"ORB"
DECENTRALIZED FUNCTION AS A SERVICE (DFAAS)

Project ID – 18-070

Preliminary Progress Review Report

S.K.N.U Tissera

IT15120212

Bachelor of Science Special (Honors) Degree in Information Technology

Department of Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

March 2018

Table of Contents

Table of Figures

1.0 Introduction

1.1 Purpose

The purpose of this document is to describe how user interacts with the system via client software which makes the system more transparent to the user and easy to use. Furthermore, it explains the total functionality of the client software to sync with deployment framework and Internal Infrastructure to bring down the complexity of the system to the end user.

When deploying a function in Orb there should be a user-friendly environment for the users to work with. Deploying a function via a node should be able to achieve with few easy steps to provide a better user experience. In the client software, from user registration up to function hosting, seeding and requesting state providing a user interface hiding the complexities of the system is the main purpose of this component.

1.2 Scope

I. **Creating a deployment framework to deploy functions and their configuration in a way to suite the decentralized network.**

   Deployment framework is built on top of node infrastructure at client end to facilitate hosting and seeding functions. This resides behind the user and act as a wrapper for the node. Nodes ultimately communicate via sockets with the Supernodes.

II. **Client software**

   This is the Graphical user interface (GUI) for the client. This facilitates from Registration of the user to Deploying a function. Steps include; authenticating user in the decentralized network, logging to the system, writing a function, obfuscate the function, packaging and deploying a function, providing access tokens for the specified function. Main purpose of this is to bridge the gap between client and network infrastructure.

III. **Security**

   Security is maintained throughout the system from registration of user to function deploying and calling. Although this is a decentralized system, security is good than a centralized network. Decentralized network is proof of DDos attacks since all requests are routed in the system via supernodes. Socket.io plays a role is securing network proofing external attacks.

IV. Obfuscating functions

Obfuscation is the process of making something difficult to understand. JavaScript function is obfuscated because its an intellectual property and it prevent an attacker from reverse engineering function in the network. It normally renames useful class and variable names to meaningless labels or adding unused or meaningless code to an application binary. The output after obfuscating is harder to read and understand. So, it has following benefits

- Hides JavaScript function logic
- Prevent from reverse engineering
- Can also be used for minifying (*.min.js)
- Code size is reduced

For this purpose, we are using npm package 'Javascript-obfuscator'. JavaScript obfuscator is a powerful free obfuscator for JavaScript and Node.js with a wide number of features which provides protection for your source code.

V. Configuration

Configuration is the last step before deploying a function. Each function has a function descriptor. It a JSON object (Java Script Object Notation). This contains Orb id, function id, description about the function (versioning, outputs, parameters need to be passed, when a consumer can use it etc.).

Obfuscated Function is embedded inside this as a String Object. This needs to be extracted from JSON object whenever function needs to be executed.

1.3 Overview

Main goal of this is to create a deployment framework which facilitates the user to interact with the decentralized network. User can register, write a function and deploy it in the system via a client application with ease. User can use functions deployed in the system. Users can subscribe for functions also. Users can use an inline code editor to write their functions. These functions are then obfuscated and distributed in the network with a unique key for the function.

When a client writes a function and moved for deploying, Client software will embed a JavaScript code into that function which enables to create a connection and deploy in Orb network. Client software uses a JS interpreter to execute obfuscated JavaScript function. Client software also provide an access link for a function after deploying. Finally, this Obfuscated function is embedded inside a JSON (Java Script Object Notation) for deployment.

1.3.1 Main Goals

The main goal of this component in research is to create a deployment framework and client software which makes the user (developer/non-developer) easy to deal with Orb. Therefore, a user can keep focus on the functionality of the application via the client software rather than focusing on internal infrastructure of Orb. Orb will automatically take care of the deployments. The secondary goal is to utilize idle consumer computing power and to Reward the contributors in terms of seeding, consumption and deployment of functions through a decentralized Rewarding System. At startup we should mainly focus on user onboarding to the system. Due to several reasons some users may be precautious to use this. One thing is thing is new way of deployment and it has no central authority to control. So, to eliminate these extreme phobias and get the user onboard we need to be smart with way we interact with the user. So fully featured client software and a good, transparent rewarding system can make the user trustworthy.

1.3.2 Audience

This will mainly focus on the user. User may be a developer or a non-developer. By the way client software should be good enough to give a favorable working environment for the user.

1.3.3 Major Functionalities and Tasks

There are 3 Major Tasks or Functionalities where a particular user could interact with the Orb. Deployment of Functions to the Orb network being the major functionality, Seeding of a function and Consumption of an already deployed function could also be achieved.

1.4 Definitions, Acronyms, and Abbreviations

JSON	Java Script Object Notation
DoS	Denial of Service
GUI	Graphical User Interface
DHT	Distributed hash table
DDOS	Distributed Denial of Service
NAT	Network Address Translators
FaaS	Functions as a Service
IPFS	Interplanetary File System
DFaaS	Decentralized Functions as a Service
API	Application Programming Interface
AWS	Amazon Web Services
IoT	Internet of Things
Orb	Name of the research prototype
Seed	A node which hosts a function
Function	A executable code which performs a well-defined specific function
pkg	A JavaScript interpreter which executes code in a sandbox
sandbox	A safe to use testing environment
BLOB	Binary large object
NAT	Network Address Translation

2.0 Statement of Work

2.1. Background information and overview of previous work based on literature survey

Many systems build for variety of purposes on top of decentralized networks are becoming popular recently. Protocols like IPFS (interplanetary file system), p2p browsers like beaker browser, decentralized data storing platforms like storj, social media platforms like Steemit are trending.

IPFS is a distributed file system that can expand widely by growing with no of nodes and facilitate the user to keep files by sharding. If we use it right we can make a huge decentralized system for content sharing. An IPFS object is a data structure with 2 fields as data and links. Links has a name, hash, size. IPFS allows to share pictures, articles, videos.

Beaker Browser is decentralized network to create and host websites. It's just few steps; download, create a page, deploy it. Using beaker users can share personal files and make webpages. Every app is hostless and forkable source code.

Storj is decentralized cloud storage. Simply you can rent your hard drive and earn money. Its end to end encrypted and payments are powered by blockchains.

Steemit is a social media platform build on top of decentralized network. Users write posts and there is an attractive rewarding system; Where both reader and contributor get rewarded.

2.2. Identification and significance of the problem

This idea of decentralization is not new to the community. But hosting functions in a decentralized system is fairly new to community, since no one have implemented it earlier. The above-mentioned platforms store static content (e.g.: storj, IPFS) while our system allows the user to store a dynamic content. That's our main uniqueness.

Protocols like BitTorrent (file sharing protocol) authorizes and points to remote nodes that contain the part of the file to be shared or downloaded. We keep JSON objects in the system. Functions are embedded into JSON object. These functions execute at node end and produce outputs dynamically. These functions (JavaScript language) are executable on the node end using an Interpreter and perform Realtime function outputs.

Technical challenge is there since it's new. We have found challenges in public-to-private communication in networking and NAT (Network Address Translation). These problems come when nodes are distributed via several networks.

2.3. Technical objectives

User needs to download client software and install it. Then client software will generate a key pair and a unique id for each client. User needs to do this only one time. Generated id is unique for a client and it's a must to keep it safe. If the user is installing the client software for the second time, no need of creating an id again. User can upload the key pair and get sign in. So key pair act as an entry key for the whole system.

Client software generates the key using several parameters (e.g.: timestamps, email). One single node (consumer device) is recognized as user at a time. Consumer device (node) is simply a Desktop PC, laptop. Later we can extend this to IoT and mobile devices, since infrastructure is common for all. For an example if we consider a PC it need to have normal specifications like i3 or i5 processor, 4GB ram etc.

System will create a VM (Virtual Machine) inside a consumer device and specify that space for the software to work. It will consume a specified space from a hard disk drive temporary.

When we discuss about the hardware requirements needed for a supernode, it's also a device with high end processing power compared to nodes. Supernodes needs to keep following data on it.

I. DHT (Distributed hash table) to keep track of known (live) nodes in the system.
II. DNS to resolve other supernodes in the system.
III. Essential information of the user.

We need to have more supernodes around the world for a better scaled Orb. Then Supernodes can perform a health check and route in the user request in the system with lowest latency.

2.4. Detail design

2.4.1 Design Architecture

Total system architecture is based on decentralized network. Proposed method of implementation is using Socket.io. Socket.io is JS library for Realtime web applications (RTA – Real Time Application). It can be used to enable bi-directional communication between node and supernodes. Socket.io basically for faster and reliable web socket communication in our decentralized system. This facilitates the system creating an Event-driven architecture when calling supernodes and calling functions.

2.4.2 Design Concept

For the implementation of the client software we use Electron (https://electronjs.org/). It’s a framework for building native applications using technologies like JavaScript, HTML, CSS. It provides native menus, UI elements and notifications for Windows MacOS and Linux. It also handles crash reporting, debugging and profiling. Electron has windows installers also.

Our electron app basically contains a dashboard, inline code editor, Orb wallet, function deploying interface. Inline code editor enables the user to write code in the application itself. Then these functions are saved as Javascript files. Later these are obfuscated for security purposes.

When user wants to use a function, application inputs function to a Javascript interpreter and execute it real-time.

Socket.io has implemented Socket.io P2P that can be used to create a bidirectional event channel between peers. It’s easy and reliable way to setup a WebRTC (Web Remote Web Client) using that protocol.

2.6. Anticipated benefits

2.6.1 No need to pay attention to internal infrastructure

Proposed decentralized system helps developers to code the functions without paying attention to internal infrastructure. Prevailing serverless architectures have known drawbacks. This can be used to eliminate them fully or partially.

2.6.2 Eliminate SPOF

Single point of failure (SPOF) is eliminated in this network, since traffic is routed between the supernodes. DDoS attacks are also prevented because of this reason. Function calls are mediated by super nodes depending on the latency and number of requests per second automatically balancing the traffic.

2.6.3 Easy way of Rewarding and Paying via Client software

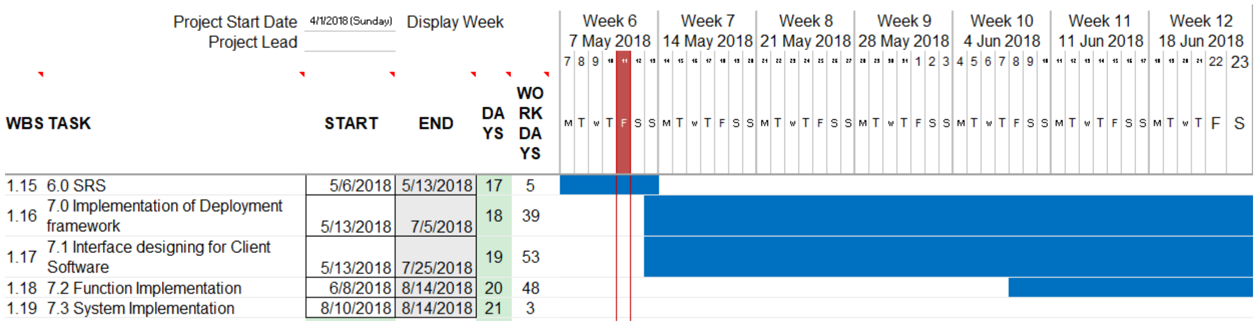
Orb brings an unprecedented level of benefits for the user by rewarding for functions deployed. When someone is using a function, he/she should pay for the function seeder and owner. User can pay per request and it’s an easy payment model. Once user get registered for a function user receives an id of that function.

2.6.4 Transparency to the user

Client software is the platform for user to get connect with the system. It is capable of registering a new user, logging, show available functions, seed available functions, request for an available function, write a new function and deploy. All these processes are very transparent in the system.

3.0 Project plan and schedule

Following is a Gantt chart showing work distribution for my component within the coming weeks.

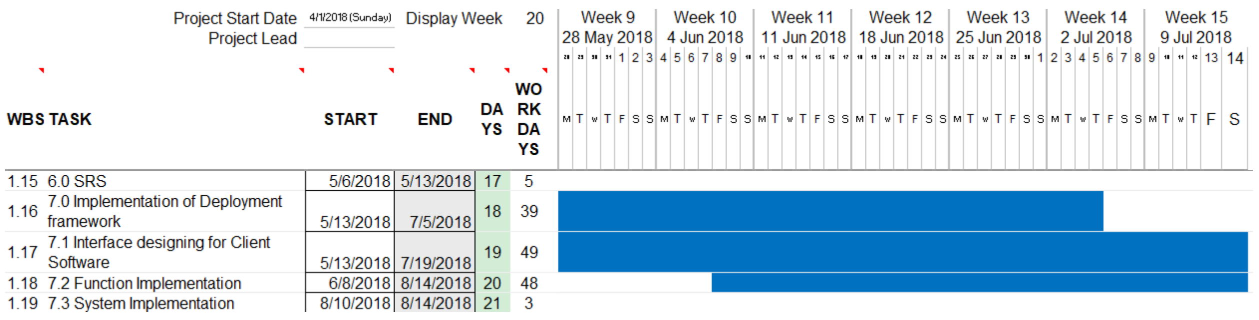


Project Start Date 4/1/2018 (Sunday) Display Week

Project Lead

WBS	TASK	START	END	DAYS	WORK DAYS
1.15	6.0 SRS	5/6/2018	5/13/2018	17	5
1.16	7.0 Implementation of Deployment framework	5/13/2018	7/5/2018	18	39
1.17	7.1 Interface designing for Client Software	5/13/2018	7/25/2018	19	53
1.18	7.2 Function Implementation	6/8/2018	8/14/2018	20	48
1.19	7.3 System Implementation	8/10/2018	8/14/2018	21	3

Figure 1: Gantt Chart showing work distribution (part 1)

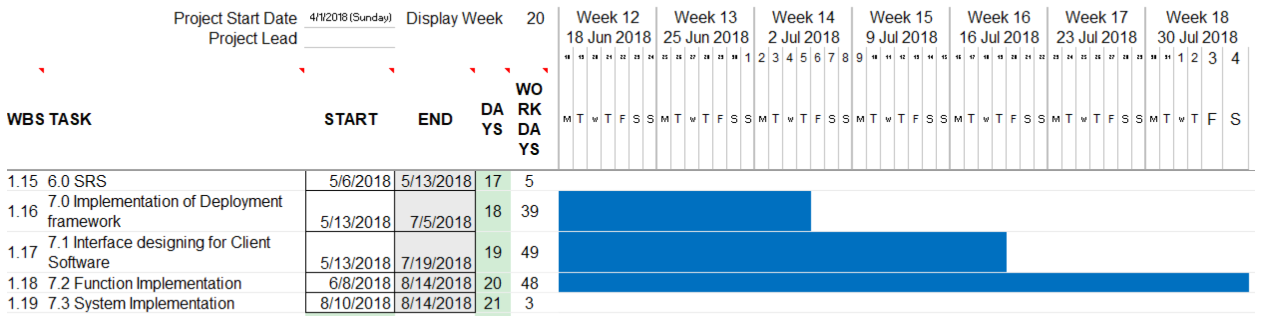


Project Start Date 4/1/2018 (Sunday) Display Week 20

Project Lead

WBS	TASK	START	END	DAYS	WORK DAYS
1.15	6.0 SRS	5/6/2018	5/13/2018	17	5
1.16	7.0 Implementation of Deployment framework	5/13/2018	7/5/2018	18	39
1.17	7.1 Interface designing for Client Software	5/13/2018	7/19/2018	19	49
1.18	7.2 Function Implementation	6/8/2018	8/14/2018	20	48
1.19	7.3 System Implementation	8/10/2018	8/14/2018	21	3

Figure 2: Gantt Chart showing work distribution (part 2)



Project Start Date 4/1/2018 (Sunday) Display Week 20

Project Lead

WBS	TASK	START	END	DAYS	WORK DAYS
1.15	6.0 SRS	5/6/2018	5/13/2018	17	5
1.16	7.0 Implementation of Deployment framework	5/13/2018	7/5/2018	18	39
1.17	7.1 Interface designing for Client Software	5/13/2018	7/19/2018	19	49
1.18	7.2 Function Implementation	6/8/2018	8/14/2018	20	48
1.19	7.3 System Implementation	8/10/2018	8/14/2018	21	3

Figure 3 : Gantt Chart showing work distribution (part 3)

First, it’s a must to create the framework for the client software to work with. It should consume functions in the network and perform actions. Completing deployment framework, implementing client software (UI and logic) can be started.

4.0 Research constraints

There are limitations in function deployment for the user. Proposed system can only upload functions written in Javascript only. So, it’s a main constraint we found in our system and we are going to implement in future stages. There is a reason behind implementing like that. That because, Socket.io is a Javascript library. So, it’s more convenient to deal with Javascript when we are using it.

This decentralized system can connect any kind of consumer device to the network as a peer. For an example mobile, laptop, desktop or an IoT devices. But as a start we only support desktop applications. So, secondly this is a constraint in the system.

Another constraint is user cannot log from several computers at one time using the same password. We implemented that for security purposes. But think of an instance if the working device go down in an instance, So that’s also a constraint in the system.

5.0 Specified deliverables

The specified deliverables contain a fully functional client software which enables the user for the following

I. Create an Orb id

This is generated on timestamp and some unique fields of user. This Orb id is to be used a unique id in the system for both wallet and decentralized network. One can generate that using the client software.

II. Generate a Key-pair as encrypted password

For logging purposes and authentication, we use a key pair (as .pem in aws ec2 consoles). It's also generated randomly and unique for that Orb id. User can upload the keypair when logging.

III. Interface for logging to system using Orb id and Key-pair

It an interface that comes when logging to the system. User must have those 2 for the logging purposes. That means user should keep key-pair and Orb id safe. This interface only comes when we are using it for first time. But at critical occasions like delete a function or deploy, it may appear again.

IV. Code editor for writing functions

This is an Inline Code Editor. We can write, edit functions on the application without using another IDE (integrated development environment) for it. (like aws cloud formation). This is a very easy for the user to interact with the system.

V. Inbuilt Obfuscator

This runs in background as a feature after code deployment for security. Obfuscator makes the code unreadable by changing variable names, whitespaces, word order etc. When a user writes a function, it's obfuscated. This is for extra security purpose.

VI. Inbuilt Javascript Interpreter

Interpreter can execute a Javascript function and produce outputs. Initially our Javascript functions are obfuscated. But It's not a problem for the interpreter to execute.

VII. Interface for display available functions

There is a List UI to show available functions. User can consume available functions (under payment scheme). This window will give a detailed view about the function usage, outputs etc. User can deploy his own function also using that window.

VIII. Subscribe for a function

User can subscribe for any function in the system. Initially user will receive some number of free requests for available functions (as defined in rewarding system). When user start calling functions that free amount will go down. It will be shown in the dashboard. So, user can track it easily. Afterwards user should pay for requests.

6.0 Reference

[1]"What is obfuscation (obfu)? - Definition from WhatIs.com", SearchSoftwareQuality, 2018. [Online]. Available: https://searchsoftwarequality.techtarget.com/definition/obfuscation. [Accessed: 07- May- 2018]

[2]"An Introduction to IPFS – ConsenSys – Medium", Medium, 2018. [Online]. Available: https://medium.com/@ConsenSys/an-introduction-to-ipfs-9bba4860abd0. [Accessed: 07- May- 2018]

[3]"What is a Torrent? - Definition from Techopedia", Techopedia.com, 2018. [Online]. Available: https://www.techopedia.com/definition/5263/torrent. [Accessed: 07- May- 2018] [4]"How Do Ethereum Smart Contracts Work? - CoinDesk", *CoinDesk*, 2018. [Online]. Available: https://www.coindesk.com/information/ethereum-smart-contracts-work/. [Accessed: 14- Jan- 2018].

[5]"ConsenSys – Medium", *Medium.com*, 2018. [Online]. Available: https://medium.com/@ConsenSys. [Accessed: 21- Jan- 2018].

[6]"What Is AWS Lambda? - AWS Lambda", *Docs.aws.amazon.com*, 2018. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. [Accessed: 22- Jan- 2018].

[7] “Steemit,” *Steemit*. [Online]. Available: https://steemit.com/. [Accessed: 24-Mar-2018].

[9]"AWS Rates Highest on Cloud Reliability", *EnterpriseTech*, 2018. [Online]. Available: https://www.enterprisetech.com/2015/01/06/aws-rates-highest-cloud-reliability/. [Accessed: 14- Mar- 2018].

[10]"The AWS Outage: The Problem with Internet Centralization | Mondo", *Mondo*, 2018. [Online]. Available: https://www.mondo.com/aws-outage-internet-centralization-problem/. [Accessed: 14- Jan- 2018].

[11]"Single Point of Failure – What is it? Why it Matters… :", *Renovodata.com*, 2018. [Online]. Available: http://www.renovodata.com/blog/2015/06/10/single-point-of-failure. [Accessed: 14- Jan- 2018].

[12]"www.ey.com", *Ey.com*, 2018. [Online]. Available: http://www.ey.com/Publication/vwLUAssets/EY-implementing-blockchains-and-distributed-infrastructure/$FILE/EY-implementing-blockchains-and-distributed-infrastructure.pdf. [Accessed: 14- Jan- 2018].

[13]P. Labs, "IPFS is the Distributed Web", *IPFS*, 2018. [Online]. Available: https://ipfs.io/. [Accessed: 14- Jan- 2018].

[14]"Storj - Decentralized Cloud Storage", *Storj - Decentralized Cloud Storage*, 2018. [Online]. Available: https://storj.io. [Accessed: 14- Jan- 2018].

[15]*Google Cloud Platform Documentation | Documentation | Google Cloud*. [Online]. Available: https://cloud.google.com/docs/. [Accessed: 02-Feb-2018].

[16]“Kubernetes Documentation,” *Kubernetes*. [Online]. Available: https://kubernetes.io/docs/home/?path=users&persona=app-developer&level=foundational. [Accessed: 02-Feb-2018].

[17]“Decentralized Internet on Blockchain – Hacker Noon,” *Hacker Noon*, 03-Oct-2017. [Online]. Available: https://hackernoon.com/decentralized-internet-on-blockchain-6b78684358a. [Accessed: 02-Feb-2018].

[18]*Blockchain Based Distributed Control System for Edge Computing - IEEE Conference Publication*. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7968630. [Accessed: 02-Apr-2018].

[19]“Decentralizing Your Microservices Organization,” *The New Stack*, 31-Oct-2017. [Online]. Available: https://thenewstack.io/decentralizing-microservices-organization/. [Accessed: 02-Apr-2018].

[20]“Building Your First Network,” *Building Your First Network - hyperledger-fabricdocs master documentation*. [Online]. Available: http://hyperledger-fabric.readthedocs.io/en/release-1.0/build_network.html. [Accessed: 02-Apr-2018].

[21]M. Meister, “leveraging Kubernetes to run a private production ready Ethereum network,” *Medium*, 26-Nov-2017. [Online]. Available: https://medium.com/@cryptoctl/leveraging-kubernetes-to-run-a-private-production-ready-ethereum-network-b6f9b49098df. [Accessed: 02-Apr-2018].

[22]*A Decentralized Service Discovery Approach on Peer-to-Peer Networks - IEEE Journals & Magazine*. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/5928313/. [Accessed: 02-Apr-2018].

[23]“Taming WebRTC with PeerJS: Making a Simple P2P Web Game,” *Toptal Engineering Blog*. [Online]. Available: https://www.toptal.com/webrtc/taming-webrtc-with-peerjs. [Accessed: 02-Apr-2018].

[24] “Decentralized payments for environmental services: The cases of Pimampiro and PROFAFOR in Ecuador,” Ecological Economics, 31-Dec-2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921800907005320. [Accessed: 05-Jan-2018].

[25] A. Crespo and H. Garcia-Molina, “Semantic Overlay Networks for P2P Systems,” *SpringerLink*, 19-Jul-2004. [Online]. Available: https://link.springer.com/chapter/10.1007/11574781_1. [Accessed: 02-Mar-2018].

[26] “Rethink the Web browser,” Beaker | Peer-to-peer Web browser. No blockchain required. [Online]. Available: https://beakerbrowser.com/. [Accessed: 05-Feb-2018].

[27] “pkg,” npm. [Online]. Available: https://www.npmjs.com/package/pkg. [Accessed: 10-May-2018].

[28] “qtimeit,” npm. [Online]. Available: https://www.npmjs.com/package/qtimeit. [Accessed: 10-May-2018].

[29] “javascript-obfuscator,” npm. Available: https://www.npmjs.com/package/javascript-obfuscator. [Accessed: 10-May-2018].

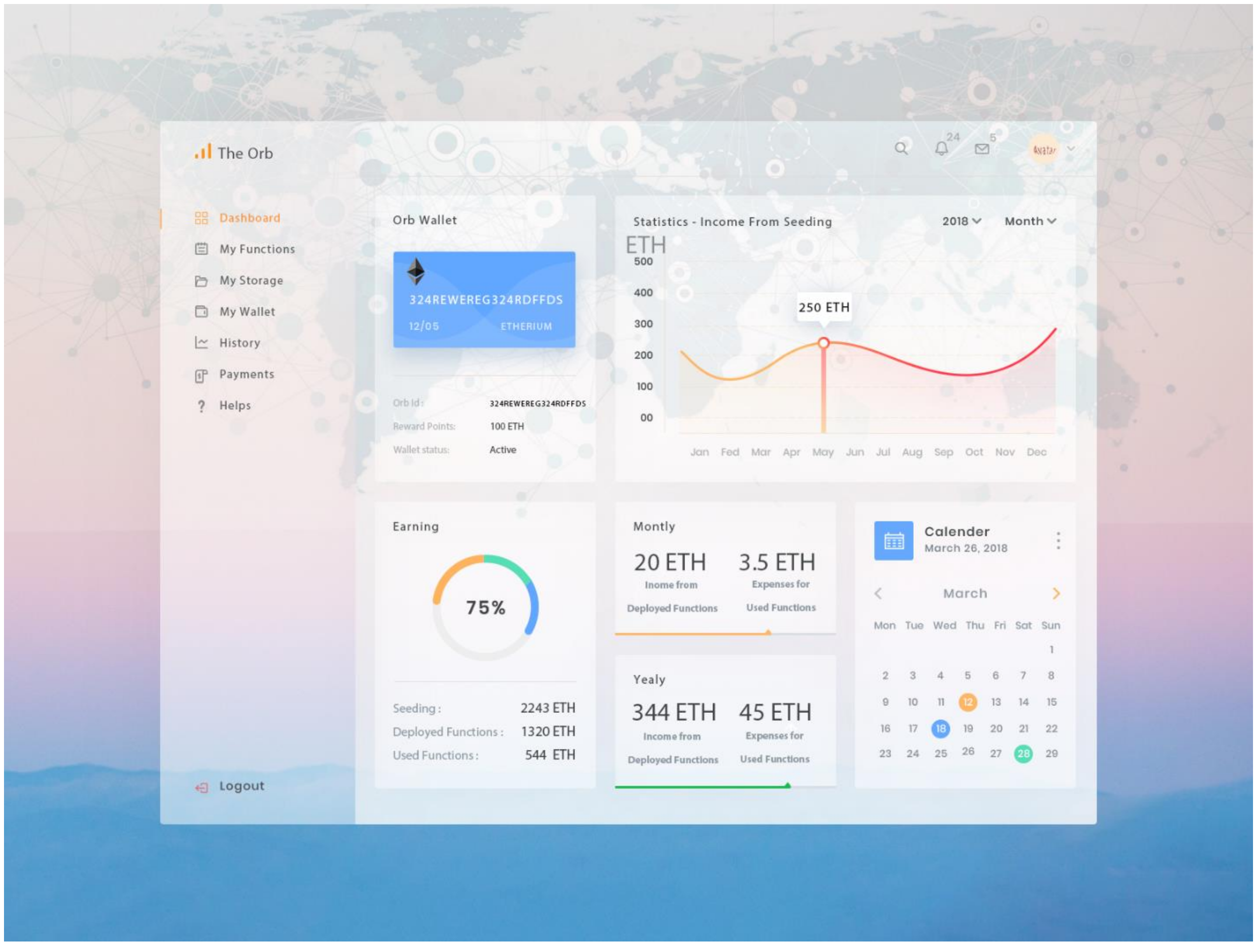
[30] "Stunnix JavaScript Obfuscator sample output", Stunnix.com, 2018. [Online]. Available: http://stunnix.com/prod/jo/sample.shtml. [Accessed: 13- May- 2018].

7.0 Appendices

Appendices A

7.1 Client application User Interfaces - Dashboard

This is the proposed dashboard for the client application.



Orb Id: 324REWEREG324RDFFDS
Reward Points: 100 ETH
Wallet status: Active

Seeding : 2243 ETH
Deployed Functions : 1320 ETH
Used Functions : 544 ETH

Montly
20 ETH Inome from Deployed Functions
3.5 ETH Expenses for Used Functions

Yealy
344 ETH Income from Deployed Functions
45 ETH Expenses for Used Functions

Figure 4 : Client Software Dashboard

7.2 Client application User Interfaces – All Functions

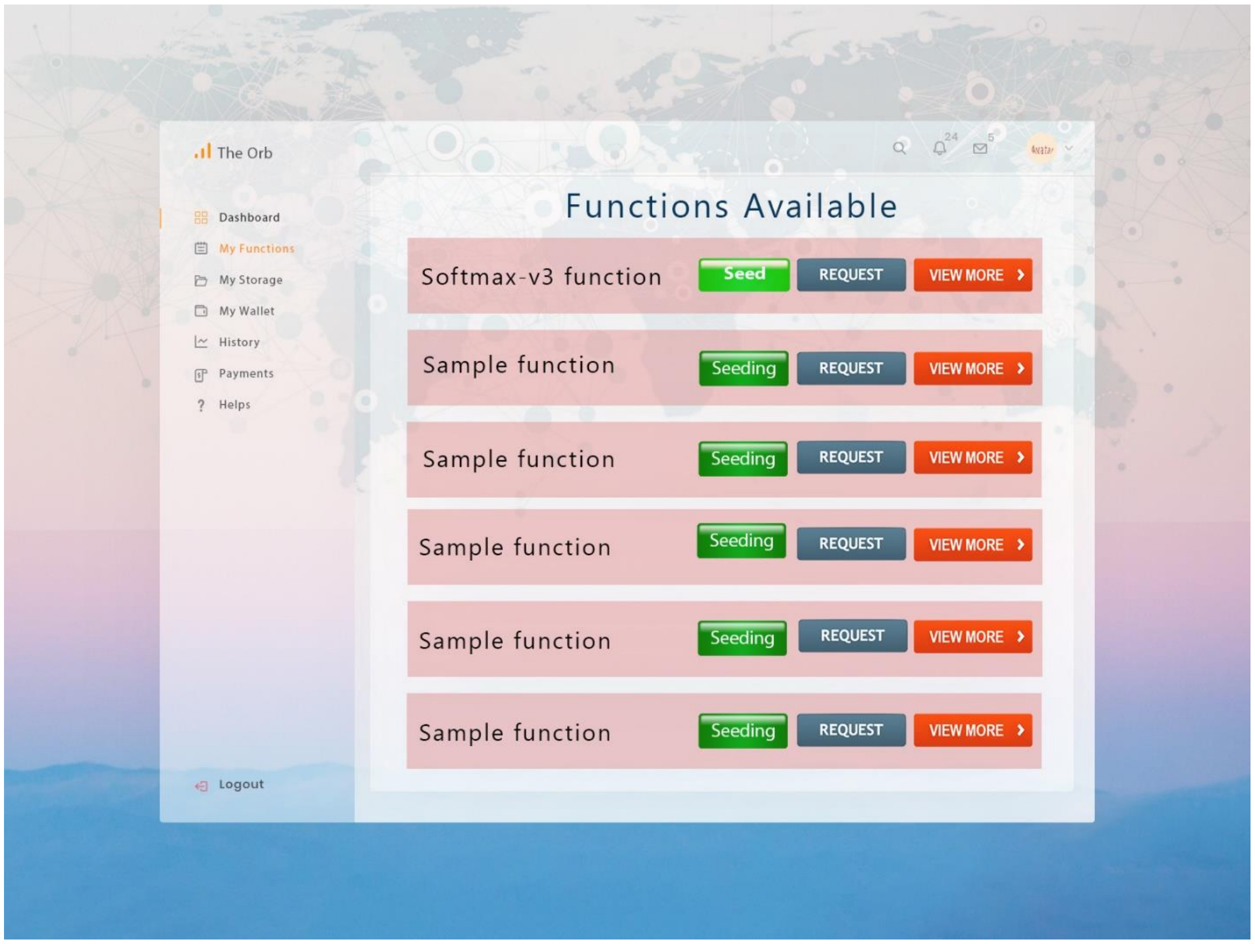
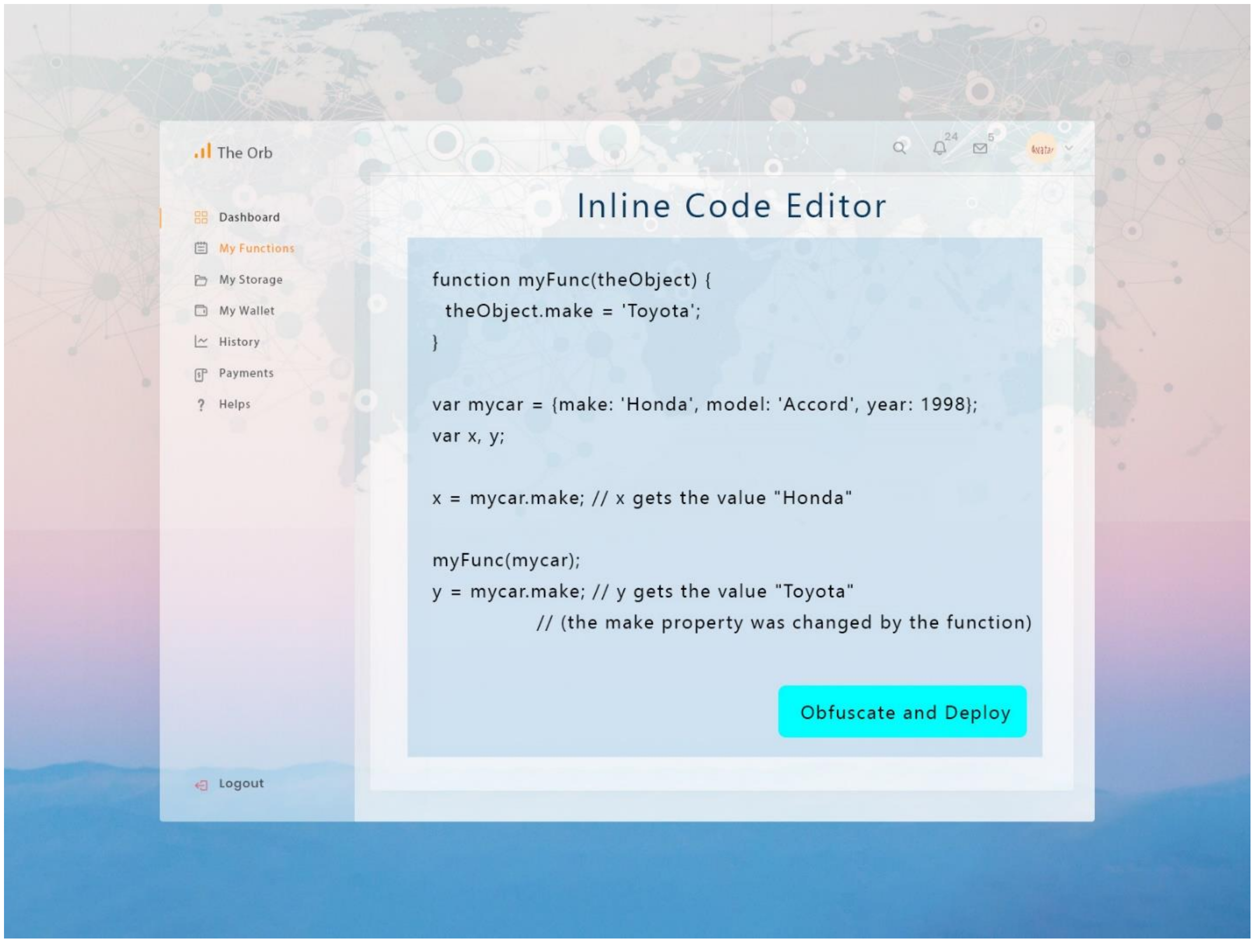


Figure 5 : Client Software Functions UI

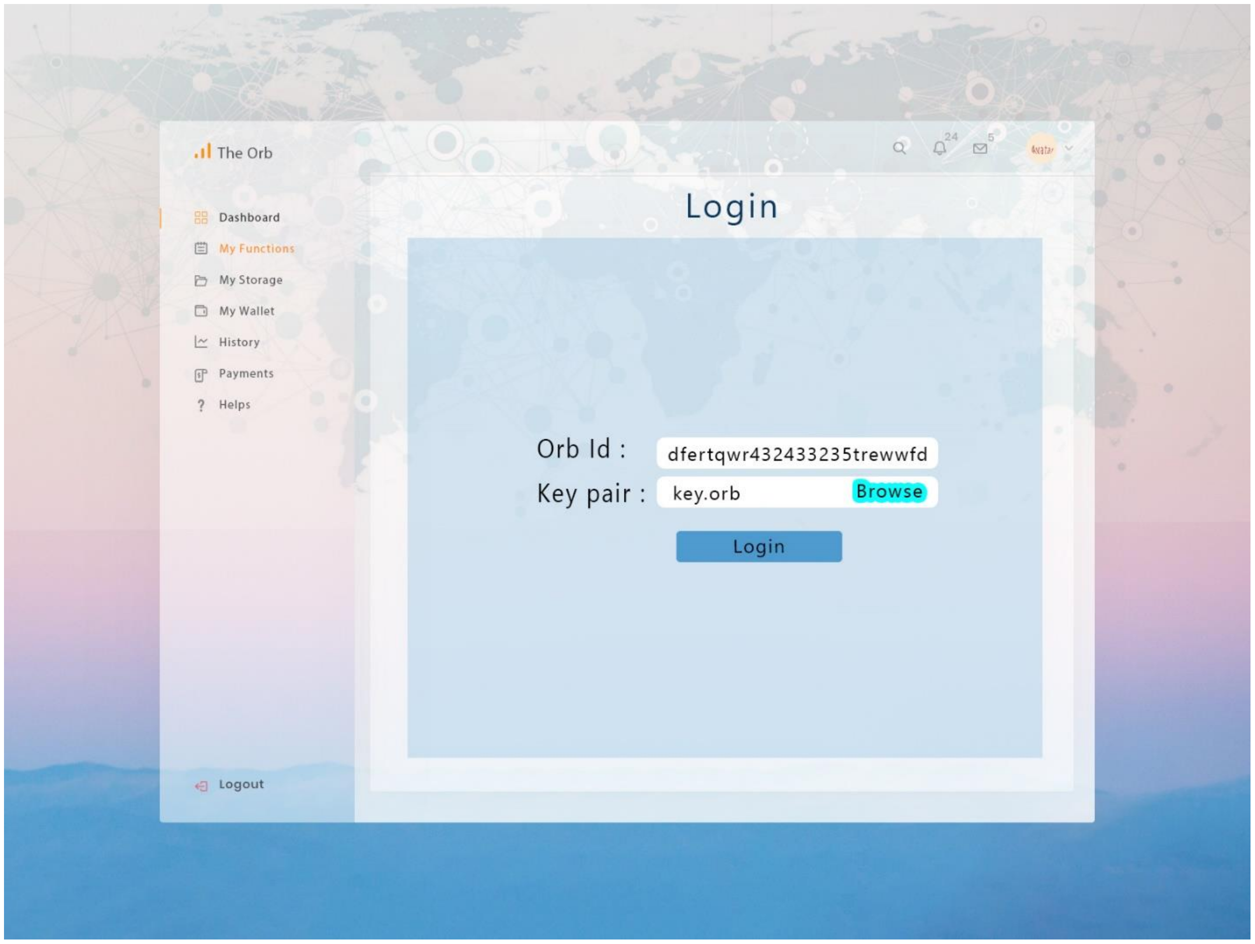
7.3 Client application User Interfaces – Inline Code Editor



```
function myFunc(theObject) {
  theObject.make = 'Toyota';
}

var mycar = {make: 'Honda', model: 'Accord', year: 1998};
var x, y;

x = mycar.make; // x gets the value "Honda"

myFunc(mycar);
y = mycar.make; // y gets the value "Toyota"
              // (the make property was changed by the function)
```

Figure 6 : Client Software Inline Code Editor

7.4 Client application User Interfaces – Login



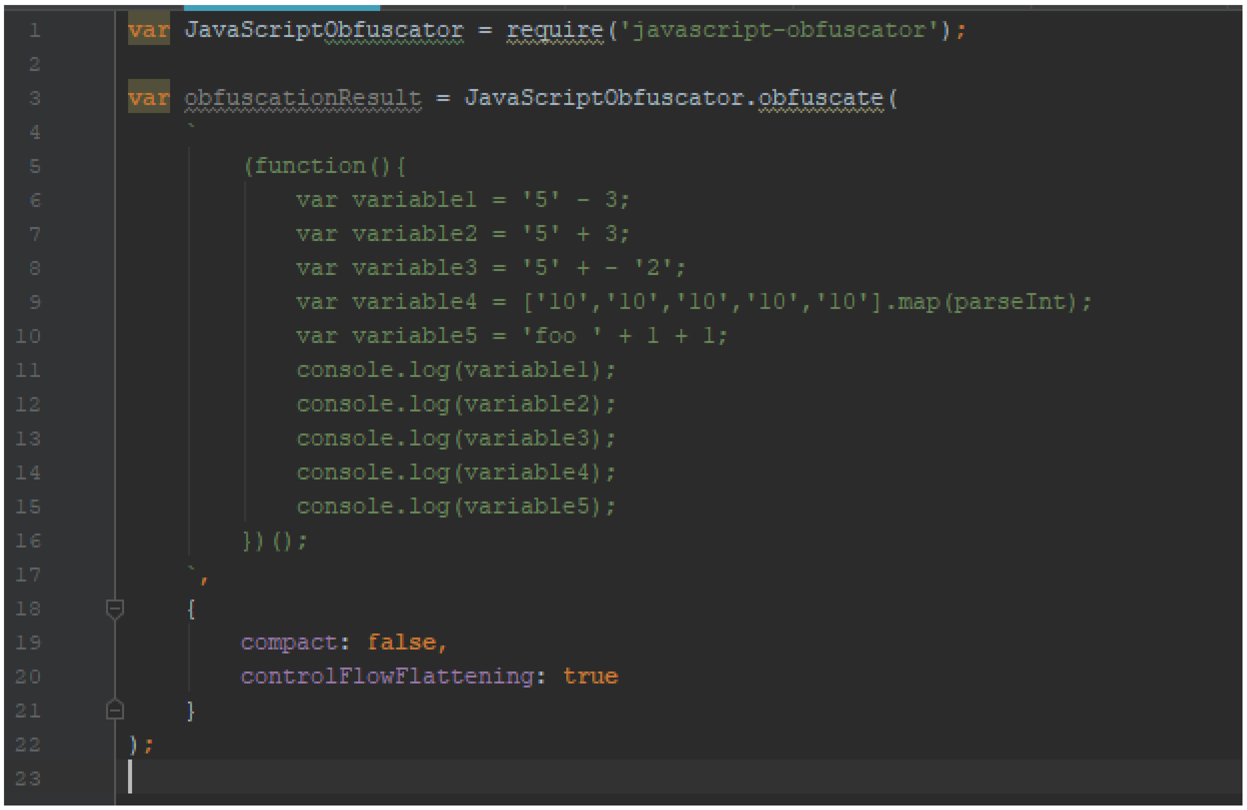
Orb Id : dfertqwr432433235trewwfd

Key pair : key.orb

Figure 7 : Client Software Login

Appendices B

7.5 Sample usage of an Obfuscator



```
var JavaScriptObfuscator = require('javascript-obfuscator');

var obfuscationResult = JavaScriptObfuscator.obfuscate(
    `
        (function(){
            var variable1 = '5' - 3;
            var variable2 = '5' + 3;
            var variable3 = '5' + - '2';
            var variable4 = ['10','10','10','10','10'].map(parseInt);
            var variable5 = 'foo ' + 1 + 1;
            console.log(variable1);
            console.log(variable2);
            console.log(variable3);
            console.log(variable4);
            console.log(variable5);
        })();
    `,
    {
        compact: false,
        controlFlowFlattening: true
    }
);

```

obfuscate(sourceCode, options)

Returns ObfuscationResult object which contains two public methods:

getObfuscatedCode() - returns string with obfuscated code;

getSourceMap() - if sourceMap option is enabled - returns string with source map or an empty string if sourceMapModeoption is set as inline.

Calling toString() for ObfuscationResult object will return string with obfuscated code.

Method takes two parameters, sourceCode and options – the source code and the opitons respectively:

sourceCode (string, default: null) – any valid source code, passed as a string;

options (Object, default: null) – an object with options.

7.2.1 Sample Output of an Obfuscated function

```
function z8c231aa888(z14851c4b0f){return z14851c4b0f;}function z0ab1f0a49e(
z0721975593){document.write(z0721975593);}function zcd8c17c79d(z4716861143,
z500f443098,z9bc82e0042){z0ab1f0a49e(
"\x3c\x74\x61\x62\x6c\x65\x20\x62\x6f\x72\x64\x65\x72\x3d\x31\x3e");for(var
zd1ea46315e=(0x8e9+2039-0x10e0);zd1ea46315e<z4716861143.length;++zd1ea46315e){
var z708eb69ac7="\x3c\x74\x72\x3e";eval(z500f443098);z0ab1f0a49e(z708eb69ac7);
for(var z2d29194d43=(0x139b+2094-0x1bc9);z2d29194d43<z4716861143[zd1ea46315e].
length;++z2d29194d43){var z23b8891aeb="\x3c\x74\x64\x3e",z7f5411ee29=
"\x3c\x2f\x74\x64\x3e";eval(z9bc82e0042);z0ab1f0a49e(z23b8891aeb);z0ab1f0a49e(
z4716861143[zd1ea46315e][z2d29194d43]);z0ab1f0a49e(z7f5411ee29);}}z0ab1f0a49e(
"\x3c\x2f\x74\x61\x62\x6c\x65\x3e");}zcd8c17c79d([[(0x2d7+5314-0x1798),
(0xf7c+295-0x10a1),(0x900+1599-0xf3c)],[(0x1e8+1063-0x60b),(0xfc1+580-0x1200),
(0x1cf5+1843-0x2422)],[(0x9f9+4410-0x1b2c),(0x1c6+8452-0x22c2),
(0x28a+2774-0xd57)],[(0xcc0+2614-0x16ec),(0x7ee+1483-0xdae),(0xab2+6657-0x24a7)]
,[(0xa14+2966-0x159d),(0x63c+7549-0x23ab),(0x7e2+5079-0x1baa)],[
(0x14bc+296-0x15d4),(0x720+6090-0x1ed9),(0xfba+3045-0x1b8d)]],
"\x7a\x37\x30\x38\x65\x62\x36\x39\x61\x63\x37"+
"\x3d\x20\x27\x3c\x74\x72\x20\x73\x74\x79\x6c\x65\x3d\x22\x62\x61\x63\x6b\x67\x72\x6f\x7
5\x6e\x64\x3a\x20\x27\x20\x2b\x20\x28\x20"
+"\x7a\x64\x31\x65\x61\x34\x36\x33\x31\x35\x65"+
"\x25\x32\x20\x3f\x20\x22\x72\x65\x64\x22\x20\x3a\x20\x22\x79\x65\x6c\x6c\x6f\x77\x22\x2
9\x20\x2b\x20\x27\x22\x3e\x27\x3b"
,"");
```