# Общие рекомендации

Под надежным прикрытием автоматической защиты вы можете строить свою внутреннюю защиту на основе Guardant API. При этом целесообразнее следовать следующим советам:

1. **Не храните в явном виде коды доступа к ключу**. Не следует хранить в «чистом» виде Личные коды, по которым производится доступ к электронному ключу. Для того чтобы осложнить хакерам их поиск в теле приложения, Личные коды можно закодировать (например, при помощи операции XOR). Непосредственно перед вызовом функции API раскодируйте их, а после того как функция отработала – сразу же удаляйте из памяти раскодированный вариант Личного кода. Также можно хранить Личные коды частями в разных переменных.

2. **Не храните «лишние» коды доступа в защищенном приложении**. Например, если приложение использует только чтение из ключа и запуск его аппаратных алгоритмов, нужно хранить в приложении только Личный код для чтения (Private Read Code). Остальные Личные коды не будут нужны, а их присутствие в теле приложения может сильно облегчить хакеру задачу получения доступа к ключу.

3. **Располагайте вызовы функций API по всему телу приложения**. Это резко увеличивает объем кода, который надо изучить хакеру для локализации кода проверок и его модификации.

4. Организовывайте взаимодействие с электронным ключом из **разных потоков**. Многопоточные приложения гораздо труднее отлаживать, а значит и взламывать. При этом полезно усложнить взаимодействие между потоками, организуя его так, чтобы потоки выполняли часть логики приложения, и нельзя было их просто остановить или завершить.

5. Очень хорошим приемом являются **«вероятностные» вызовы**, когда функция API вызывается не всегда, а с какой-то вероятностью – например 1/7. В разных местах приложения можно использовать различные вероятности вызовов (причем в жизненно важных местах приложения вероятность может быть больше, в менее важных местах – меньше). В результате хакер может попросту не найти те вызовы, которые происходят с малой вероятностью – и тогда, якобы «взломанное», приложение рано или поздно преподнесет сюрприз нелегальным пользователям.

6. **Усложните логику обработки кодов возврата функций API**. Если вы будете проверять код возврата функции API простым **сравнением**, хакеру не составит труда уничтожить это сравнение прямо в теле приложения – и таким образом снять защиту. Разработайте более сложную логику. Например, используйте коды возврата в качестве индексов каких-то массивов данных, стартовых значений для генераторов псевдослучайных чисел, используйте их в алгоритмах математических вычислений, реализованных в приложении и т. п. Таким образом, точка, в которой защищенное приложение принимает решение о своей дальнейшей работе, в чистом виде будет отсутствовать. Хакеру придется разбираться не только в логике работы защиты, но и в хитросплетениях процессов, происходящих в самом приложении. И защита станет полноправной частью приложения, без которой станут неверными его вычисления, будут использоваться не те данные и т. д.

7. **Откладывайте момент реакции приложения на коды возврата функций API**. Хорошо зарекомендовал себя прием, когда приложение принимает решение о своей дальнейшей работе не сразу по получении кодов возврата функции API, а значительно позже. Например, функция API вызывается при запуске приложения, а анализ возвращенных ею значений производится при выполнении пользователем каких-либо действий (открытие или сохранение файла, вызов меню настройки приложения и т. п.). В таком случае хакеру будет сложно проследить причинно-следственную связь поведения приложения. Кроме того, этот прием заставит хакера исследовать в отладчике большие фрагменты кода приложения.

8. **Ограничивайте возможности приложения, если электронный ключ не найден**. Этот прием можно использовать в дополнение к описанному выше. Если функция API возвращает ошибку, можно не принимать кардинальных решений относительно дальнейшей работы приложения, а лишь ограничить его возможности. Например, перестать давать возможность экспортировать файлы в другие форматы, сохранять настройки, распечатывать отчеты и т. п. Причем, в ответ на неудачу при вызове API из разных мест приложения можно «лишать» его все новых и новых возможностей, все более превращая в демонстрационное. Помимо важных, можно блокировать и несколько второстепенных возможностей – это увеличит вероятность того, что хакер попросту не заметит их отсутствия, и приложение в таком виде попадет на «черный рынок». Пользователей же такой «демо-версии» отсутствие части ее возможностей подтолкнет на приобретение легальной копии приложения.

9. **Подсчитывайте хэш** важных участков кода приложения. Это мощное средство защиты от модификации хакером кода приложения. Предварительно вы подсчитываете хэш тех мест в программе, в которых производятся вызовы функций API, анализируются выходные данные функций и принимаются решения о дальнейшей работе приложения. В процессе работы приложения вы еще раз производите подсчет этих участков – и получаете достоверную информацию о том, были ли эти участки нелегально модифицированы. Производить подсчет хэшей лучше в отдельном потоке, причем делать это следует значительно раньше или значительно позже того, как управление попадет в контролируемый участок кода. Один и тот же участок целесообразно контролировать из нескольких мест приложения. А реакция на факт модификации программного кода должна быть примерно такой же, как и на ошибку при вызове функции API. В результате хакер будет вынужден дополнительно искать по всему телу приложения места, в которых вы проверяете контрольные суммы. Для вычисления хэш-функции можно воспользоваться функцией Guardant API GrdHash.

10. **Меняйте логику работы приложения с модулями защиты**. Очень полезно бывает время от времени (оптимальный вариант – в каждой новой версии защищенного продукта) менять логику его работы с модулями защиты. Используйте новые приемы, подобные описанным здесь, по-иному комбинируйте их и т. п. – и тогда при выходе новой версии продукта хакер будет вынужден начинать работу по ее исследованию, что называется, «с нуля». Все его прежние наработки мгновенно теряют актуальность – ведь теперь приложение

взаимодействует с модулями защиты совершенно по-иному.

11. **Используйте обфускаторы**. Какой бы совершенный код защиты вы не написали, он почти всегда доступен хакеру для исследования посредством отладчиков, дизассемблеров. Есть специальные программы для затруднения анализа программ и понимания смысла того, что они делают. Их называют обфускаторами. Эти программы обрабатывают готовые приложения (exe-файлы) и модифицируют их таким образом, что стандартными средствами невозможно их понять, надо создавать специальные средства именно для данного обфускатора, а то и для данной защищенной программы (обфускаторы, как правило, полиморфны). Есть неплохие сторонние обфускаторы, которые можно использовать для этих целей. Дополнительно обфускаторы часто защищают приложения от изменения таким образом, что невозможно изменить в них даже один бит, сохранив работоспособность программы. Для взлома в таком случае приходится тоже создавать специальные инструменты, что долго и очень дорого.