Questions

* How does the system deals with fake time, changed by configuration on OS, virtualization/emulation or firmwares?
  + The incremental log of operations can reveal times that are out of sync, for cases with fake time done through OS or firmware configurations. The hacker would have to modify the logged data in order to avoid this kind of monitoring;
  + Fake time done through virtualization/emulation can maintain a valid time using backtrack techniques.

*The current version does not implement any model of protection agains fake time. We only have the key expiration period and store the key in a protected memory. It would be good if the log analysis can help to reveal this situation.*

* The monitoring should consider only the operations done through app functionalities or should monitoring OS operations too (e.g navigations or file access using the native filesystem)?

*It can monitor anything, but so that the functionality or usability of the whole system doesn’t suffer. The user of app should not be complaining that everything is slow because of monitoring.*

* About the paper, the Sections 4 and 6 have some overlapping?

*You mean the section mentioned in the beginning of the paper? The section 4 and 6 are – other works and proposed solution. In section 4 we discuss the previous works and problems. In section 6 we describe our work.*

* How the system will deal with dictionary attacks that bypass client, through direct access to encrypted files?

*The dictionary attack on AES key or ABE key is inefficient. This is a cryptography with really big keys. The dict attack can be efficient on a PIN or a PASSWORD. This cannot be done without client. Otherwise hacker has to steal all the intermediate keys and encrypted files from the client (in one session, before they are modified) and try to perform to restore the shared secret (the ABE key)*

Points and suggestions

* It is necessary to be aware about the network traffic necessary to transmit the monitored data
  + The communication between client and server can be done if the client does not have the current model (matrix, data distribution or other model calculated by server), or after some local data modification, that implies into model actualization;
  + Use a data transferring scheme like git does, based on incremental local modification saved into a local database, followed by delta transferring and data compaction (this is a high level view).
  + Git's approach make possible to evaluate the log before apply the "push" on server's data. This can be useful for avoiding the server contamination from a compromised client, identified through log analysis.

**So your suggestion is git approach? Can you describe it in greater detail or provide some links so that I can see how it suits into the current client model and how it affects the security?**

* How will we demonstrate the efficiency of the system? It is not enough to show that it works, but we have to show that it works efficiently? (Edison's Question)  
  + Suggestion:
    - Comparisons between anomaly detection techniques suitable for offline threats
      * DIstribution Based Analysis;
      * HMM;
      * MOS + Similarity Analysis;
      * DNN;

**Actually, when I saw the proposed mobile device monitoring systems – they are all very simple, there is no correlation analysis, just the collection of data. So our point of efficiency can be:**

* **First data analysis in the offline (which no one deals with)**
* **First sophisticated correlation analysis on a mobile infrastructure**
* **Analysis performed at a low computation cost (this point we will have to proof, just to check the time of operations on a mobile with the data analysis happening in the background)**
* What will be the measured metrics for the quality measurement of the proposed solution? (Edison's Question)  
  + Suggestion:
    1. Anomaly Identification Rate (Success, False Positives, Lack of identification);
    2. Adaptability to different scenarios or data distribution;
    3. Processing Time;
    4. Energy consumption.

**I agree. Adaptability is a bit hard to proof with testing though, so I would suggest to analyze first 2-4 most basic scenarios and the points 1,3,4 on them.**

* Would be nice to know the article that was rejected by SAC and its review?

**I will send you the copy of the review and article.**

* Malicious Behavior Detection
  + Data Modelling
    - Matrix of File x Time (This structure can be used to evaluate the features related to files, over time);
    - Matrix of Directory x Time (This structure can be used to evaluate the features related to directories, over time);
    - Matrix of Access Attempt x Time (This structure can be used to evaluate the features related to access attempts, over time);
  + Distribution based analysis
    - I suppose that a small set of files concentrates the file operations, maybe pareto or poisson distributions can fit well;
    - To identify the probability distribution followed by the selected features;
    - To evaluate distribution violations;
    - Architecture
      * Client monitors the selected features;
      * Client store the collected data locally;
      * Client sends the collected data, timely;

**Yes the client can send the data to server once it connects to server and renews the keys. So that server is aware – ok, the client is fine, so I can renew the keys.**

* + - * Client applies MOS and similarity analysis to identify anomalies;
      * Server updates the models (e.g calculates the matrices for MOS and similarity analysis);
      * Server implements lambda architecture for combining batch processing and online processing;
        + lambda architecture ([25] N. Marz and J. Warren. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications, 2013.)
        + Bach processing = Model updating and other heavy processings;
        + Online processing = Anomaly analysis, if the calculus on client side have restrictions of energy or processing capacity
      * Client obtains updated models and data, timely
* Hypothesis
  + Large file operations can indicate anomalies and be identified through MOS (covariance) and similarity analysis;
  + Sparse file operations, with low number of operations distributed over different files or directories, during short period of time can indicate anomalies and be identified through MOS (correlation) and similarity analysis;
  + Typical file selection by browser type
    - Without access by command line or other automated way, the file selection only can be done through navigation and file selection;
    - Fast navigations can indicate an automated behaviour, such as a bot for automated file request;
    - Navigations may follow patterns according to application type (web browser, mobile app), such as time between navigations, that can be small on web browsers in comparison to mobile apps.
    - Short time between navigations can indicate an anomaly
      * MOS based on correlation analysis can effective to identify

**I don’t know whether it can help, but th mobile client as well as desktop ui has its own browser, i.e. files are opened within the application. I will send you the link to the ui, to see how it works.**

* + File access in contrast to mean time required for navigation (reachability)
    - The time between file requests is lower than the mean expected navigation time between the files.
    - This pattern is analogous to credit card fraud detection according card usage on different locals at the near time or in an interval of time lesser than the required time to go from a place to other one;  
      distributed file modification over a range of time (window)  
      Repeatedly downloading the files that are already on the device;
  + Short time between attempts can indicate a brute force attack;
  + Correlated time between pass attempts can indicate a dictionary attack programmed to wait for a specified time;
  + Correlated time between the result and new pass attempt can indicate a dictionary attack programmed to wait for a specified time;
  + Number of wrong pass attempts per time;
  + Number of wrong pass attempts after a significant number of successful attempts, without recent password/PIN changing;
  + The selected features should follow a known probability distribution, that can be identified and used to evaluate distributio violations

Proposed Features:

* Access Time of Directories;
* File Request Time and Location (Directory);
* File Selection Time and Location (Directory);
* File Download, Start Time, End Time and Location (Directory);;
* File Upload, Start Time, End Time and Location (Directory);;
* *File Encryption, Start Time, End Time and Location (Directory);;*
* *File Decryption, Start Time, End Time and Location (Directory);;*

***This is done by the client automatically, so it does not depend on the user behavior. I don’t think it is necessary, we logged it for the purpose, to show that client works fast.***

* File Update Time and Location (Directory);
* *Enter Pass (Time and obtained status);*
* *Enter PIN (Time and obtained status).*

***Here you mean to count the tries of entering the PIN and PASS?***

***Yes***

Good morning, all

Thank you, Thiago, a good work, and your mind map contains some very important observations!!! I think we can really do a unique work.

I am attaching my answers and comments. I am waiting the analysis of basic scenarios:

1) A hacker - downloading and uploading a bulk of files

2) A malicious user - using and old password and downloading a bulk of files.

These scenarios can be analyzed in online and offline mode.

I will send you the SMART article and its review and the link to the storgrid browser interface.

Brs, Tanya