eng.bce_desktop_mobile_application

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

Blockchain example. Desktop/Mobile application implementation.

This example of using blocked chains based on experience with desktop application developing where the "Chromium Embedded Framework (CEF)" was the core of application. For the first time this approach has been using almost 10 years ago, in 2009. For now it has been implementing in a few applications, including developing for different mobile platforms.

This example is simplified especially because of necessity to illustrate the blockchain approach but not the full featured solution for any cases. The final solution is totally depend on huge list of facts:

- platforms
- technical abilities of devices
- the goal of application
- etc

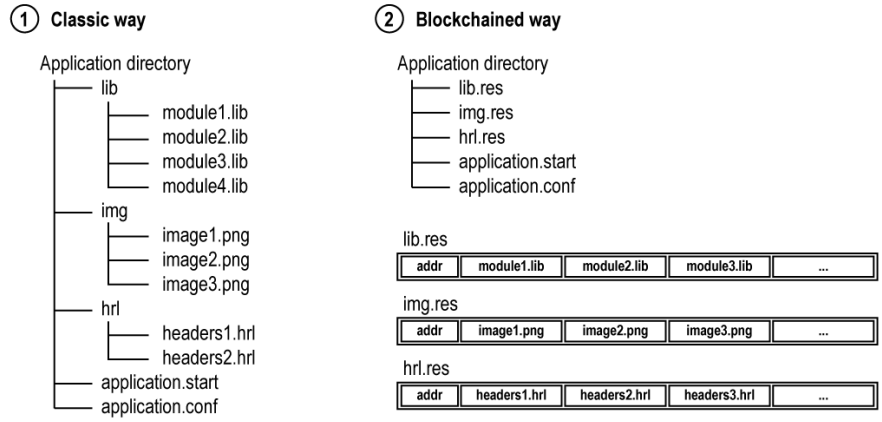
The application's global task (application idea) will always define the chain of elements that has to be blocked.

The task: develop the client application that will be strictly dependable on server and using web-technology for accessing to the data stored on the server. In other words it was a browser that is hard-coded for using only one server within improved level of security.

One of the solution for achieving of improving security was blockchain for avoiding injections. The task for blockchain - is excluding as possible the ability to inject anything in the compiled application on the user's side or at least to reduce the time for recognising compromised application.

Coming back to the blockchain it's always required to define the chain of elements that should be blocked. In this example there are files that used by application.

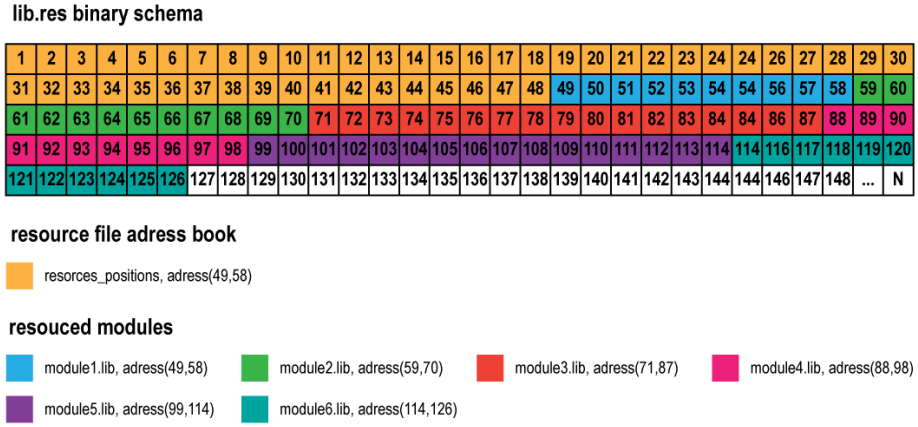
In other words the task might be rephrased: the chain of bytes that is contained in files that is contained in application should be blocked and became constant.



The classical way for application is storing in directory and if it needs within subdirectories.

Every file for application is unique and the uniqueness might be calculated by MD5 function. But! The result of MD5 calculation always constant for one constant file. This is the biggest problem in this case. Anything that is standing at one place always possible to get around sooner or later, it's matter of a time only.

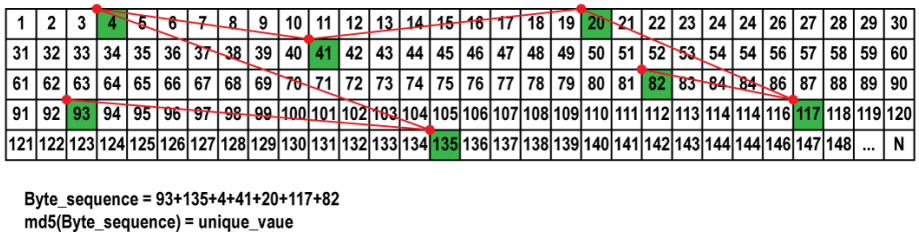
The task has been complementing by sub-task: dynamically generated value that is ensuring uniqueness of application that is going to be ran. This issue might be solved by concatenating the files that application containing into resource file by especially defined order where in begin of file or somewhere else located "address book" of concatenated resources. Might looks like own file system inside of the one file.



The order of concatenation might be different:

- one-by-one: when the files concatenated as it is one by one and position storing in "address book" (to values - first byte and last byte)
- fragmented: when the files splitting on fragments and concatenated by fragments in random order within storing fragments position in "address book"
- custom: when the developer is providing something special by his own

At time of compilation the files concatenated in resource files and this resource files stored on server like constant etalon - one side of equality.



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

At time of application start, the application loader only calling server that is sending list of bytes positions that has to be calculated for getting uniqueness. Before sending it to client server is calculating it by self and storing value and waiting the client application (client application reading bytes by position and calculating value fro return to server) value for checking equality and if all is ok sending kind of PROXY module on a binary level that is unpacking resources to

memory and making available each of application module to other by defining special namespace. Sort of connector when absence of it making impossible to use application, sort of session token but token not the value, token is the function that gotten from server and generating namespace.

The same approach might be implemented for the case of memory allocated for application checking.

This approach is rising attackers upon more difficulties or at least making time for attack recognition really less in case of implementing it into kind of heartbeat application module.

And again this is not final unique solution for any case. Every case should be reasoned and got a clear understanding what is chain of element that need to be blocked.