Trustless 2 Factor Authentication for OpenSSH server leveraging Blockchain and Ethereum Smart Contracts

Varun Amrutiya (2017H1120244P), Siddhant Jhamb (2017H1120243P), Pranjal Priyadarshi(2017H1120242P)

Motivation

Two-Factor Authentication(2FA) is widely prevalent in banking transactions, Email, or in any highly secure web service available online/offline. Generally, it is provided through methods like one-time-passwords, hardware tokens, or SMS messages with secret codes. This means that to provide 2FA there is a need to tie-up with third party software or entity providing an additional layer of security by inherently trusting the third party for the services offered.

In an enterprise data center large number of servers resources, handling remote access to server is mainly handled by OpenSSH server which is a very secure and industry trusted tool which is used more than 97% of time. The security of the OpenSSH protocol comes from the use of public-key pairs this calls for the need of key-management software for enterprise needs thus most of the organizations pay hefty amounts for such key-management solutions. But one security aspect we didn’t look though is how this is actually done internally, most of the key-management solutions uses on-the-fly configuration change for OpenSSH server to actually work. This means that these solutions are just wrappers and their security only depends on the security of the tool which is injecting keys in the remote OpenSSH server, once this tool is compromised the security of the server can be easily exploited. Such an key-management solution rewards the ease of handling resources but comes at a price of security. Instead of solving the challenges of ssh key-management in this project we try to toughen such weak security links by adding an extra trustless two-factor Authentication layer to OpenSSH server.

In the previous paragraph we already discussed that the security of the solution depends upon the security of the tool deployed in the server, here we propose to use Ethereum smart contract to solve that issue. In simple terms Ethereum Smart contract can be explained as execution of instruction set which cannot be ill-executed and has the properties of being trustless and byzantine fault tolerant. Here we plan to deploy a blockchain based solution which can provide 2FA using the underlying smart contract functionality to authenticate large server resources.

Salient Features

- Decentralized design
- Trustless smart contracts
- Peer-to-Peer connectivity (no single point of failure)
- Uses the blockchain provided application module rather than programming out-of the box solution.
- No need to integrate with auto-generated mailing system or SMS gateways.

- Completely free.
- Free from data storage responsibility (thnx to blockchain…)

Implementation Design

The idea is to save a user generated token(like an OTP) tied to its Ethereum account identifier on the blockchain in a hashed form. This would act as an additional layer of security when a user tries to login to that server after providing the username and password, followed by the OTP which would be matched with the value stored in blockchain.

The major components that were built to achieve the set goals of this project are mentioned below,

1. **Ethereum Smart Contract**: It acts the core part of the blockchain that will allow us to embed the functionalities required by us to fulfill the goals and provide us with an fault-tolerant execution of code.The following tasks would be carried out by smart contract with utmost security standards in mind,
    - User administration.
    - System administration.
    - Storing secure access tokens (storing otp in SHA3 form)
    - Providing access to system
    - Revoking access of system in case of multiple incorrect attempts
    - Provides workflow administration via approving/disapproving requests based on staff requirements.

2. **DApp** (Decentralized Application): It will act as an web based interface via which staff and user can communication with the smart contract existing on the blockchain. In technical aspects it will be a simple html web page which will connect to any ethereum node via is JSON-RPC API. Its sole purpose is to give an basic html based UI to smart contracts so that users can interact with it. The functionalities provided by would be,

- Provide an web based UI design for smart contract
- Connect html UI and ethereum blockchain using web3.js library
- Take inputs from user and store them in blockchain.
- Loads user credentials/keys.

Note: It is just an HTML webpage there is no transfer of credentials/keys analogous to web service. It only performs task of signing data with the keys and pushing it to blockchain where blockchain will perform various security checks.

3. **Metamask Plugin**: This browser plugin allow the users of the system to load and unload their credentials in the browser storage on the fly while working with DApp. With this plugin the users are no longer required to setup your own personal ethereum node to interact with the smart contract. This plugin can be configured to digitally sign all the web3 transaction and send it to a common gateway where all the transactions are added to the pool and processed by the miners. As part of real world deployment, we need only one single ethereum endpoint where all the users can just configure this endpoint in the plugin and start making their transactions.

4. **PAM** (Pluggable Authentication Module): Its is originally an functionality provided by Linux operating system to extend its authentication methods by loading user defined modules. We programmed and special PAM module which will act as a bridge between blockchain and OpenSSH server. Based on the token input during the login screen PAM would query the authenticity of token via blockchain and based upon the response will inform the OpenSSH server regarding the decision of granting access. All the servers as a part of this ecosystem are required to have their own configured ethereum and PAM module loaded for its proper functioning.

Tools Used:

OS: Ubuntu
Remix-IDE for contract testing and deployment.
Ganache: A ethereum blockchain for testing and development.
MetaMask: A browser plugin for key storage.
Python-Flask: A simple web server to serve the DApp

Use Case:

We have added various set of functionalities for User as well as Staff who are the main actors of the system, some of them are mentioned below,

Staff Functionalities (in DApp):

- Register System: To register name and ethereum address of the system.
- Register User: To register name and ethereum address of the servers.
- Current Request: To view and approve requests made by users for access of the system.
- System Status: To see a complete list of registered user, registered system and also see list of revoked system and unlock them if required. This provides a birds-eye view of whole infrastructure.

User Functionalities (in DApp):

- Request System Access: To get a permanent access of any system as part of this.
- My Systems: To show all the systems allocated to the user by the staff.
- Generate OTP: To generate an random 6 digit pin and store it SHA3 form in the blockchain for later use.

Workflow (User/Staff perspective, Step-by-Step):

1. Staff register an system which is part of this infrastructure
2. Staff also register the set of users who can communicate with the smart contract
3. User sees all the system available, and places the request for system which he wishes to use.
4. Staff sees all the requests made by the user and accepts them.
5. User can now generate an OTP and save it in blockchain for the approved system.

(User would be required to configure the metamask plugin before hand with his private keys and Ethereum endpoints IP address and Port.)

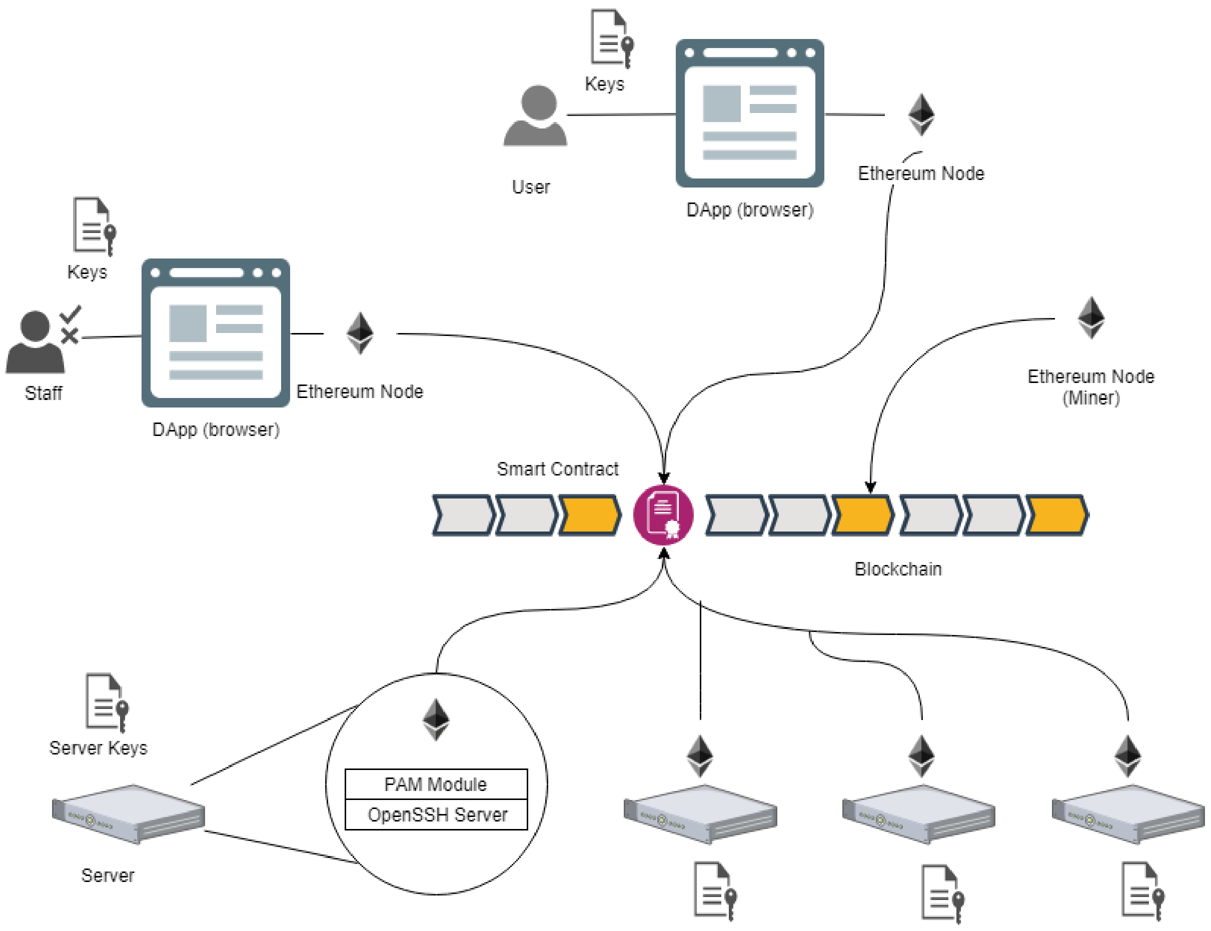
Workflow (PAM perspective):

1. User initiates and SSH connection to the system which he wishes to use.
2. Initially the SSH will authenticate the user using normal username and password as a part of normal ssh authentication.
3. PAM module will check in the blockchain whether the system is revoked or not. If revoked it won't allow any access to the user.
4. If the System is not revoked it will provide an prompt to the user to enter his OTP for this access.It will retrieve all stored token for particular system at this point for future use.
5. If OTP is correct it will be deleted from the blockchain so that it cannot be reused.
6. If OTP is incorrect appropriate message will be show and on 3 incorrect OTP entries the system would be marked as revoked in the blockchain and won't allow any further ssh access until staff de-revokes it.

(PAM module would pre-configured with contract address and System ethereum address at compile time.)

Block Diagram

A rough block diagram showing the details as explained in the earlier section, from a high level, is shown below.


Keys
User
DApp (browser)
Ethereum Node
Keys
Staff
DApp (browser)
Ethereum Node
Ethereum Node
(Miner)
Smart Contract
Blockchain
Server Keys
Server
PAM Module
OpenSSH Server