Research on the managing method for distribution of Node -based project by encrypted file set

Yong Dok Pyon*,Chol Ju Han, Kwang Chol Bang and Song Jin Jong
Faculty of Information Science, Kim Chaek university of Technology, Yonggwang St, Kyougu-dong No.60, Pyongyang, DPRK

Abstract

When you are going to create a Node-based project, it will be necessary to find solutions for the management of tens or hundreds of thousands of node modules and sources, and for the protection of the source code in the distribution. Using the existing versions of various products such as NW.js, you could find those solutions.

In this paper, we proposed a file request processing method using the encrypted file set on the basis of the analysis of the file request processing of the open source project Node.js, implement the managing method of the source code and resource files, verify the practicality and effectiveness of the proposed methods, and newly established the distribution and protection mode of Node-based product.

Keywords: Node.js, JavaScript, Security, Source Code

1. Challenges in Node-Based Project Distribution

In general, there are two problems in the distribution of a Node-based project.

The first problem is that it takes a lot of time to copy files because of the large number of node-module. Even though you optimize the project before distribution, the overall capacity is less than 300 ~ 500 MB, but the number of files is more than tens or hundreds of thousands. Therefore, it will take tens of minute to extract compression even on the Core i3 class computer.

And the second problem is that it is distributed without security of the source code and data.

In a Node-based project, the development language is JavaScript or TypeScript. Thus the result exists as a text file rather than binary or intermediate code, which is a fatal error when the developed product is finally introduced into another unit.

Several ways and products to solve this problem have been developed, such as Electrons and NW.js. Although the purpose of these products is to develop applications using JavaScript and HTML/CSS, both of them are based on Node.js and support the management modes for node-modules and source files, as they include Node.js as an executable environment.

The characteristics of these products are shown in the table.

Table 1. Character comparison Electron with NW.JS

Characteristics	Electron	NW.JS
File copy mode when distribution	ASAR	ZIP
Source Code Encryption	X	File unit
Run-time Code Provisioning	ASAR	Extracting
Data Set mode	Embedded in ASAR	X

As can be seen from the table, it has some limitations in source code and data distribution management.

2. Source Code and Data File Management Mode of Node

Node.js is an open source project, with a source project that can be built on several operating systems open to the Internet.

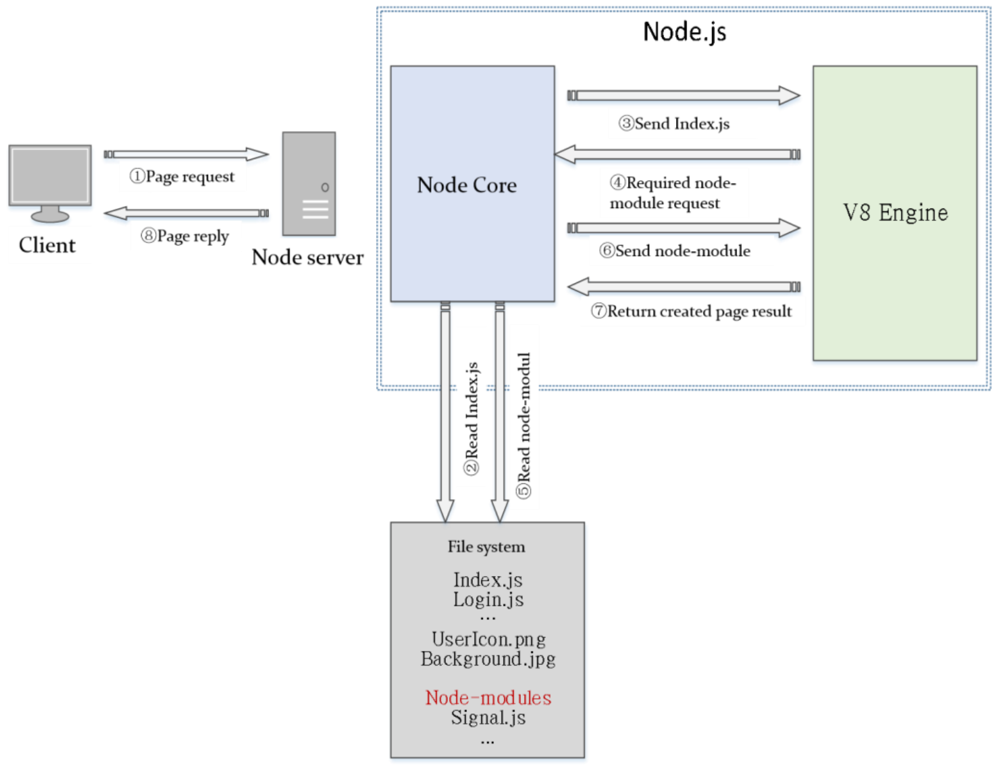

Fig.1. Operation process of existing Node.js.

If a page request comes from terminal, the Node kernel finds a JavaScript source file corresponding to the URL in the file system and reads the content and maps it to the V8 interpreter.

The V8 interpreter requests the Node core to send the associated node-module or source file, and finally the page result generated by the V8 interpreter is sent to the terminal via the Node kernel to the response.

The terminal's request for static resources (figure, text, music, etc.) is also sent to the terminal from the server's file system via Node.

3. Managing Method of Source Code and Data in Node-Based on the Encrypted File Packet

Based on the analysis of the source code of Node.js, the action of the file request handler can be modified to effectively handle source code and data management.

First, we create a fast search tree structure that makes source code and resources a set of encrypted files and can respond quickly to file requests.

Then, based on the generated search tree, it returns the result of decoding the corresponding content of the encrypted file packet to the file request that occurs in the Node kernel.

3.1. Encrypted File Packet Generating Method

Definition of Search Tree Structure

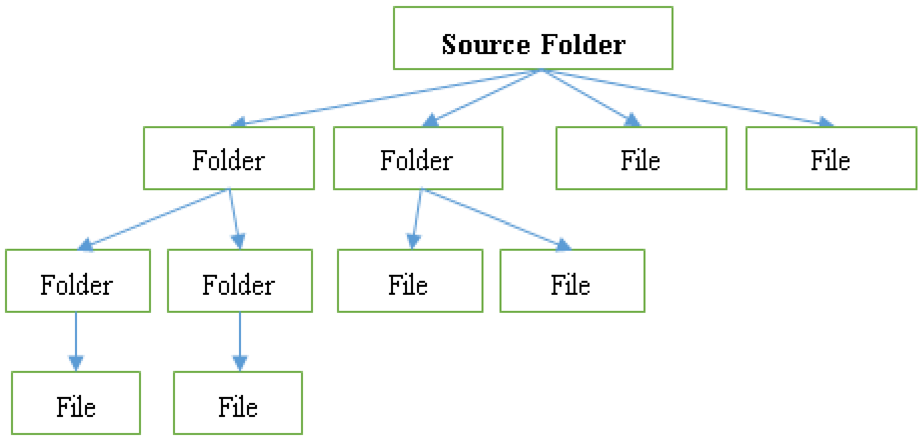

Fig.2. Structure of the Search Tree.

```
typedef struct
{
 int fileSize, offset;
 BYTE bDirectory;
 WCHAR* fileName;
 int length, szChild, numberOfChild;
 int* child;
} Node;

typedef struct
{
 int numberOfNodes, szNodes;
 Node* nodes;
 int indexOffset;
 HANDLE file;
 int decryptionFlag;
} IndexTree;
```

In the defined structure, all files are stored in the leaves of the tree, and the node means the directory. In the leaves of the tree, information is stored to read the file contents in the file stack.

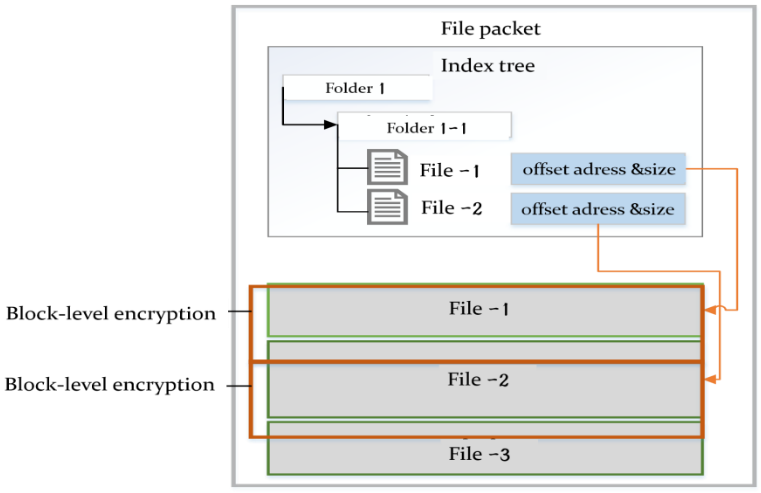

Fig.3. Structure of File Packet.

The first header of the file stack is an index tree, and then the file content partition is encrypted in a block of certain size, resulting in source code files and the required resource files being encrypted and stored in one file packet.

3.2. Node Kernel File Request Processing Method

The file request of the kernel is generated by the client's page request, the resource request, or during the execution of the V8 interpreter. The features of Nodes are Non-Block, asynchronous-event-driven mode, so that the response to the file request is not synchronized and is completed by dividing it into the file request directory and file request execution.

The file request directory checks the availability of the file request and registers in the queue. In the file request execution section, the requests registered in the queue are taken out and executed one by one. In the case of a large file, the file request is not completed in a running and is completed several times.

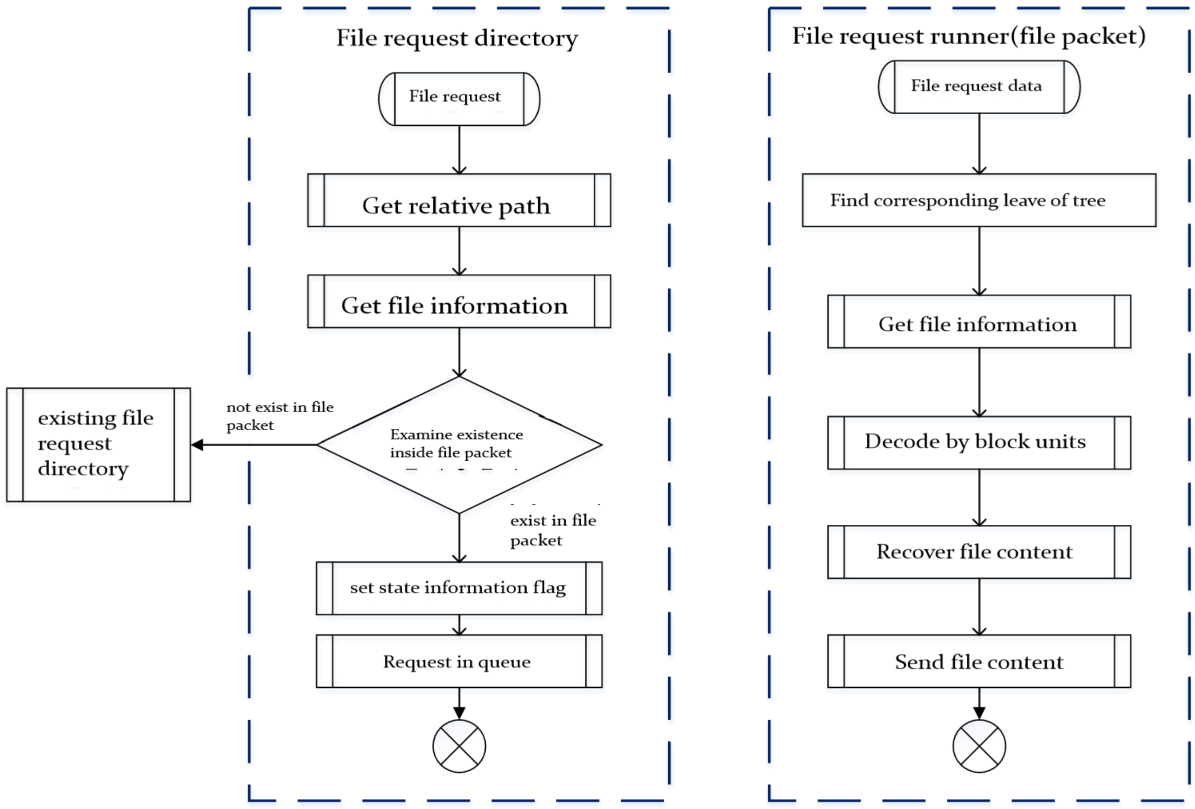

Fig.4. Updated Handing Process of File Request.

4. Experiment and Analysis of Results

We compared our approach with Electron, NW.js and Source Node running environments. The ex perimental data were used by writing seven JavaScript pages referring to external library files of 10, 2 0, 30, 40, 50, 60, and 70.

As can be seen in Fig. 5, the response to file requests was fastest in Electron, which was related to the fact that Electron was using an unencrypted file package in ASAR mode, in this experiment, file unit encryption was performed when building projects with NW.js, which resulted in the longest response time for file requests.

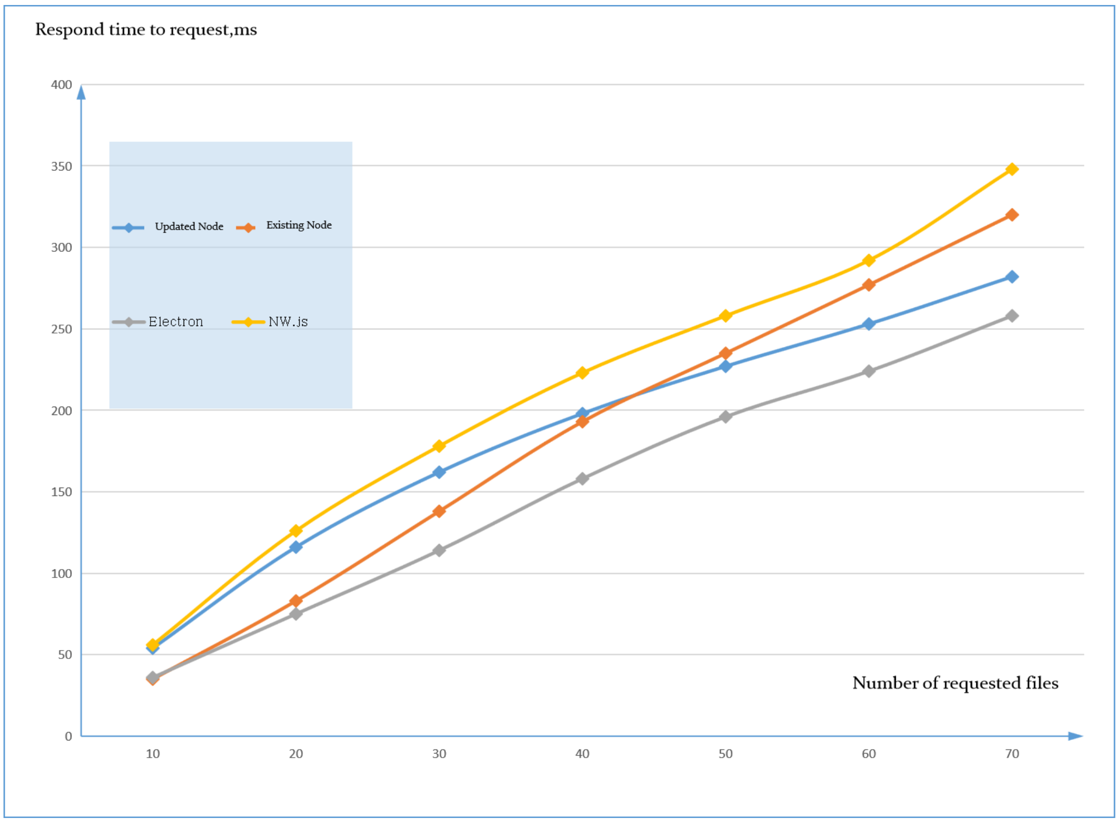

Fig.5. Response Processing Time with Increase in the Number of Files requested.

The updated Node was initially measured to be delayed by decoding time compared to the existing Node, but as the number of request files increased, the response time was faster than the existing Node, because the frequent file input power in the existing Node would rather delay the response time, whereas the updated Node would only maintain an I/O of file and encrypt/decode in the block unit, thereby increasing the decoding efficiency as the number of files was constant, resulting in a faster response time than the existing Node.

Table 2. Additional Memory Consumption with File Number of File Packets

Number of files in the file packet	Memory(RAM) consuming amount
1000	300KB
10000	3MB
50000	15MB
100000	30MB

As the Table 2 shows, even if we get the number of files in the stack is large enough, the memory consumption can be significantly reduced as a feature of the Dual Core class computer, resulting in a fully improved Node performances as a web server, and efficient management of source code and resources can be achieved.

5. Conclusion

In this paper, we have modified the file request handler of the Node.js kernel to implement source code and resource management methods based on one encrypted file set, verify the practicality and effectiveness of the proposed method, and newly established the distribution and protection mechanisms of Node-based products.

Acknowledgment

The authors would like to thank Prof. Dr. Chol Man Kim who is a boss in the development of NSDL, Hung Dok Cha, Chol Hyok Jo, Jong Chol Ri, Wol Il Ri, Dong Kuk Kim, and Yong Gil Han who are developers, and Dr. Yun Hak Choe and Gwang Jo Kim for his valuable data. Also, we would like to thank all developers of the convenient and powerful Node.js.

References

[1] Adam Rapley, Xavier Bellekens, Mayall: A Framework for Desktop Javascript Auditing and Post-Exploitation Analysis, arXiv, 2018, 3-18
[2] Brad Dayley, Brendan Deayley, Node.js, MongoDB and Angular Web Development, 2010, O'Addison-Wesley, 39~190
[3] Tom Hugbes-Croucber, Mike Wilson, Node Up and Running, O'REILLY, 2014, 25~67
[4] David Mark Clements, Node Cookbook, Packt Publishing, 2013, 47~48
[5] Alessandro Benoit, NW.js Essentials, Packt Publishing, 2015, 143-160
[6] Tilkov, S.; Vinoski, S. Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Computing 2010, 14, 80–83
[7] Meyerovich, L.A.; Zhu, D.; Livshits, B. Secure cooperative sharing of JavaScript, browser, and physical resources. Proceedings of the Workshop on Web 2.0 Security and Privacy, 2010
[8] Carter, B. HTML Educational Node. js System (HENS): An Applied System for Web Development. Information and Computer Technology (GOCICT), 2014 Annual Global Online Conference on. IEEE,2014, 27–31
[9] GitHub. Apps Built on Electron, 2017. URL: https://electron.atom.io/apps/ [Accessed 6 April 2017]
[10] Node.js;. Available from: https://nodejs.org/en/
[11] Williams, C. How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript, 2015. URL: https://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/ [Accessed 12 April 2017]
[12]GitHub.Electron Documentation — Electron Versioning, 2017. URL:https://electron.atom.io/docs/tutorial/electron-versioning/ [Accessed 12 April 2017].
[13] Node Security. Advisories, 2017. URL: https://nodesecurity.io/advisories [Accessed 24th April 2017].