коментуватися.

2.2. Опис архітектури додатку

Розробник доклав значних зусиль у процесі розробки архітектури проекту та структури програмного коду додатку, що забезпечить лаконічність програмного коду, зручність розробки та модифікації. Список запланованих класів виглядає так:

MainForm – головна форма, з якої починається ланцюжок викликів, процедур встановлення системи, перевірок дієздатності, блокувань та розблокувань. Основне призначення цього класу – справити перше враження від користування додатком, зібрати необхідні дані для захисту системи, виконати їх перетворення, передати дані до інших класів та ініціювати набір методів для захисту.

GlobalHook – клас, що призначений для накладання глобального хуку на пристрої вводу, а також прикріплення свого коду до зареєстрованих подій у системі. Цей клас виконує функцію слідкування за активністю користувача, щоб у разі бездіяльності заблокувати систему. Також цей клас може використовуватись для перехоплення та аналізу даних із миші, клавіатури. Будь-яке натискання, утримання та відпускання клавіші буде зареєстровано. Усі зміни координат стрілки миші, кліки та обертання колеса можна відслідкувати й зреагувати відповідним методом.

KeyboardFilter – Надзвичайно важливий захисний клас, основним

призначенням якого є перехоплення окремих натисків чи комбінацій клавіш на клавіатурі, з можливістю підмінити сигнал, чи навіть заблокувати його. Це потрібно, щоб користувач не зміг згорнути чи закрити вікна програми, коли вона перебуває у режимі блокування. Також цей клас унеможливить використання Alt+tab, Win+D, Alt+F4, та інших комбінацій, що можуть призвести до несанкціонованого доступу.

LockScreen – Являє собою клас типу Form, графічний елемент, основним призначенням якого є відображення блокуючих екранів та інтерфейсу на них, у якому зберігається набір методів, що забезпечують безпеку та зручність користування.

Logger – Клас, що призначений для діагностики готового продукту. Його основне призначення – логування всіх подій, що виконуються користувачем, а також автоматичних подій. Методи у ньому формують текстовий документ із назвою події, програмним коментарем та часом, коли вона відбулась. При збої у програмі, клієнт може звернутися до розробника і надати йому лог, завдяки чому розробнику буде простіше зрозуміти причини збою, адже у ньому зберігається ланцюжок викликів безпосередньо перед збоєм. Це допоможе розробнику полагодити програму.

MsgSender – Захисний клас, методи якого знімають фокус з цільової програми а також посилають їй сигнал згортання. Це потрібно, коли деякій програмі, чи навіть користувачу вдалося відкрити стороннє вікно поверх екрану блокування. У цьому випадку «Secure IT» змушує її згорнутись.

Program – Клас, що є основною точкою входу у програму. Автор пристосував його для отримання параметрів командного рядку, що дозволяє програмі стартувати у режимі відновлення, звичайному режимі, а також у режимі захисту.

Security – Головний захисний клас, у якому зібрані усі найважливіші захисні механізми програми, а саме: сканування USB пристроїв, аналіз моніторів (для коректного відображення екранів блокування, якщо їх більше

ніж один), перевірка встановленої версії захисту, захист від подвійного запуску, методи знищення копій захисту у оперативній пам'яті, отримання даних заліза флеш-накопичувачів, шифрування, запис авторизаційних даних до регістру, методи блокування \ розблокування, методи зняття захисту.

Setup – клас, у якому міститься механіка встановлення програмного продукту на персональний комп'ютер. Також у класі зберігаються публічні поля з інформацією про положення файлів у файловій системі, та декілька варіантів встановлення: перше встановлення, оновлення та деінсталяція.

Shortcut – клас, що використовує бібліотеку IWshRuntimeLibrary створює на робочому столі ярлик із параметрами для запуску системи.

USBSerialNumber – захисний клас, що робить запит до операційної системи щодо всіх USB пристроїв, із них вибирає лише ті, які мають тип removable і з них повертає той пристрій, буква якого відповідає критеріям запиту. Саме цей клас зчитує так звані дані заліза флеш-пристрою, які пізніше хешуються та поміщаються в регістр, а також зчитуються при перевірці носія.

WindowsWathcer – захисний клас, призначенням якого є постійний контроль за усіма відкритими вікнами, і, якщо потрібно – згортання цільового вікна. Цей клас працює разом із класами Security та MsgSender, які доповнюють та посилюють його ефективність, а також збільшують ефективність захисту вцілому.

Якщо візуалізувати приведений вище опис класів у вигляді UML діаграми, отримаємо графічне представлення наступного вигляду:

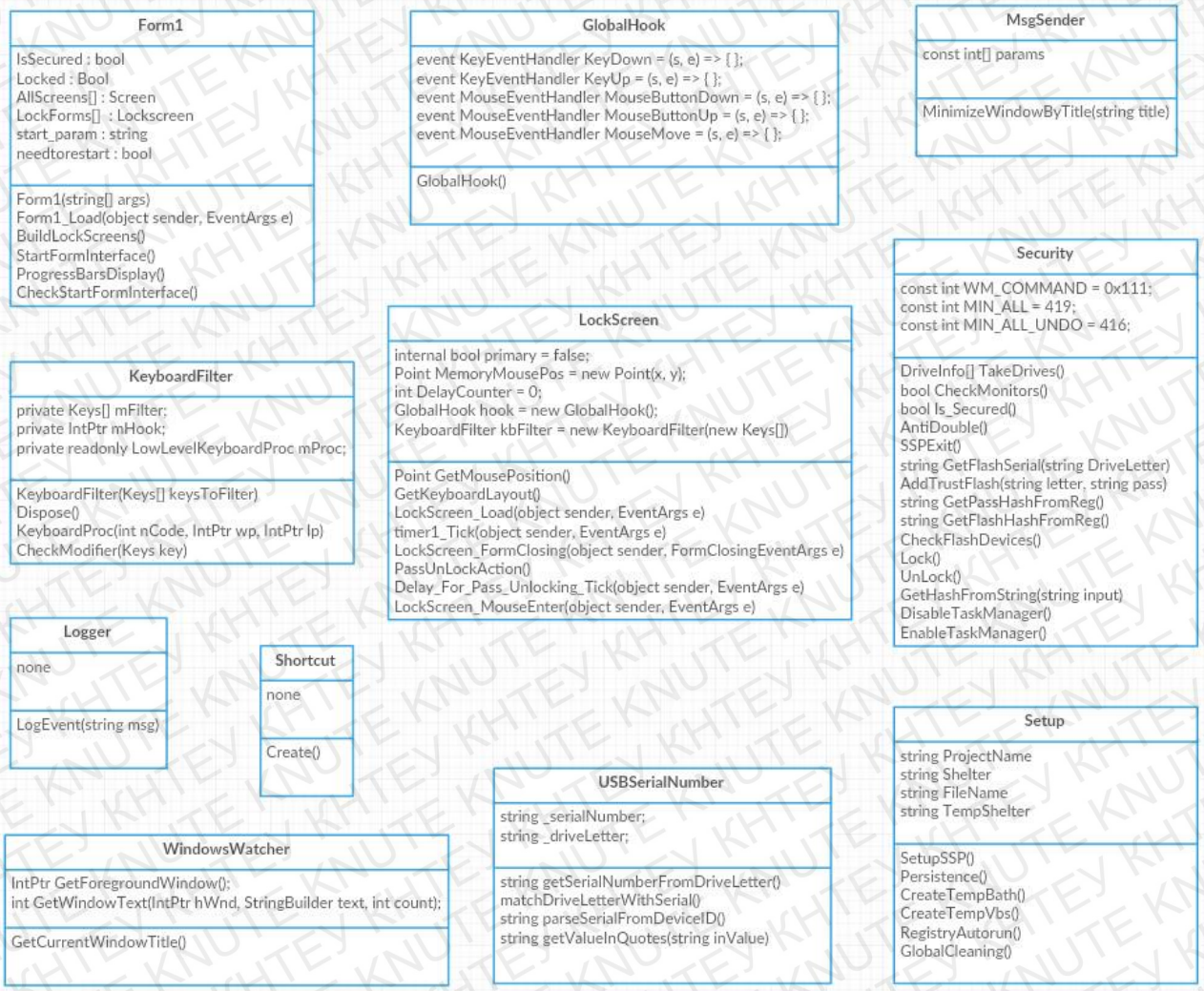

Рис. 2.1. Візуалізація класів, полів та методів

Як ми бачимо, додаток буде досить складним та багатофункціональним. Кожен клас буде схожий на окрему полицю для методів єдиного призначення. Методи кожного класу мають свої коментарі, щоб розробник у будь який момент міг у них розібратися. Деякі класи мають публічні поля, доступ до яких можуть отримувати методи інших класів, а також приватні поля, які статично зберігаються у оперативній пам'яті комп'ютера. Взаємодія класів побудована на передачі параметрів від методу до методу, а також від публічного поля до методу. Кожен клас буде доступний лише для внутрішнього користування. Це означає, що інші програми, як і користувач не зможуть викликати методи із зовнішнього середовища, що може призвести до несанкціонованого розблокування системи, завдяки чому

система буде досить стійкою до злому. Звернемо увагу на класи ще раз, і додамо лінії двосторонньої взаємодії між класами. Отримаємо схему міжкласових зв'язків у програмі:

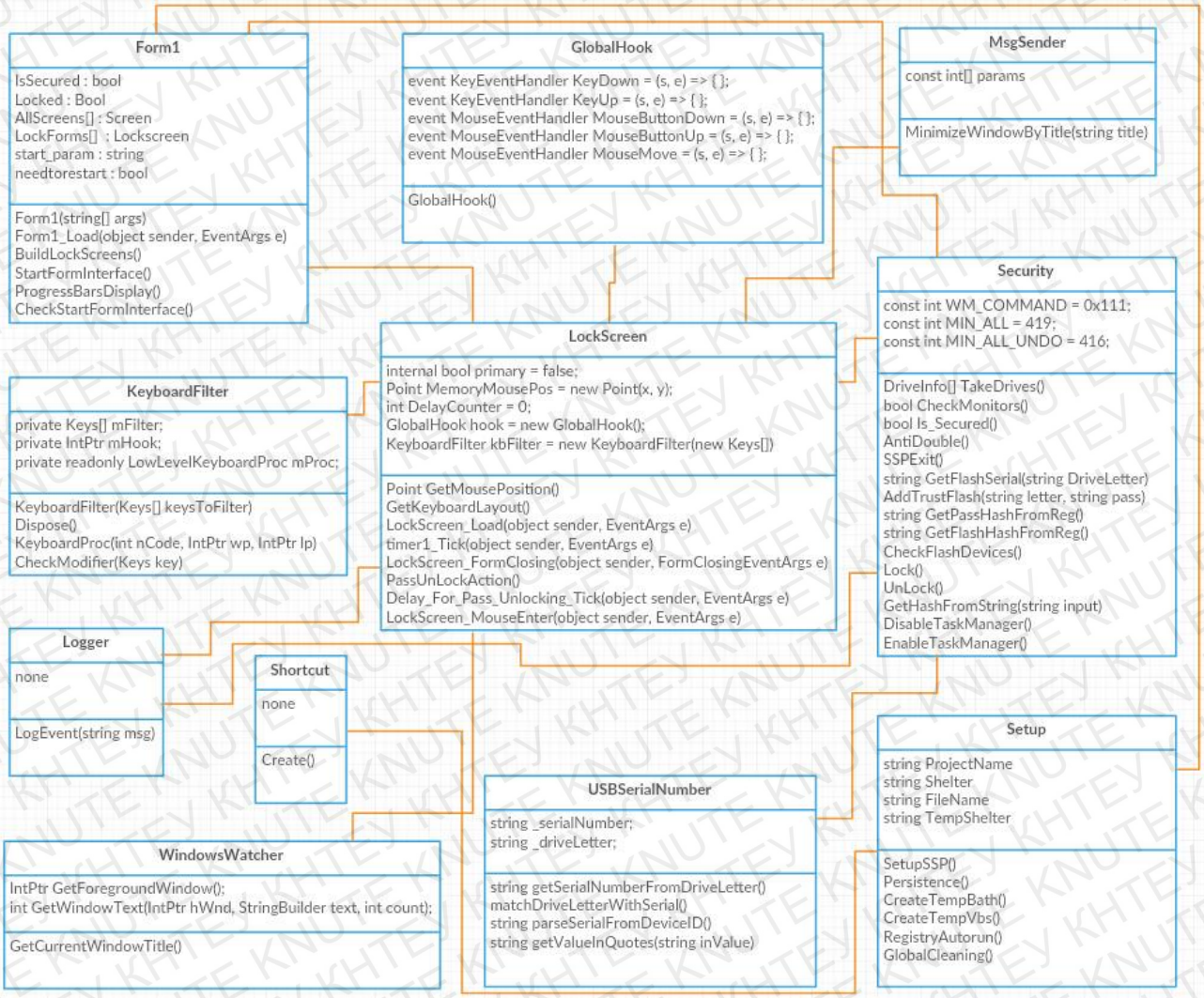

Рис. 2.2. Візуалізація взаємодії класів

2.3. Опис алгоритмів роботи додатку

Автор розробив детальний опис алгоритму роботи додатку для операційної системи Microsoft Windows 10, а також для будь-яких інших версій, із якими сумісний .NET Framework 4.5. Далі автор описує алгоритм роботи додатку у текстовому вигляді: Після старту клас Program отримує параметри командного рядку (якщо вони є, у іншому випадку передається null), передає ці параметри у клас Form1, метод Form1(string[] args). Після цього стартує перевірку на стан захисту. Якщо метод перевірки стану захисту

повертає false – система «Secure IT» не встановлена на даному комп'ютері. Якщо метод перевірки стану захисту повертає true – система «Secure IT» встановлена. У системі реалізований mutex. Це механізм, що не дозволить запустити систему двічі – друга самостійно закриває свій процес. Далі система аналізує параметри командного рядку і якщо вони відсутні – місце старту не є місцем встановленої програми чи будь-якою іншою службовою адресою у файловій системі. Далі йде черга відображення інтерфейсу. Якщо захист не встановлено – програма відображає стартовий інтерфейс та пропонує встановити захист. Стартовий інтерфейс складається з двох текстових полів для введення паролю, списку для відображення доступних у системі флеш-накопичувачів, графічного лейблу, чекбоксу для прийняття ліцензійної згоди, та кнопки для встановлення захисту. Щоб встановити захист на свій комп'ютер – користувач має під'єднати накопичувач у USB порт, обрати його у програмі, ввести пароль на випадок, якщо флешку буде втрачено, прийняти ліцензійну згоду та натиснути на кнопку підтвердження. При цьому додаток виконає копіювання на жорсткий диск, додасть себе до автозапуску, захешує та помістить авторизаційні дані до регістру, й перезапустить себе вже із місця встановлення. Якщо при старті програми захист вже встановлено та mutex не завершив процес - програма запускає ланцюжок викликів для блокування системи. При цьому перевіряється, чи під'єднаний до комп'ютера довірений пристрій. Якщо він під'єднаний – блокування не відбувається. Блокування являє собою створення й відображення на екрані(ах) кількості об'єктів класу LockScreen, що дорівнює кількості моніторів. Кожен об'єкт збирається під окремий монітор, враховуються параметри домінантності, кольорової гами та розмірів для кожного конкретного монітору. На кожній формі інтерфейс динамічно пристосовується під висоту та ширину екрану, займає центральне положення. Після присвоєння об'єктам параметрів вони зберігаються у масив типу Form, і кожен раз, коли відбувається блокування – ці об'єкти відображаються,

кожен на своєму моніторі. Якщо під час роботи додатку характеристики моніторів змінюються – блокуючі форми динамічно пристосовуються під нові розміри та кількість моніторів, інтерфейс всередині них також коректно займає внутрішній простір. У режимі блокування форми займають всі екрани, стрілка миші знаходиться на головному дисплеї, та не може його покинути. При ініціалізації блокуючих форм в кожну із них поміщаються екземпляри об’єктів KeyboardFilter та GlobalHook. Завдяки першому програма перехоплює небажані натиски та комбінації клавіш, унеможливлюючи згортання \ закриття блокуючих форм \ відкриття будь-яких вікон поверх блокуючих форм \ перехід на другий робочий стіл, при цьому користувачу нічого не заважає користуватися елементами інтерфейсу та вводити пароль. Завдяки другому – блокуючі форми слідкують за мишею, тримають її вказівник у невеличкій області в центрі головного монітору, не дозволяють покинути головний екран. Головна блокуюча форма містить у собі мінімально необхідний інтерфейс для брендування додатку (логотип та назва), а також елементи управління для розблокування комп’ютера за допомогою пароля. Також на ній є кнопка для вимкнення комп’ютера. На усіх інших (не головних) моніторах блокуючі форми мають лише один графічний елемент – логотип у центрі екрану. У системі постійно активний хук на накопичувальні пристрої. При кожному підключенні чи відключенні флеш-пристроїв програма реагує відповідним методом, логіка якого вирішує, чи необхідно блокувати \ розблокувати систему. Саме на цьому методі зав'язана система розпізнавання свій-чужий. Більш зрозуміло та компактно автор зобразив алгоритм роботи додатку у наступній діаграмі:

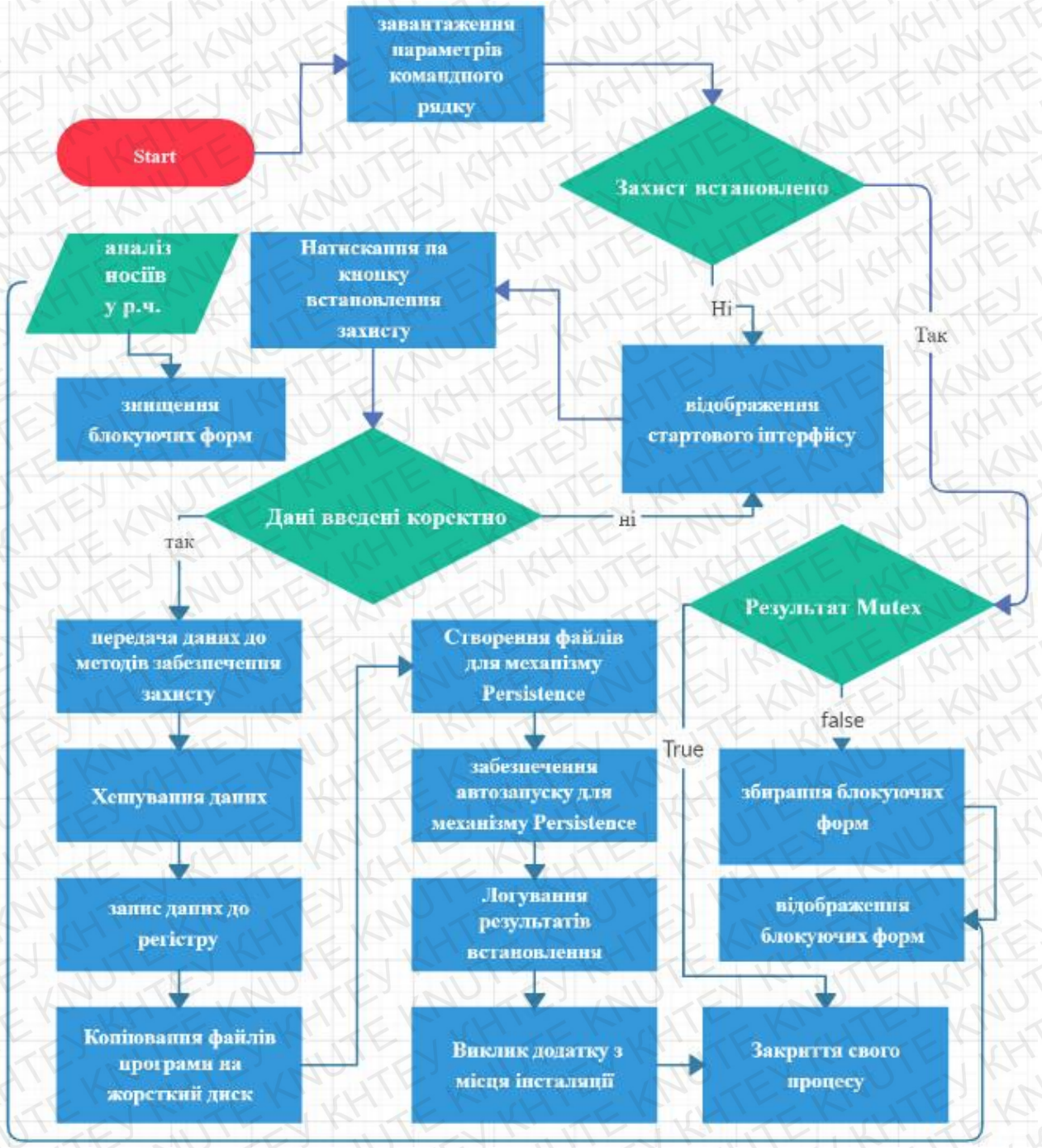

Рис. 2.3. Алгоритм роботи додатку

2.4. Середовище Microsoft Visual Studio 2017 і розробка додатку

Розробником прийнято рішення розробляти додаток у середовищі Microsoft Visual Studio 2017, мовою C#. Дане середовище є універсальним рішенням для розробки програмних продуктів на багатьох популярних мовах програмування, а версія від 2017 року визнана багатьма розробникам як «стабільний білд».

Створимо новий проект, у меню шаблонів оберемо «додаток WindowsForms»:

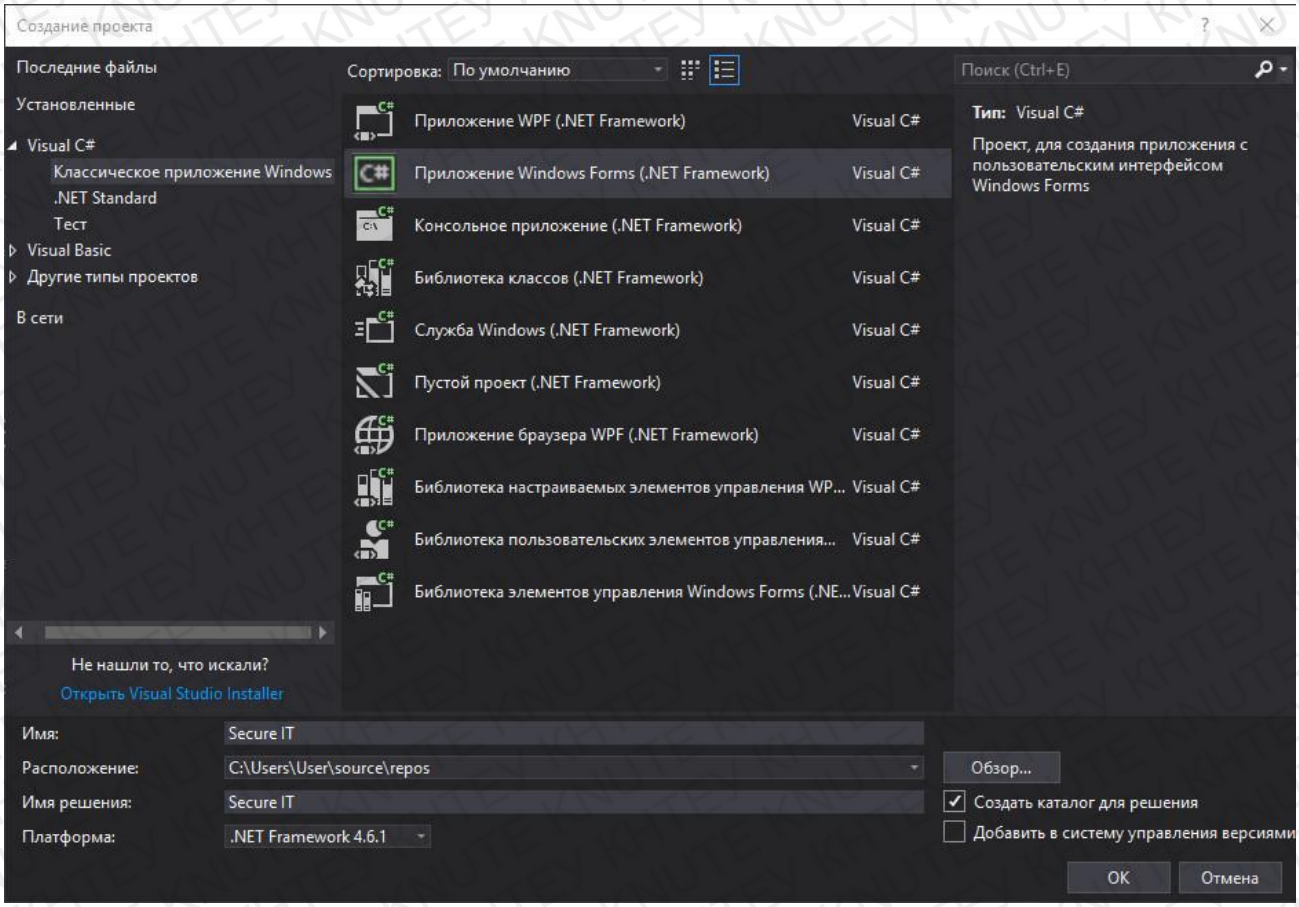

Рис. 2.4. Початок розробки додатку

Розмістимо на формі наступні елементи управління: зображення з лейблом проекту, два текстових поля, одне поле-список, чекбокс, панель, два прогрес-бари, кнопку. Пересунемо їх у зручне для користувача місце, змінимо кольорову гаму на прості на приємні кольори, додамо свої графічні елементи для закриття вікна. Отримаємо наступний результат оформлення інтерфейсу:

Рис. 2.5. Макетування стартового інтерфейсу

Створимо процедури для перевірки правильності вводу праоля, а також його підтвердження. Також додамо до прогрес-барів реакцію на зміну кількості символів у текстових полях, зв'яжемо процедури у єдину подію:

```
private void textBox1_TextChanged(object sender, EventArgs e)       // прогресс напротив пароля
{
    if (textBox1.Text.Length < progressBar1.Maximum)
    {
        progressBar1.Value = textBox1.Text.Length;
    }
    else { progressBar1.Value = progressBar1.Maximum; }

    ProgressBarsDisplay();
}

private void ProgressBarsDisplay()          // визуализация проверки пароля
{
    if ((textBox1.Text.Length >= progressBar1.Maximum) && (textBox2.Text == textBox1.Text))
    {
        progressBar2.Value = progressBar1.Maximum;
    }
    else { progressBar2.Value = 0; }
}
```

Рис. 2.5. Візуалізація перевірки введених даних

Створимо ще декілька процедур, що виконуватимуть суто косметичну функцію, а саме підсвітку лейблів для прийняття ліцензійної згоди, унеможливимо написання тексту там, де користувач не повинен писати, створимо зручний перехід до наступного елемента управління, шляхом натискання клавіші enter:

```
private void textBox2_KeyDown(object sender, KeyEventArgs e)        // смена фокуса
{
    if (e.KeyCode == Keys.Enter)
    {
        CheckStartFormInterface();
    }
}

private void label1_MouseLeave(object sender, EventArgs e)          // подсветка автора
{
    label1.ForeColor = Color.WhiteSmoke;
}

private void label5_MouseEnter(object sender, EventArgs e)          // подсветка соглашения
{
    label5.ForeColor = Color.Orange;
}

private void Form1_MouseMove(object sender, MouseEventArgs e)// выделение текста в comboBox1
{
    comboBox1.SelectionLength = 0;
}

private void checkBox1_Click(object sender, EventArgs e)   //  выделение текста в checkBox1
{
    label5.Focus();
}
```

Рис. 2.6. Косметичні процедури для зручності користування додатком

Створимо класи, про які йдеться у описі архітектури додатку, розробимо дизайн у директорії проекту й отримаємо наступне дерево проекту:

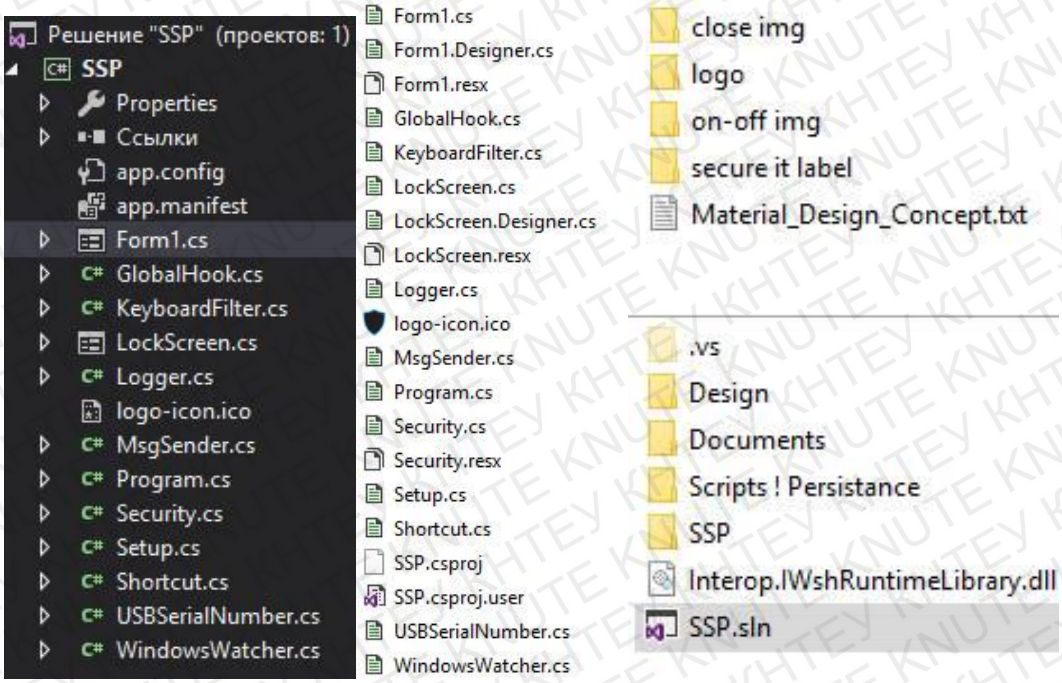

Рис. 2.7. Дерево проекту

Створимо процедуру перевірки правильності вводу користувацької інформації, виконаємо сортування та передамо її до класу Security:

```
private void CheckStartFormInterface()                          // проверка введённых данных
{

    if (comboBox1.Items.Count < 1 || comboBox1.Text == "USB Drives not found")
    {
        MessageBox.Show("Connect your flash Drive!", Setup.ProjectName);
    }
    else if (textBox1.Text.Length < progressBar1.Maximum)
    {
        MessageBox.Show("This password too short!", Setup.ProjectName);
        textBox1.Focus();
    }
    else if (textBox1.Text != textBox2.Text)
    {
        MessageBox.Show("Passwords do not match!", Setup.ProjectName);
        textBox1.Text = "";
        textBox2.Text = "";
        textBox1.Focus();
    }
    else if(!checkBox1.Checked)
    {
        MessageBox.Show("You should accept author`s policy!", Setup.ProjectName);
        label5.ForeColor = Color.Orange;
        label5.Focus();
    }
    else
    {
        string letter = null;

        if(comboBox1.Text.Length == 3)
        {
            letter = comboBox1.Text;
        }
        else
        {
            letter = comboBox1.Text.Substring(comboBox1.Text.Length-3, 3);
        }

        notifyIcon1.Visible = false;
        this.Hide();
        Security.AddTrustFlash(letter, textBox1.Text); // делаем ключики
        needtorestart = true;
        Setup.SetupSSP();                                       // копируемся, прописываемся в автостарт
    }
}
```

Рис. 2.8. Метод, що ініціює встановлення захисту

Перейдемо до класу Security й розробимо бізнес-логіку головного захисного класу. Цей клас вміщатиме методи для сканування периферійних пристроїв, аналізу доступних моніторів, перевірок захищеності, отримання даних серійних номерів накопичувачів, встановлення та видалення захисту,

шифрування, блокування та розблокування, а також сигнатури для контролю над відкритими вікнами у системі:

```
internal static DriveInfo[] TakeDrives()                    // L Возвращает массив с инфой о каждом usb накопителе...
internal static void CheckMonitors()                        // L перезапускает, если мониторы изменились...
internal static bool Is_Secured()                           // !L Проверяет, защищена ли система (есть ли запись в реестре)
internal static bool Is_Runned()                            // !L Проверяет, запущена ли система (если имя файла было измен
internal static void AntiDouble()                           // !L убивает себя, если экземпляр уже запущен...
internal static void SSPExit()                              // !L правльное и безоговорочное убийство себя...
private static string GetFlashSerial(string DriveLetter)    // !L возвращает серийник флешки по букве ...
internal static void AddTrustFlash(string letter, string pass) // L добавляет запись в реестр о *правильной флешке*...
internal static string GetPassHashFromReg()                 // L возвращает хеш пароля из реестра...
internal static string GetFlashHashFromReg()                // L возвращает хеш флешки из реестра...
internal static void RemoveTrustFlash(bool stop)            // L удаляет запись из реестра о *правильной флешке*...
internal static void CheckFlashDevices()                    // L проверяет все доступные флешки с целью обнаружения правиль
internal static void Lock()...
                                                            // Блок\разблок
internal static void UnLock()...
internal static string GetHashFromString(string input)      // !L возвращает SHA384 из строки...
private static void DisableTaskManager()...
                                                            // откл\вкл диспетчер
internal static void EnableTaskManager()...
```

Рис. 2.9. Структура головного захисного класу

Перейдемо до класу Logger, створимо у ньому метод для отримання інформації про подію, додамо цикл що приведе стрічку у зручну для читання форму, а також додасть часовий підпис:

```
if ((System.DateTime.Now - File.GetCreationTime(Setup.Shelter + "Logfile.log")).TotalDays > 60)
{
    File.Delete(Setup.Shelter + "Logfile.log");
    File.Create(Setup.Shelter + "Logfile.log").Close();
}

msg += " (" + (System.DateTime.Now).ToString("dd.MM.yyyy|HH:mm:ss") + ")";

string CorrectMsg = "";
int startindex = 0;      //первая буква реквизита, от которого начинаем считать до следующего пробела
int endindex = 0;        //пробел, до которого считаем

for (int i = 0; i < msg.Length; i++)
{
    if (msg[i].ToString() == " ")
    {
        endindex = i;

        for (int a = 0; a < 34 - (endindex - startindex); a++)
        {
            CorrectMsg += " ";
        }
        startindex = i;
    }
    else { CorrectMsg = CorrectMsg + msg[i].ToString(); }
}


StreamWriter streamwriter = new StreamWriter(Setup.Shelter + "Logfile.log", true, System.Text.Encoding.GetEncoding("utf-8"));
streamwriter.WriteLine(CorrectMsg);
streamwriter.Close();
```

Рис. 2.10. Механізм логування

Заповнимо клас Setup методами встановлення, додамо скрипковий механізм Persistence, розмістимо метод для додання усіх необхідних файлів до автозапуску, шляхом пропису їх у регістр. Також розробимо аварійний механізм, завдяки розробник зможе повністю деінсталювати додаток, що забезпечить більш якісне та зручне тестування:

```
internal class Setup // установка, копирование, автозагрузка, обновление, Persistence, uninstall
{

    internal static string ProjectName = "Secure IT"; // Имя всего проекта меняется динамически, этой переменной
    internal static string Shelter = Environment.GetFolderPath(Environment.SpecialFolder.Windows).Substring(0, 3) + @"Users\"
    internal static string FileName = ProjectName + ".exe"; // имя + .exe
    internal static string TempShelter = Environment.GetFolderPath(Environment.SpecialFolder.Windows).Substring(0, 3) + @"User


    internal static void SetupSSP() // Создаём убежище, копируем прогу, прописываем в реестре, обновляемся: (если нужно)...

    internal static void  Persistence() // устойчивость от закрытия...

    internal static void CreateTempBath() // создаем батник для настойчивости...

    internal static void CreateTempVbs() // скрипт, для тихого запуска настойчивости ...

    internal static void RegistryAutorun()
    {
        // открываем нужную ветку в реестре
        Microsoft.Win32.RegistryKey Key = Microsoft.Win32.Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\Curre

        //первый параметр - название ключа, Второй - путь к исполняемому файлу программы.
        Key.SetValue(ProjectName, Shelter + FileName);
        Key.SetValue("SSPWatcher", TempShelter + "silentWatcher.vbs");
        Key.Close();    // записываем

    }

    internal static void GlobalCleaning() // полное удаление системы с компьютера...
}
```

Рис. 2.11. Структура класу Setup

Зробимо зручний запуск додатку, шляхом розміщення на робочому столі ярлику з логотипом проекту та підписом «Secure IT». Мова C# не має вбудованих засобів для створення ярликів, тому автор знайшов альтернативне рішення на GitHub. Суть рішення полягає у доданні до проекту стороннього .DLL файлу, у якому вже реалізований клас для прийняття параметрів, необхідних для створення ярлику, а також готовий метод, що на основі переданих параметрів створює ярлик. .DLL файл може бути написаний на будь-якій іншій мові програмування, головне правильно передати параметри методам, що в ньому знаходяться.

Залежність, яку автор додав до проекту називається IWshRuntimeLibrary. Готовий лістинг додання ярлику виглядає так:

```
class Shortcut // при установке создавать ярлык на робочем столе
{
    public static void Create()
    {
        try
        {
            WshShell shell = new WshShell();

            IWshShortcut shortcut = (IWshShortcut)shell.CreateShortcut(Path.Combine(Environment.GetFolderPath(Environment.Spec
            shortcut.Description = Setup.ProjectName;
            shortcut.Hotkey = "Ctrl+Shift+S";
            shortcut.TargetPath = Setup.Shelter + Setup.FileName;
            shortcut.Arguments = @"-Shortcut";
            shortcut.Save();

            Logger.LogEvent("Shortcut_created!");
        }
        catch
        {
            Logger.LogEvent("###_Shortcut.Create() ERROR");
        }
    }
}
```

Рис. 2.12. Додання ярлику

Внесемо методи стеження за активністю вікон у клас WindowsWatcher. Цей клас призначений для того, щоб у будь-який момент часу, коли це необхідно, повернути заголовок того вікна, з яким зараз працює користувач. Завдяки цьому можна підвищити захищеність у режимі блокування. Таке вікно може були лише одне, а показник, що вказує на його стан у реальному часі називають фокусом.

```
internal class WindowsWatcher // Через этот класс происходит наблюдение за активностью окон
{
    [DllImport("user32.dll")]
    static extern IntPtr GetForegroundWindow();

    [DllImport("user32.dll")]
    static extern int GetWindowText(IntPtr hWnd, StringBuilder text, int count);

    internal static string GetCurrentWindowTitle()
    {
        try
        {
            const int nChars = 256;
            StringBuilder Buff = new StringBuilder(nChars);
            IntPtr handle = GetForegroundWindow();

            if (GetWindowText(handle, Buff, nChars) > 0)
            {
                return Buff.ToString();
            }
            return null;
        }
        catch
        {
            Logger.LogEvent("###_GetCurrentWindowTitle() ERROR"); Security.SSPExit(); return null;
        }
    }
}
```

Рис. 2.13. Механізм відстеження активності вікон

Шукаючи спосіб відрізнити один флеш-носій від іншого (не зберігаючи на цільовому носії файл з ключем), автор відвідав багато технічних ресурсів, і знайшов наступне рішення цієї технічної проблеми: необхідно зчитувати ті дані, що не змінюються для кожного конкретного носія. Такими даними є дані заліза, а також серійні номери (але автор зробив висновок, що цей спосіб не є досить надійним, оскільки кожен виробник у різних партіях одних і тих самих флешок може прописувати однакові серійні номери, не кажучи про дані заліза, які завжди є ідентичними у двох однакових товарів). Щоб уникнути колізій, розробник знайшов більш надійне та дієве рішення – використання унікального ідентифікатору носія, що присвоюється операційною системою при першому підключенні, і може змінюватись лише при форматуванні. При кожному підключенні нового пристрою операційна система генерує новий ідентифікатор, не схожий на інші. З'єднавши цей ідентифікатор із місткістю (у байтах) та типом файлової системи, отримуємо досить стійкий до колізій хеш. Саме цей хеш додаток вираховує при підключенні носіїв, та порівнює із хешем зареєстрованого пристрою. Програмно цей механізм реалізовано наступним чином:

```
internal class USBSerialNumber // возвращает серийник флешки по букве
{
    string _serialNumber;
    string _driveLetter;

    internal string getSerialNumberFromDriveLetter(string driveLetter)
    {
        try
        {
            this._driveLetter = driveLetter.ToUpper();
            this._driveLetter += ":";
            matchDriveLetterWithSerial();
            return this._serialNumber;
        }
        catch
        {
            Logger.LogEvent("###_getSerialNumberFromDriveLetter() ERROR"); Security.SSPExit(); return this._serialNumber;
        }
    }

    private void matchDriveLetterWithSerial() // ищем цель среди устройств...

    private string parseSerialFromDeviceID(string deviceId) // парсим ИД устройства...

    private string getValueInQuotes(string inValue) // парсим значение для получения серийника...
}
```

Рис. 2.14. Механізм аналізу підключеного пристрою

Шукаючи ефективний метод стеження за пристроями вводу (миша, клавіатура) автор знайшов відповідне рішення на GitHub. Суть знайденого рішення полягає у підписці на цільові події у системі. Коли вони відбуваються – операційна система повертає сингал про подію. Програмно можна прив'язати свій метод до будь-якої події, і при її виконанні автоматично спрацьовуватиме прив'язаний до неї метод. Такий підхід до відстеження подій називають хуком.

```
        // Константы WM_*
        public enum WindowsMessage ...

        public delegate int KeyboardHookProc(int code,
            WindowsMessage wParam, ref KeyboardHookStruct lParam);

        public delegate int MouseHookProc(int code,
            WindowsMessage wParam, ref MouseHookStruct lParam);

        [DllImport("user32")]
        public static extern int CallNextHookEx(IntPtr hHk, int nCode,
            WindowsMessage wParam, ref KeyboardHookStruct lParam);

        [DllImport("user32")]
        public static extern int CallNextHookEx(IntPtr hHk, int nCode,
            WindowsMessage wParam, ref MouseHookStruct lParam);

        [DllImport("user32")]
        public static extern IntPtr SetWindowsHookEx(WindowsHook idHook,
            KeyboardHookProc lpfn, IntPtr hMod, uint dwThreadId);

        [DllImport("user32")]
        public static extern IntPtr SetWindowsHookEx(WindowsHook idHook,
            MouseHookProc lpfn, IntPtr hMod, uint dwThreadId);

        [DllImport("user32")]
        public static extern bool UnhookWindowsHookEx(IntPtr hHk);
    }
}

// Дескрипторы хуков
private readonly IntPtr _keyboardHookHandle;
private readonly IntPtr _mouseHookHandle;

// Хуки
private readonly WinAPI.User32.KeyboardHookProc _keyboardCallback;
private readonly WinAPI.User32.MouseHookProc _mouseCallback;

// События
public event KeyEventHandler KeyDown = (s, e) => { };
public event KeyEventHandler KeyUp = (s, e) => { };
public event MouseEventHandler MouseButtonDown = (s, e) => { };
public event MouseEventHandler MouseButtonUp = (s, e) => { };
public event MouseEventHandler MouseMove = (s, e) => { };

public GlobalHook()...
```

Рис. 2.15. Механізм стеження за пристроями вводу

Розробимо логіку блокуючих форм. Додамо до проекту нову форму, назвемо її LockScreen. Цей клас служитиме шаблоном, який буде наслідуватись таку кількість разів, скільки у системі фізичних моніторів. Для кожного монітору кожен екземпляр буде приймати параметри, у відповідності із параметрами моніторів. Блокуючі форми займатимуть увесь простір на відповідному моніторі, їх не можна буде згорнути чи відкрити стороннє вікно поверх них. Інтерфейс всередині форм буде динамічно пристосовуватись під параметри моніторів, навіть якщо монітори змінюватимуться у процесі роботи додатку. Розмістимо на формі графічні елементи: чотири компоненти для зображень, два лейбли, текстове поле, кнопку. Змінимо кольорову гамму, розмістимо елементи у тому вигляді, у якому вони мають бути на єкрані блокування. Отримаємо наступний шаблон:

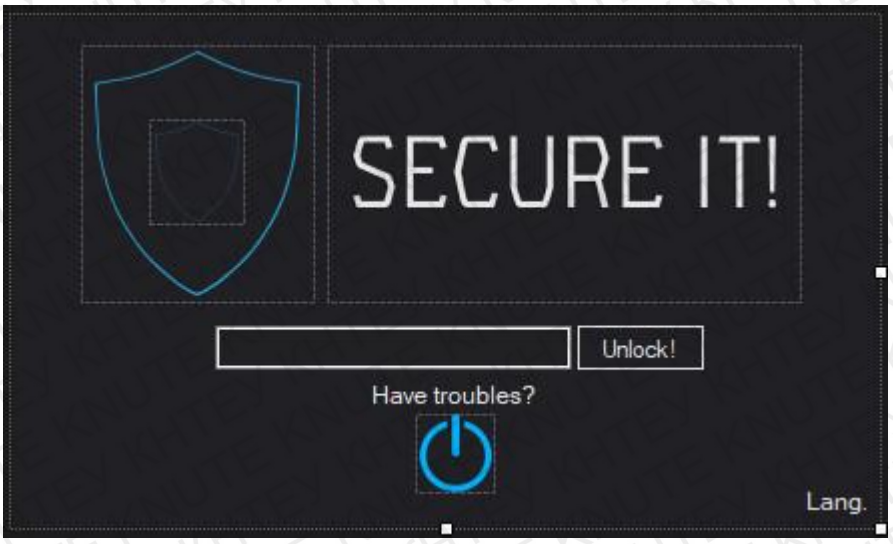

Рис. 2.16. Готовий шаблон блокуючих форм

Напишемо бізнес-логіку у шаблоні (код також буде наслідуватись, відрізнятиметься лише зовнішній вигляд), пропишемо сигнатури для стеження за мишею, унаслідуємо об’єкти для хуку пристроїв вводу, а також реалізуємо періодичне стеження за вікнами. Створимо логіку розділу блокуючих форм на головну та другорядні. Головна форма буде на головному моніторі, і через неї користувач зможе перейти у режим розблокування за допомогою пароля, а на другорядних залишимо лише

логотип. Всі непотрібні деталі інтерфейсу знищимо. Розмістимо набір методів для перевірки паролю, додамо таймер для перехоплення керування мишею, а також таймер для блокування через бездіяльність користувача (на випадок, коли користувач перейшов у режим розблокування через пароль, і покинув робоче місце). У автора вийшов інтерфейс, що періодично виконує безліч перевірок, і прийме захисні заходи, якщо це буде потрібно. Всі механізми блокуючих форм виглядають так:

```
[return: MarshalAs(UnmanagedType.Bool)]
internal static extern bool GetCursorPos(ref Win32Point pt);

[StructLayout(LayoutKind.Sequential)]
internal struct Win32Point...

public static Point GetMousePosition()...
// -----GetMousePosition().X\Y.ToString();---------------------------------------------------- ...


[DllImport("user32.dll")]
static extern IntPtr GetKeyboardLayout(int idThread);
[DllImport("user32.dll", SetLastError = true)]
public static extern int GetWindowThreadProcessId([In] IntPtr hWnd, [Out, Optional] IntPtr lpdwProcessId);
[DllImport("user32.dll", SetLastError = true)]
public static extern IntPtr GetForegroundWindow();

static ushort GetKeyboardLayout()...

// int id = GetKeyboardLayout(); ...


private void LockScreen_Load(object sender, EventArgs e)...

private void timer1_Tick(object sender, EventArgs e)                    // не подпускаем мышь ...

private void LockScreen_FormClosing(object sender, FormClosingEventArgs e)      // при попытке закрытия ...

private void label3_MouseEnter(object sender, EventArgs e)              // цвет лейбла (косметика)...

private void label3_MouseLeave(object sender, EventArgs e)              // цвет лейбла (косметика)...

private void label3_Click(object sender, EventArgs e)                   // показываем поля ввода...

private void pictureBox1_DoubleClick(object sender, EventArgs e)        // на случай, если пошло по пи

private void pictureBox3_Click(object sender, EventArgs e)              // L выключение компа...

private void button1_Click(object sender, EventArgs e)                  // L разблокиров-очка...

private void textBox1_KeyDown(object sender, KeyEventArgs e)            // L разблокиров-очка...

private void PassUnLockAction()                                         // L разблокиров-очка...

private void Delay_For_Pass_Unlocking_Tick(object sender, EventArgs e)  // L при разблокировке прл

private void LockScreen_MouseEnter(object sender, EventArgs e)...
```

Рис. 2. 17. Механіка роботи блокуючих екранів

2.5. Висновки до розділу 2

У сучасному світі технології для захисту інформації є надзвичайно затребуваними. Автор доклав значних зусиль для розробки додатку для захисту персональних комп'ютерів від несанкціонованого доступу, отримавши багато корисного досвіду та професійних навичок. Усього було написано одинадцять класів програмного коду, на що розробник витратив три місяці, і ще один місяць на тестування готового продукту. Як і планувалось, розробка додатку була завершена до початку 2020 року. Додаток було протестовано на декількох комп'ютерах, на кожному із яких проблем у роботі програми виявлено не було. Також розробник тестував програму із використанням різних флеш-накопичувачів одночасно, і прийшов до висновку, що вірогідність колізій дуже мала, а отже додаток працює більш ніж надійно. Враховуючи теперішній стан проекту, розробнику варто дописати деякі механізми (а саме методи роботи з мережевими адаптерами, візуальний аналіз зображення на екрані, контроль за подіями у системі) та розробити захист ліцензії, щоб довести продукт до товарного вигляду та захистити його від піратства. Використовуючи цю систему варто пам'ятати, що вона не дасть повного захисту від зловмисників. Її потрібно комбінувати із такими інструментами як «BitLocker» та «shadow defender», а також приймати відповідні організаційні заходи. Також не варто забувати, що який би захист не встановили спеціалісти із кібербезпеки, рано чи пізно хакери знайдуть спосіб його обійти, однак технології розвиваються надзвичайними темпами, і з часом подібні системи також покращуватимуться. Враховуючи це, автор потурбувався про зручність модифікації системи, створивши зручні класи із зрозумілими методами та структурованим кодом. У процесі масової інформатизації на підприємствах люди все більше потребують застосування технологій для зберігання, обробки та безпечного доступу до конфіденційної інформації.

РОЗДІЛ 3.
ОПИС ЗАПРОПОНОВАНОГО ПРОГРАМНОГО РІШЕННЯ ДЛЯ ЗАХИСТУ ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА ВІД НЕСАНКЦІОНОВАНОГО ДОСТУПУ

3.1. Огляд готового програмного продукту для захисту персонального комп'ютера від несанкціонованого доступу

Автор закінчив розробку та тестування додатку, згідно технічному завданню, що було складено ще на початку концептування проекту. Якщо ще раз проаналізувати характеристики схожих за тематикою додатків-конкурентів – прийдемо до висновку, що система автора не поступається їм у багатьох важливих параметрах:

	Простота та зрозумілість інтерфейсу	Програмний Persistence	Зберігання ключа на носії	Механізм самовідновлення
Predator	-	+	-	+
USB System Lock	-	-	-	+
WinLockr	-	+	+	-
Secure IT	+	+	+	+

Рис. 3.1. Порівняння проекту із конкурентами

Як бачимо, проект автора має масу переваг. Перейдемо до безпосередньої демонстрації проекту. Скомпілюємо останню версію проекту. Відкриємо .exe файл, що постачається розробником і побачимо, як плавно відпрацьовує косметична процедура для плавного відображення інтерфейсу додатку. Після повного завантаження програми спостерігаємо наступне вікно:

Рис. 3.2. Зовнішній вигляд стартового інтерфейсу

Дане вікно є не лише засобом для реєстрації носія, а й візитівкою програми. Автор доклав зусиль у піклуванні про користувача, тому інтерфейс вийшов досить простим та зрозумілим. Також у інтерфейсі присутній механізм Press&Drug. Це означає, що даний елемент інтерфейсу можна переміщати не лише за верхню частину вікна, а й за будь-яке інше місце.

Підключимо флеш-носій до комп'ютера, оберемо його у списку, придумаємо пароль та поставимо мітку у чек боксі згоди з правилами користування. При введенні паролю бачимо, як поле вводу підсвічується зеленим. Це зроблено для того, щоб сповістити користувача про мінімальну кількість необхідних символів у полі. Другий чек бокс підсвічується зеленим, коли паролі у обох полях співпадають. Коли всі необхідні дані введено, отримаємо наступний вигляд:

Рис. 3.3. Реакція інтерфейсу на заповнення даними

Натиснемо кнопку «Go!». У цей час програма виконає встановлення міток щодо захищеності системи, проведе копіювання файлів, шифрування авторизаційних даних, запис параметрів до регістру. При успішному виконанні всіх процедур програма сповістить користувача про результат наступним повідомленням:

Рис. 3.4. Сповіщення користувача про успішне встановлення захисту

Починаючи з цього моменту програма встає на захист системи. Щоб перевірити, чи працює захист, необхідно відключити флеш-носій від комп'ютера, на що програма зреагує, і заблокує систему. Розблокування відбувається автоматично, при підключенні того пристрою, що був зареєстрований у програмі. Усі інші флеш-пристрої програма ігнорує. Відключимо зареєстрований пристрій, і зображення на екрані зміниться:

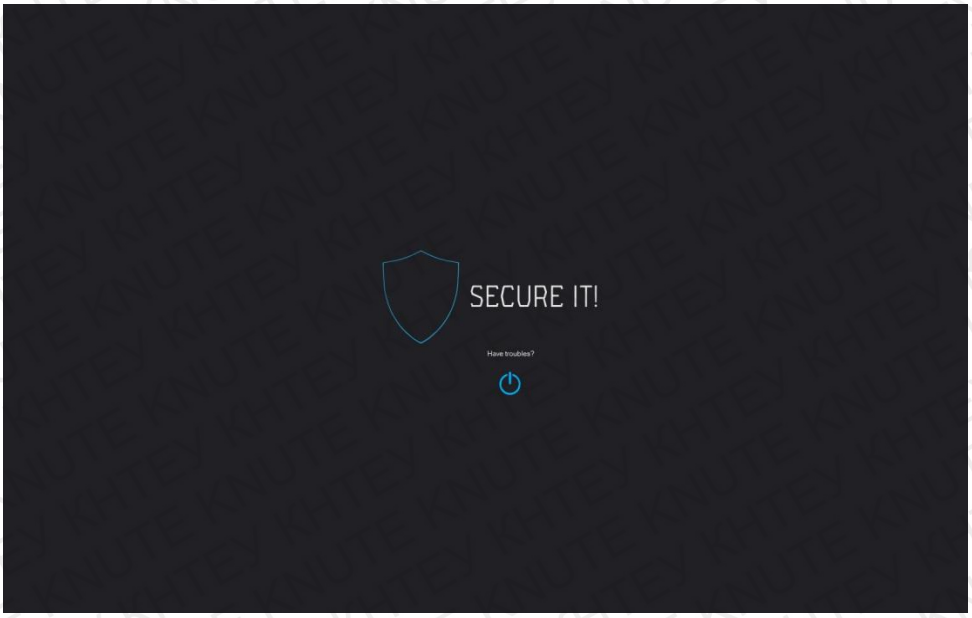

Рис. 3.5. Демонстрація результату блокування

Автор звертає увагу на те, що навіть якщо до комп'ютера підключено декілька моніторів, блокуючі екрани відобразяться на кожному з них. Кожен блокуючий екран коректно буде вписаний у свій монітор, а інтерфейс у них займе центральне положення. На головній формі – логотип, назва проекту, лейбл та кнопка вимкнення. На другорядних – лише логотип. Вказівник миші не може покинути невелику робочу область головного монітору, а небажані натиски перехоплюються програмою. Ось так виглядає блокування на комп'ютері із двома моніторами:

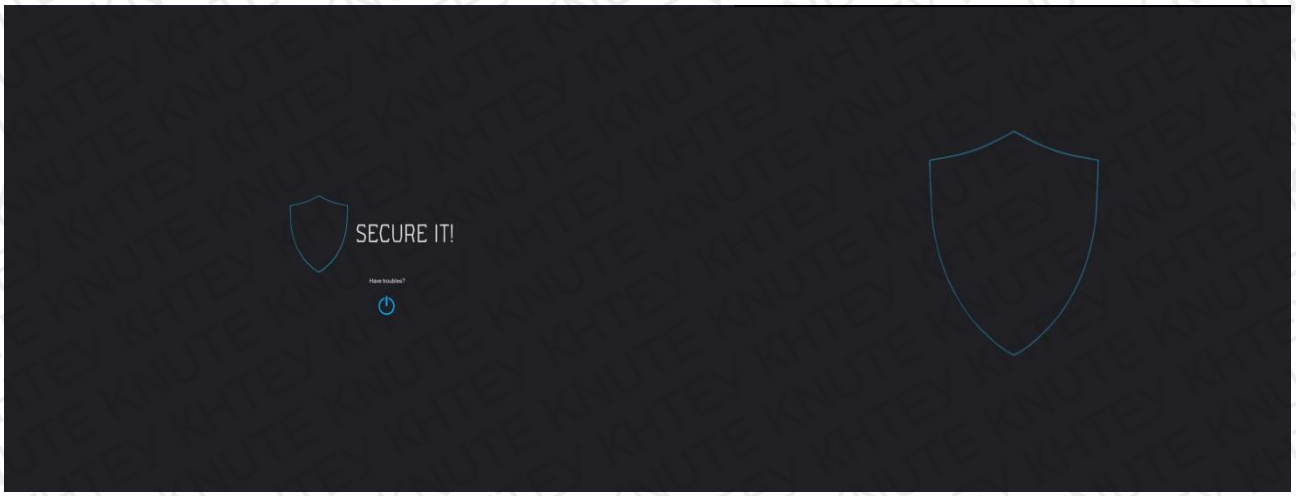

Рис. 3.6. Демонстрація результату блокування на двох моніторах

Як автор описав раніше – блокування можливо зняти за допомогою флешки, що зареєстрована у програмі. Та що робити, якщо користувач втратив носій? Саме для цього програма запрошувала пароль (Рис. 3.3). Щоб отримати можливість ввести пароль, користувач повинен натиснути на лейбл «Have troubles?» на головному екрані. Після цього у пустому просторі біля напису з'явиться поле для вводу паролю. Якщо пароль введено правильно – користувачу буде даний вибір: розблокувати одноразово, чи скинути захист. У першому випадку програма знімає блокування до натиску на іконку програми \ перезапуску системи \ бездіяльності користувача на протязі десяти хвилин. У другому випадку програма видалить дані про флеш-носій і запропонує користувачу зареєструвати новий. Виглядає це так:

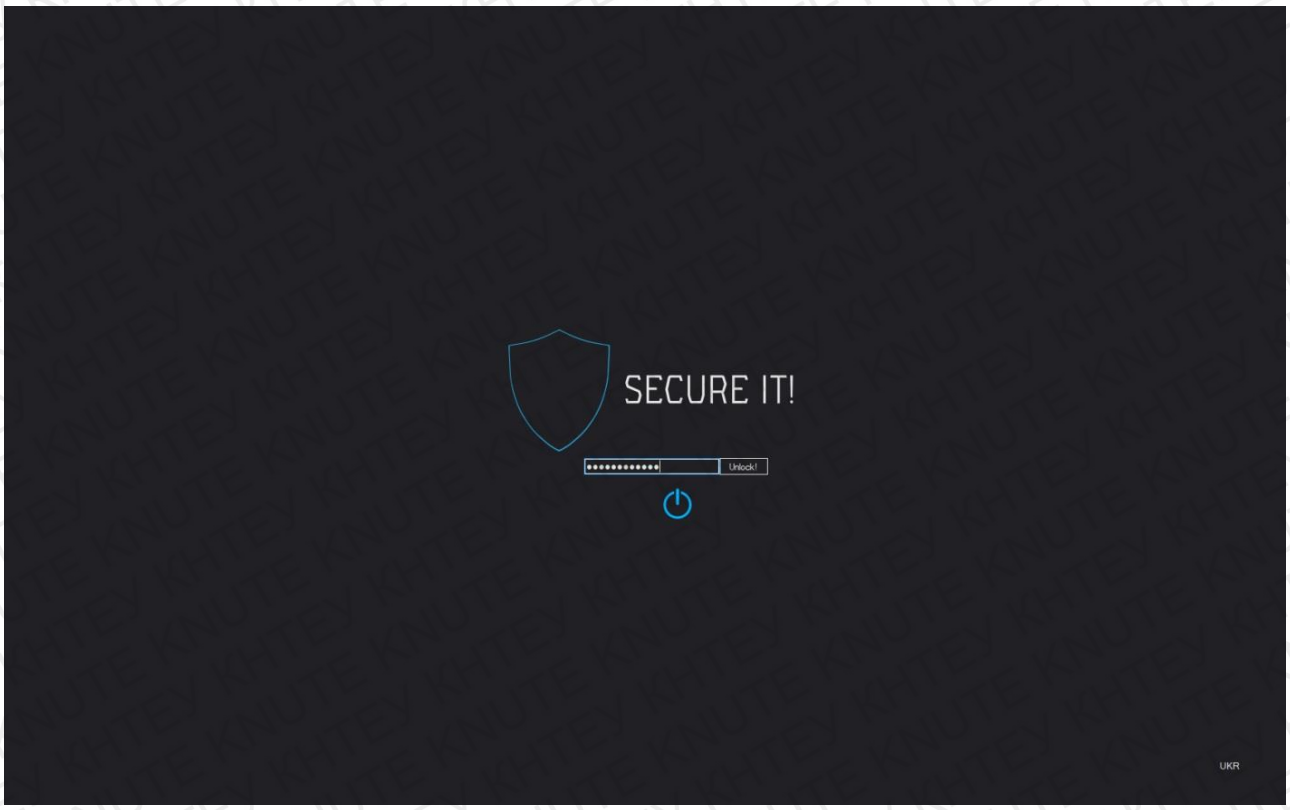

Рис. 3.7. Демонстрація вводу паролю

Рис. 3.8. Демонстрація механізму доступу за допомогою паролю

3.2. Висновки до розділу 3

Сучасний ІТ ринок вимагає від розробників швидкої та якісної реалізації програмних продуктів і зменшення грошових витрат, що направлені на розробку. Щодо якості – вона має бути не нижчою, ніж у аналогічних продуктів, або навіть вищою. Також необхідно враховувати характеристики зручності, надійності й безпеки. Саме за такими принципами було створено рішення захисту персональних комп'ютерів від несанкціонованого доступу на мові C#. Працюючи над програмою, автор зіткнувся з необхідністю вирішення наступних проблем:

- Проектування й документування складних та розгалуджених архітектур ПЗ;
- Аналіз та синтез алгоритмів для сканування пристроїв у системі;
- Генерація простих та дієвих ідей для забезпечення максимальної ефективності роботи додатку, а також їх реалізація;
- Розробка механізмів для безпечного зберігання та доступу до інформації;
- Багатопотоковість у програмуванні;
- Тестування, аудит недоліків, дорозробка проекту.

Додаток вийшов досить дієвим, зручним та безпечним. Автор впевнений, що даний етап розробки цього програмного забезпечення не є останнім. Планується подальша розробка додатку до стану повнофункціональної захисної системи з безліччю додаткових захиних механізмів, щоб досягти запланованих економічних результтів, а також підняти рейтинг продукту у очах потенційних користувачів. Даний програмний продукт у майбутньому буде включати в себе інструменти для контролю над мережевими адаптерами, візуального аналізу картинки на головному та другорядних дисплеях й механізм повного контролю над USB портами. Розробник пройшов складний та цікавий шлях, шо стало для нього надзвичайно корисним досвідом.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Темою дипломного проекту поставлено завдання створити та задокументувати зручну та якісну систему для захисту персональних комп'ютерів від несанкціонованого доступу. Автор виконав технічне завдання більш ніж повністю. На початку розробки планувалось, що цей додаток включатиме лише мінімально необхідний функціонал для даного дипломного проекту, але пізніше автор помітив, що ринок подібних програм залишає бажати кращого. Виходячи з ситуації на ринку, було проведено глибокий аналіз функціоналу та основних характеристик схожих додатків. У результаті аналізу технічне завдання зазнало значних змін, і збільшилось приблизно вдвічі. Початкова архітектура також зазнала змін: реляційна модель взаємодії всередині програми стала значно складнішою та розгалуженішою, а кількість класів зросла втричі. Автору сподобалась задумка проекту, і з'явилась ідея, яка переросла у виклик по створенню найкращого продукту на ринку, ціллю якого було створити більш зручне та якісне програмне рішення, і не допустити помилок конкурентів. Щоб задовольнити свої наміри, автор шукав додаткові рішення для найскладніших механізмів на Github, Cyberforum, stackoverflow та адаптував їх під свої потреби. Додаток розроблявся більш ніж три місяці, і тестувався ще місяць. У процесі написання програми автор ще раз переконався, що розробка подібних програм на платформі C# не обмежується додатками Windows Forms, робота з кодом є надзвичайно зручною та приємною для розробника, а вбудовані механізми мови спрощують розробку у рази: класи та об'єкти просто і зрозуміло вписуються у загальну структуру, що масштабується у зручне дерево проекту, а графічні елементи можна налаштувати на свій смак.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Основний:

1. Мартин Дж. Организация баз данных в вычислительных системах. / Дж. Мартин – М.: Мир, 1980. – 662 с.

2. Мейер Д. Теория реляционных баз данных / Д. Мейер – М. : Мир, 1987.. – 608 с.

3. Тиори Т. Проектирование структур баз данных: в 2-х кн. Кн. 2. / Т. Тиори, Дж. Фрай – М. : Мир, 1985. – 320 с.

Додатковий

4. Андколи Ананд. Oracle9i для Windows : пер. з англ. / Ананд Андколи, Рама Велпури – К. : «Лори», 2006. – 498 с.

5. Базы данных: Учебник для вузов / Под ред. А. Д. Хоменко, авт. А. Д. Хомен-ко, В. М. Цыганков, М. Г. Мальцев – СПб. : Корона принт, 2009. 736 с.

6. Васвани В. Zend Framework. Разработка веб-приложений на PHP / В. Васвани – СПб: Питер,2012.-432 с.

7. B. Albahari. C# 7.0 in a nutshell: The Definitive Reference 1st Edition / B. Albahari, J. Albahari, O'Rielly media:2017. – 1090 p.

8. Гаврилова Т. А. Базы знаний интеллектуальных систем / Т. А. Гаврилова, В. Ф. Хорошевский / СПб: Питер, 2000.-384 с.

9. Гольцман В. MySql 5.0. Библиотека программиста. (Интерфес с PHP, Perl, Java) / В. Гольцман - Спб: БХВ-Перербург, 2005. – 252с.

10. Давыдов С. В. IntelliJ IDEA. Профессиональное программирование на Java. / С. В. Давыдов, А. А. Ефимов – СПб.: БХВ – Петербург, 2005. – 800с.

11. Дейт К. Дж., Введение в системы баз данных, 8-е издание: Пер. с англ. / К. Дж. Дейт – К.; М.; СПб.: Издательский дом «Вильямс», 2005. – 1328 с.

					КНТЕУ 121 07-13.БР		
Зм.	*Аркуш*	*докум*	*Підпис*	*Дата*			
Зав.кафедри		*Криворучко О.В.*			*Розробка системи захисту персонального комп'ютера від несанкціонованого доступу*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		*Савченко Т.В.*				*СД*	*46*	*46*
Гарант		*Цензура М.О.*						
Розроб.		*Поляков Я.С.*			*Список використаних джерел*	*Факультет інформаційних технологій, 4 курс, 7 група*		

ДОДАТКИ

Додаток А

Лістинг головного захисного класу

```
using System.Diagnostics;
using System.IO;
using System.Windows.Forms;
using System.Security.Cryptography;
using System.Text;
using Microsoft.Win32;
using System;
namespace SSP
{
    internal class Security // В этом классе все действия по работе с флэшками, шифрование, проверки валидности || методы, помеченные "*" - не используется или не завершены
    {
        //-------------------------свернуть все окна--------------------------------
------
        [DllImport("user32.dll", EntryPoint = "FindWindow", SetLastError = true)]
        static extern IntPtr FindWindow(string lpClassName, string lpWindowName);
        static extern IntPtr SendMessage(IntPtr hWnd, Int32 Msg, IntPtr wParam, IntPtr lParam);
        const int WM_COMMAND = 0x111;
        const int MIN_ALL_UNDO = 416;
        //IntPtr lHwnd = FindWindow("Shell_TrayWnd", null);                    // свернуть все окна
        //SendMessage(lHwnd, WM_COMMAND, (IntPtr)MIN_ALL,
```

```
        IntPtr.Zero);      // свернуть все окна
            //IntPtr lHwnd = FindWindows("Shell_TrayWnd", nul);                 //
развернуть  все окна
            //SendMessage(lHwnd,  WM_COMMAND,  (IntPtr)MIN_ALL_UNDO,
IntPtr.Zero); // развернуть  все окна
            //-------------------------------------------------------------------------------
------
            /// L - logging procedure
            /// !L - Not logging procedure
            internal  static  DriveInfo[]  TakeDrives()                          // L
Возвращает массив с инфой о каждом usb накопителе
            {
          DriveInfo[] All_Drives = DriveInfo.GetDrives();                 //получаем
данные по всем носителям
               DriveInfo[]          Removable_Drives          =          new
DriveInfo[All_Drives.Length];     //В этот массив кладём только извлекаемые
(usb), выборка из All_Drives
                  int counter = 0;
               try // если флешка не готова к чтению, этот блок отловит
ошибку
               {
                  foreach (DriveInfo drive in All_Drives)
                  {
                     //если находим флешку...
                     if (drive.DriveType == DriveType.Removable & drive.IsReady)
// определяем тип
                     {
                        Removable_Drives[counter] = drive;
                        counter++;
```

```
        }
            }
            return Removable_Drives;
        }
        catch
        {
            Logger.LogEvent("###_Usb_scanning ERROR");
            return Removable_Drives;
        }
    }
        internal static void CheckMonitors()                    // L
перезапускает, если мониторы изменились
    {
        try
        {
            Screen[] Current_Screens = Screen.AllScreens;          // массив
с текущими мониторами в системе
            if (Current_Screens.Length != Form1.All_Screens.Length) // если
количество мониторов изменилось = перезапуск (т.к. нужно переопределить
массив ммониторов)
            {
                foreach (LockScreen ls in Form1.Lock_Forms) // уничтожаем
блокирующие формы
                {
                    ls.Dispose();
                }
                EnableTaskManager();
                Logger.LogEvent("Monitors_changed!_rebooting...");
                Process.Start(Setup.Shelter + Setup.FileName); //
```

```
перезарускаемся
                SSPExit();
                Application.Exit();
            }
            for (int i = 0; i < Current_Screens.Length; i++)        // если количество мониторов не изменилось, но начальные не соответствуют текущим  = перезапуск
            {
                if ((Current_Screens[i] != Form1.All_Screens[i]) || (Current_Screens[i].Primary != Form1.All_Screens[i].Primary) || (Current_Screens[i].WorkingArea != Form1.All_Screens[i].WorkingArea))
                {
                    Form1._Locked = false
                    foreach (LockScreen ls in Form1.Lock_Forms) // уничтожаем блокирующие формы
                    {
                       ls.Dispose();
                    }
                    EnableTaskManager();
                    Logger.LogEvent("Monitors_changed!_rebooting...");
                    Process.Start(Setup.Shelter + Setup.FileName); // перезарускаемся
                Application.Exit();
                }
            }
            catch
            {
               Logger.LogEvent("###_CheckMonitors() ERROR");
```

```
        }
    }
    internal static bool Is_Secured()                    // !L Проверяет,
защищена ли система (есть ли запись в реестре)
    {
      using (RegistryKey registryKey =
Registry.CurrentUser.OpenSubKey(@"SOFTWARE\\Licenses\\"))
      {
        if (registryKey == null)
        { return false; }

        string value = (string)registryKey.GetValue("SSP_Flash");

        if (value == null || value == "")
        {
          return false;
        }
        else return true;
      }
    }
    internal static bool Is_Runned()                     // !L Проверяет,
запущена ли система (если имя файла было изменено)
    {
      Process[] ProcessList =
Process.GetProcessesByName(Setup.ProjectName);          //выборка
процессов по имени
      foreach (Process process in ProcessList)
      {
        if (ProcessList.Length >= 1)
```

```
            {
                return true;
            }
        }
        return false;
    }
    internal static void AntiDouble()                    // !L убивает себя, если экземпляр уже запущен
    {
        Process[] ProcessList = Process.GetProcessesByName(Setup.ProjectName);                    //выборка процессов по имени
        foreach (Process process in ProcessList)
        {
            if ((ProcessList.Length > 1) && (process.Id == Process.GetCurrentProcess().Id))  // убийство себя, если есть экземпляр
            {
                if (Form1.start_param == @"-Shortcut" && Form1.IsSecured)
                {
                    MessageBox.Show("System Secured! Use lockscreen to reset flash-key!", Setup.ProjectName);
                }
                else if (Form1.start_param == @"-Shortcut" && !Form1.IsSecured)
                {
                    MessageBox.Show("System already runned!", Setup.ProjectName);
                }
                EnableTaskManager();
```

```
            Form1._Locked = false;
                SSPExit();
            }
        }
    }
    internal static void SSPExit()                    // !L правльное и
безоговорочное убийство себя
    {
        Process[] ProcessList = Process.GetProcesses();
//выборка процессов по имени
        foreach (Process process in ProcessList)
        {
            if (process.Id == Process.GetCurrentProcess().Id)  // убийство
себя
            {
                Form1._Locked = false;
                EnableTaskManager();
                process.Kill();
            }
        }
    }
    private static string GetFlashSerial(string DriveLetter)          // !L
возвращает серийник флешки по букве
    {
        string serial = "";
        if (DriveLetter != "")
        {
            USBSerialNumber usb = new USBSerialNumber();
            serial = usb.getSerialNumberFromDriveLetter(DriveLetter);
```

```
        }
        else
        {
            serial = "Unknown Device";
        }
        return serial;
    }
    internal static void AddTrustFlash(string letter, string pass)  // L добавляет запись в реестр о *правильной флешке*
    {
        try
        {
            foreach (DriveInfo drive in TakeDrives())
            {
                if (drive != null && drive.Name == letter)  // определяем выбранную флешку среди доступных
                {
                    string FlashHash = GetHashFromString(GetFlashSerial((drive.Name).Substring(0, 1)) + drive.TotalSize.ToString() + drive.DriveFormat);  // хеш флешки;
                    string PassHash = GetHashFromString(pass); // хеш пароля;
                    // создаём подраздел для хранения хешей
                    RegistryKey KEY_CREATE = Registry.CurrentUser.CreateSubKey("SOFTWARE\\Licenses\\");
                    // кладём хешы в реестр
                    RegistryKey Key = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Licenses\\", true);//
```

```
        открываем нужную ветку в реестре
                    Key.SetValue("SSP_Flash",                    FlashHash);
// название ключа, значение
                    Key.SetValue("SSP_Pass", PassHash);
                    Key.Close();                                              //
записываем                                                  // записываем
                    if (Directory.Exists(Setup.TempShelter))
                    {
                       File.Delete(Setup.TempShelter + "Stop.zz");
                    }
                    Logger.LogEvent("### Trust_Flash_Registred!");
                    MessageBox.Show(Success! Use your flash device to unlock.
Use master-password, if you have troubles with flash-unlocking.\n\nImportant!
Your flash-unlocker will be work only in current USB port!, Setup.ProjectN,
MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
            }
        }
        catch
        {
            Logger.LogEvent("###_AddTrustFlas() ERROR" );
            MessageBox.Show("AddTrustFlash()  ERR!",  Setup.ProjectName,
MessageBoxButtons.OK, MessageBoxIcon.Error);
            SSPExit();
        }
    }
        internal static string GetPassHashFromReg()                  // L
возвращает хеш пароля из реестра
        {
```

```
            try
            {
                RegistryKey PassHash = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Licenses\\", true);// открываем нужную ветку в реестре
// название ключа, значение
                string value = PassHash.GetValue(SSP_Pass).ToString();
                PassHash.Close(); // Читаем
                return value;
            }
            catch
            {
                Logger.LogEvent("###_Registry->PassHash_not_accessible!");
                return "";
            }
        }
        internal static string GetFlashHashFromReg() // L возвращает хеш флешки из реестра
        {
            try
            {
                RegistryKey FlashHash = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Licenses\\", true);// открываем нужную ветку в реестре
// название ключа, значение
                string value = FlashHash.GetValue("SSP_Flash").ToString();
                FlashHash.Close(); // Читаем
```

```
            return value;
        }
        catch
        {
            Logger.LogEvent("###_Registry->FlashHash_not_accessible!");
            return "";
        }
    }
    internal static void RemoveTrustFlash(bool stop)          // L удаляет запись из реестра о *правильной флешке*
    {
        try
        {
            // удаляем данные флешки и пароля
            RegistryKey key = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Licenses\\", true); // << открываем ключ с правами на запись
            key.DeleteValue("SSP_Flash");
            key.DeleteValue("SSP_Pass");
            key.Close();
            // удаляем данные автозагрузки
            RegistryKey Key = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\", true);
            Key.DeleteValue(Setup.ProjectName);
            Key.Close();
            if (stop)
            {
                SSPExit();
```

```
                }
                if (Directory.Exists(Setup.TempShelter))
                {
                   // stop persistance
                   File.Create(Setup.TempShelter + "Stop.zz");
                }
                Form1._Locked = false;
                EnableTaskManager();
                IntPtr lHwnd = FindWindow("Shell_TrayWnd", null);          //
развернуть  все окна
                SendMessage(lHwnd,                          WM_COMMAND,
(IntPtr)MIN_ALL_UNDO, IntPtr.Zero); // развернуть  все окна

                Logger.LogEvent("### Trust_Flash_Removed!");
                Application.Restart();
              }
              catch
              {
                Logger.LogEvent("###_RemoveTrustFlash()
ERROR...Restarting...");
                Process.Start(Setup.Shelter + Setup.FileName);
                SSPExit();
              }
          доступные флешки с целью обнаружения правильной, с
дальнейшей разблокировкой
          {
            DateTime start = DateTime.Now;
            string action = "";
            CheckMonitors();                          // если мониторы
```

```
изменились = перезапуск
            string HashFromReg = GetFlashHashFromReg();
            bool NeedLock = true;
            string FlashHash = "";
            foreach (DriveInfo drive in TakeDrive())
            {
               if (drive != null) // определяем доступные флешки
               {
                  try
                  {
                     FlashHash =
GetHashFromString(GetFlashSerial((drive.Name).Substring(0, 1)) +
drive.TotalSize.ToString() + drive.DriveFormat);  // хеш флешки;
                  }
                  catch
                  {
                     Logger.LogEvent("### Device_(" + drive.Name +
")_not_accessible!");
                     break;
                  }
                  if (FlashHash == HashFromReg) // если хеш совпал
                  {
                     Form1._UnLocked_With_Pass = false;
                     NeedLock =true;        // блочить не надо
                  }
               }
            }
            if(!Form1._Locked && NeedLock) // блочим
            {
```

```
            action = "Locked";
            Lock();
          }
          else if(Form1._Locked && !NeedLock) // разблочим
          {
            action = "Unlocked";
            UnLock();
          }
          if(action != "")
          {
            Logger.LogEvent(action + "(" + (DateTime.Now -
start).TotalMilliseconds.ToString("f") + "ms.)");
          }
        }
        internal static void Lock()
        {
          IntPtr lHwnd = FindWindow("Shell_TrayWnd", null);          //
свернуть все окна
          SendMessage(lHwnd, WM_COMMAND, (IntPtr)MIN_ALL,
IntPtr.Zero); // свернуть все окна
          Form1._Locked = true;                                      // ставим флаг
          Form1._UnLocked_With_Pass = false;
          DisableTaskManager();
          foreach (LockScreen ls in Form1.Lock_Forms) // показываем
блокирующие формы на всех экранах
          {
            new System.Threading.Thread(() => {
              ls.ShowDialog();
              ls.Focus();
```

```
            }).Start();
            }
        }
    // Блок\разблок
        internal static void UnLock()
        {
            IntPtr lHwnd = FindWindow("Shell_TrayWnd", null);                    //
развернуть  все окна
            SendMessage(lHwnd,  WM_COMMAND,  (IntPtr)MIN_ALL_UNDO,
IntPtr.Zero); // развернуть  все окна
            Form1._Locked = false;                                  // ставим флаг
            EnableTaskManager();
            Cursor.Position                                =                    new
System.Drawing.Point(System.Windows.Forms.Screen.PrimaryScreen.Bounds.Wid
th / 2, System.Windows.Forms.Screen.PrimaryScreen.Bounds.Height / 2);
            foreach (LockScreen ls in Form1.Lock_Forms)
            {
               new System.Threading.Thread(() => {
                  ls.Hide();
               }).Start();
            }
        }
        internal  static  string  GetHashFromString(string  input)              // !L
возвращает SHA384 из строки
        {
            var sha = SHA384.Create();
            return
System.Convert.ToBase64String(sha.ComputeHash(Encoding.UTF8.GetBytes(inp
ut)));
```

```
            // 512 mtcEDi8E2IENcowLPGcafka7Ef++FQxA / 0eJMZs6GedPeoSPqqKNIfJ6wKpHbLbtFEMJMCXNzibwaNRVztV0Hw ==
            // 384 yWzy8GbaTNt3DAOYdYwKK5lNOcu3TH2x2zB/RevH33Oa46wJCPd+N8FRlk8pLpKh
            // 256 Llrrcy1ypssu9Eix9dDv / kosdghyv4phnj / HBDnaZwk =
        }
        private static void DisableTaskManager()
        {
            try
            {
                RegistryKey RKey = Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System", true);
                RKey.SetValue("DisableTaskMgr", 1);
                RKey.Close();
            }
            catch
            {
            }
        }
        // откл\вкл диспетчер
        internal static void EnableTaskManager()
        {
            try
            {
                Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System", true);
                RKey.SetValue("DisableTaskMgr", 0);
```

```
        RKey.Close();
            }
            catch
            {
            }
    }
```