Introduction to Blockchain and Bitcoin

Final Project Proposal

Brett Bloethner

Project Type (Simulation or traditional-blockchain code/prototype)

For my final project, I chose to do a traditional-blockchain based prototype of a terminal based application that helps improve the integrity of syslogs by acting as a syslog receiver and storing log data in a blockchain. Unlike many of the projects discussed in the course, my project revolves heavily around the creation of a NodeJS based terminal application backed by the Web3 Ethereum client. The incorporation of the Web3 Ethereum client is really the meat of the application and without it the application wouldn't involve the blockchain and would be relatively useless as a result.

Project Main Idea

The main idea behind my project revolves around using a special storage mechanism for server logs. Often time log file ingestion, processing and storage specifications are overlooked by software engineers and administrators. However, system logs play an essential role the discovery and investigation of computer system intrusions. One important concept behind good log keeping is the idea of log data integrity and security. Logs and log files are only useful if the user knows the logs are accurate and haven't been tampered with. Blockchains happen to have a few powerful features that help with this sort of data persistence and assurance.

One of the key features of all blockchain systems is in the nature of the immutable design of the blockchain itself. Blockchains incorporation of data hashing, chaining and merkle trees allow users to easily and reliably verify the integrity and accuracy of the data they're viewing. At its core, blockchain incorporates what is basically a tamper-proof ledger. This design happens to be perfect for storing log files or, if performance is a concern, log file digests. By incorporating blockchain technologies into the storage of syslogs, we can create a system that the user can rest assured will provide a truthful and accurate series of events of what happened in their computer systems. With this, log files become much more reliable and more useful when it comes to cyber security and incident response.

Target Market

I think the market for this sort of application is pretty large and primarily includes IT administrators and software engineers. Logging is often an essential part of networked device deployment as well as software and web application deployment. Individuals skilled in the art will know that logs are only as good as they are in their consistency and integrity. Because of this, assurance that logs are truthful and haven't been modified are integral to any logging or log analysis tool. This opens up a large and diverse market for this sort of tool. Specifically, the

application uses the syslog standard and while any syslog device can be adapted to work with the application, I will be using syslogs from my enterprise security gateway for testing in demonstration. I think it's important to note that this idea has the potential to be used in any computer system despite the systems users' technical aptitude or whether or not they know it's even being used. However, integration and maintenance would be the job of IT administrators or software engineers.

Workflow-logistics & Presentation

I propose that this application take the form of a terminal based web server. To the user, all of the input and user interface is conducted through their preferred terminal program capable of running the NodeJS runtime. Internally the application consists of a web server capable of receiving a steady stream of syslogs as well as code related to processing those logs, grouping them together, storing them in a traditional database, creating a digest for each log group and subsequently storing the digest hash in the blockchain. All of the application will be written in Typescript and transpiled into javascript for use in any modern computer system capable of supporting both the latest NodeJS runtime environment and an HTTP web server.

The presentation of my demo will consist of two key parts. The first part is step by step documentation outlining how to get the web server up and running and how to view logs and the blockchain to such an extent that the user can confidently see that their log digests are stored in the blockchain. The second part of the demo will consist of a short video of me outlining the project as I give a walkthrough of the software application, the step by step documentation in action and the verification that log files are correctly stored by the application.

Last Mile Analogue

The application I'm proposing for my final project interfaces primarily with other computer systems. It's because of this that my application doesn't have a very strong last mile analogue like some of the examples we've reviewed in class (the baby registration system). Still, the data in the application and be misinterpreted or incorrectly delivered to the application and stored in the blockchain permanently. An example of this scenario could be if the computer system generating the syslogs was delivering incorrectly timestamped syslogs or if the system reading the blockchain and log entries was incorrectly reading or presenting the data from the blockchain. This sort of error could be hard to detect since the application has no way of determining what is or isn't a valid log. However, malicious actors who want to attempt to take advantage of the system by inputting bad log entries could be easily stopped by using standard network security practices, incorporating a higher level of security into the HTTP web server, or by bypassing the syslog requirement altogether and instead creating agent applications that live on the computer systems and securely forward logs to the log file receiver (this app).

Scaling Difficulties and Solutions

The obvious concern when it comes to scale is the sheer size of long dated log sets. It's not uncommon for computer systems to produce millions of logs a day. Storing these in the blockchain would be extremely inefficient. The solution is to store log digests as transactions rather than adding each individual log to the blockchain as a transaction. This digest would include a hash that's a derivative of all of the log entries included in the log set. This design pattern allows each blockchain transaction to represent and verify hundreds or even thousands of log entries rather than just one. Another possible concern is log entry throughput. A network of hundreds of desktop computers and servers could easily create thousands of log entries every second. Because of this, I choose to use UDP socket based connections because of UDPs low latency and loss tolerating design. However, with larger systems like data centers this design still probably wouldn't be enough. I anticipate that at a large scale, the log receiver would need to be broken into multiple nodes and broken off from the blockchain interfacing application. This would leave two applications. One application would receive and store the logs and create the log digests. This application could live independently on tens or even hundreds of servers in a computer network (not agents, just numerous log receivers). The other application would receive the log digests from the first application and write those into the blockchain. A lot fewer of the latter applications would be required in a large computer network.

Fraud & Malicious Behavior

Log files are natural targets for malicious actors in cyberspace. Its standard practice for hackers to modify log files after they compromise a system. Doing this allows them to leave without a trace, making it harder for the computer system owner to notice they were ever hacked and even more difficult to investigate the intrusion to discover who was responsible. I see a handful of different types of malicious behavior that attackers might try to leverage against my project. It's important to note that all of these exploits would require that the attacker is already inside the computer network and has somehow bypassed the authorization required to access these resources in the network. I've outlined them in the bullet points below.

- Stopping log creation buy performing a sybil attack and building the blockchain to their liking in real time
- Rewriting logs by performing a sybil attack and rewriting the entire blockchain with their own log mutations included
- Modifying a log in the standard (non-blockchain) data persistence layer
    - This would break the blockchain hashes, notifying the user of a compromised log set
- Deleting a log in the standard (non-blockchain) data persistence layer
    - This would break the blockchain hashes, notifying the user of a compromised log set

Without a Blockchain

Special handling and treatment of log files is not a paradigm shift in software engineering or IT administration. There are many current solutions out there when it comes to ingesting, processing and storing computer system logs. A secure solution analogous to what I propose in my final project would most likely still incorporate some sort of hashing mechanism and might go as far as to even include a hash chain mechanism but fall short of incorporating a whole entire blockchain. The overall design would still be similar and be comprised of an application that acts as a server that can receive a steady stream of logs and process those logs then store them in a data store somewhere. Where that system would fall short in comparison to mine is the fact that the system might not incorporate the blockchain and could opt to not incorporate hash chaining of any sort. In that scenario, the application would simply store logs and would have no design features ensuring log integrity beyond maybe simple user/system authentication.