

rus.bce_desktop_mobile_application

Alexandr Kirilov (<https://github.com/alexandrkirilov>)

Blockchain пример для Desktop/Mobile приложения.

Данный пример использования blockchain основан на опыте разработки десктоп-приложения основанного на базе "[Chromium Embedded Framework \(CEF\)](#)". Впервые данный подход был применен в 2010 году. На сегодняшний момент, решения основанные на этом принципе используются во многих других приложениях, в том числе и для смартфонов различных типов и производителей.

Данный пример специально упрощен для иллюстрации принципа по которому велась разработка и не является готовым решением. Готовое решение на прямую зависит от того какая платформа используется и какие задачи должно решать приложение. Задача всегда будет определять цепь (chain) элементов, которые должны быть заблокированы (block).

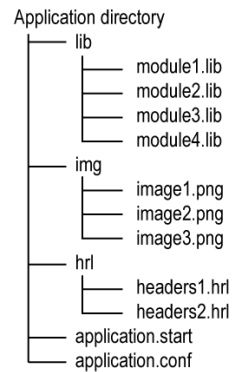
Задача: задача разработать клиентское приложение строго привязанное к серверу данных использующего в качестве пользовательского интерфейса web-технологии. Иными словами это было что-то типа браузера строго привязанного к одному сайту (серверу), с сильно увеличенным уровнем безопасности различными способами. Одним из таких способов была проверка целостности приложения при загрузке. Сейчас эту технологию называют blockchain, тогда, без малого 10 лет назад, это была просто проверка целостности.

Задача которую нужно реализовать при помощи blockchain - это обеспечить максимальную невозможность изменения файлов составляющих приложение (библиотеки, изображения, модули и т.д.), исключить возможность "инъекций" в модули приложения путем изменения библиотек максимально возможным способом.

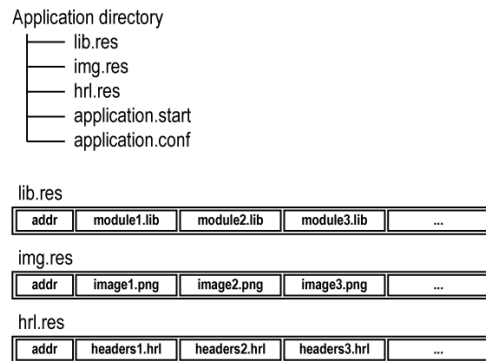
Возвращаясь к blockchain - это зафиксированная цепь элементов составляющих целое. В данном случае целое (цепь) - это приложение, файлы относящиеся к этому приложению - это элементы цепи которую мы собираемся блокировать. В свою очередь файлы это байты связанные в единую последовательность (цепь).

Иными словами задача может быть сформулирована так: последовательность (chain) байт составляющих файлы, которые в свою очередь составляют приложения должна быть заблокирована (blocked) и неизменна.

① Classic way



② Blockchain way



Классический подход при компиляции приложения это когда все файлы имеющие отношение к приложению расположены в каком-то каталоге с последующим делением на подкаталоги если это необходимо.

Уникальным является сам файл и уникальность может быть вычислена и зафиксирована при помощи функций MD5/4, но это будет константное значение которое всегда одинаковое. В случае с безопасностью любое статичное препятствие - потенциальная проблема, т.к. все что стоит неподвижно рано или поздно можно обойти.

Задача дополняется требованием об отсутствии константных значений при проверке целостности. Этого можно достичь при помощи объединения файлов необходимых для приложения в файлы коллекций, которые являются конкатенацией всех файлов в один организованный в определенной последовательности, где в начале файла расположена "адресная книга" которая указывает на то, где в теле файла расположено то или иное содержимое.





lib.res binary schema



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	...	N

resource file address book

 resorces_positions, adress(49,58)

resourced modules

 module1.lib, adress(49,58)  module2.lib, adress(59,70)  module3.lib, adress(71,87)  module4.lib, adress(88,98)

 module5.lib, adress(99,114)  module6.lib, adress(114,126)

Последовательность конкатенации файлов может быть различной:

- целой: когда файлы конкатенируются один за одним и позиция файла (номер начального байта и номер конечного байта) хранятся в адресном пространстве
- фрагментной: когда файлы делятся на отдельные фрагменты и в произвольном порядке конкатенируются а в адресное пространство записываются позиции фрагментов
- произвольная: это когда разработчик сам определяет как и что записывать в файл ресурсов

При компиляции приложения создаются файлы ресурсов, эталон которых кладется на сервер. И это является цепью которая заблокирована сервером. При авторизации или загрузке приложения сервер читает цепь эталона в произвольном порядке и вычисляет MD5 от произвольной последовательности байтов и сохраняет это значение а клиентскому приложению отправляет только номера байтов которые нужно прочесть и вычислить значение MD5 на стороне клиента которое должно совпадать с тем которое сохранено.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	...	N

Byte_sequence = 93+135+4+41+20+117+82
md5(Byte_sequence) = unique_vaue

Для увеличения безопасности данный процесс может итеративно повторяться несколько раз с различными последовательностями байтов для проверки.

Последовательность действий реализованная в приложении может выглядеть так:

- загрузчик приложения выполняет только чтение байтов и проверку приложения
- при прохождении авторизации, загрузчик, получает некий предкомпилированный модуль (в бинарном виде) с сервера, который встраивается в адресное пространство приложения и позволяет на основании "адресной книги" файла ресурсов устанавливать связи между модулями приложения, своего рода PROXY внутри приложения которое связывает элементы приложения между собой позволяя ему функционировать и при каждой загрузке адресное пространство данного PROXY новое.

Такой же принцип может быть использован для мониторинга памяти, когда память приложения представляется как цепь из элементов которая должна быть неизменна и в режиме реального времени и в произвольном порядке сканируется по принципу описанному выше.

Имплементация данного подхода в приложение сильно затрудняет возможность скомпрометировать приложение или как минимум сокращает время до момента распознавания попытки злонамеренных действий.

Еще раз нужно обратить внимание, что это не финальная реализация приложения, а только пример на основании которого можно создавать уникальное решение для какого-либо другого приложения. Перед использованием **blockchain** нужно четко понимать что есть элементы цепи (**chain**) которые должны быть зафиксированы (**block**).