Panalyt

Panalyt Security FAQs

1. What frameworks have been used for the Panalyt application?

- Google App Engine with Java EE, Firebase, React with Typescript

2. Does Panalyt have an easily discoverable way for external researchers to report security vulnerabilities in its systems?

- Yes,external researchers can report security vulnerabilities either at contact@panalyt.com or through the main Panalyt application. . Incoming reports are timely reviewed and triaged.

3. What are Panalyt's security policies around HTTPS and Mixed-Content Risks?

- The Panalyt web application is reachable exclusively over HTTPS. Even if the user manually edits the URL to start with `http://`, it won't work or it will redirect to `https://`.
- We regularly review the SSL configuration/ cipher suite advertised by the server and the protocols it uses.
- We offer forward secrecy for clients that support it and the Panalyt server supports ECDHE and DHE ciphers that offer forward secrecy.
- We have implemented the necessary steps required to protect our SSL/ TLS private keys on our web servers.
- We have specific controls in place to prevent mixed-content issues.
- We have HSTS configured with a max-age value of at least 6 months.

4. What are Panalyt's security policies around authentication and authorization?

- Our application requires regular users to log in. Most features aren't available without logging in.
- In addition to an interface for regular users, our application provides an administration interface.
- Our application features complex user management. Various roles can be assigned to user accounts.
- There are 4 simple roles :
    - **SUPERADMIN** : Access to all company level data including billing, confidentiality
    - **ADMIN**: Access to all the company level data excluding billing, confidentiality
    - **EMPLOYEE**: Access to employee permission set according to the ADMIN/ SUPERADMIN.
    - **NON-EMPLOYEE**: Access to employee permission population set by the ADMIN/ SUPERADMIN.
- Every employee by default has reporting line permission : This means that a manager has access only to the people who directly/indirectly report to him.
- Special Location/Organization level permissions can also be made to a EMPLOYEE/NON-EMPLOYEE as configured by an ADMIN/SUPERADMIN.
- The Panalyt application is integrated with the OpenID Connect / OAuth2 Login SSO mechanism
- We're using a standard OAuth2 library, and we update it when security fixes are released.
- Panalyt uses Firebase authentication to have the login credentials.
- Panalyt stores passwords using a secure cryptographic one-way hash function (such as SHA-256) of the salted password
- The initial password or a link to set the initial password is sent to users by email.
- When users log in for the first time, confidential information will already be present.
- A password reset link is sent via email to the user's registered email address.

5. What are Panalyt's security policies around Authentication Cookies and Sessions?

- The Panalyt application's authentication uses sessions that are only valid for the browser tab in which Panalyt runs.
- The web application framework we use has a built-in session ID mechanism.
- Our session IDs are randomly generated strings or numbers.
- We store a signed token as a cookie to indicate that the user is successfully logged in.
- We use Firebase SDK to create session IDs.
- The Panalyt sessions are automatically timed out after 20 minutes of user inactivity

- The Panalyt application uses a secure cryptographic pseudo random number generator(PRNG) to generate session IDs which does not allow the state of the generator to be recalculated from its output and the entropy of the session is sufficient for making brute-forcing infeasible.
- The Panalyt application offers a "log out" button or link that, when clicked, not only terminates the session (deletes cookies from the client) but also invalidates the entire session ID.

6. How does Panalyt handle User Authorization and Authorization-Related Web Vulnerabilities?

- Our application enforces both Horizontal Access Control and Vertical Access Control on the server side. We have processes in place to make sure nothing slips through the cracks.
- The Panalyt application ensures all state-changing actions against XSRF are protected by protecting requests that change the state with tokens that are bound to the user they were generated for, and that expire after a certain amount of time .
- Our application makes use of JSONP and other formats that set variables or calls functions with non-public information.
- The Panalyt application uses a clear white list of domains for which cross origins are allowed. This is handled both on the server-side as well as the client-side.
- There is no external way in which this can be breached, including clickjacking.

7. How does Panalyt handle common Web Vulnerabilities such as XSS?

- Our application has a central choke point where all user input is validated and escaped, depending on the context in which it will be interpreted.
- Many ways of protecting XSS vulnerabilities are implemented:
  - Client-side white listing for making sure that the environments are linked to the correct servers.
  - Server-side white listing makes sure the domain origin is verified domain from Google App Engine
  - Validation and strong typecasting to make sure that all the endpoint requests are in a specific schema/input.
- We use React with Typescript to create the application. React outputs elements and data inside them using auto escaping. This means that if variables containing the state of the application were somehow infiltrated by an attacker with some <script> tags, React would simply ignore it and render it as a string.
- The libraries that we use have been thoroughly tested and heavily endorsed/tested/used. Libraries for requests are also strongly typed which ensures that the integrity of the data is always maintained, otherwise no requests are fired. All requests are also personally checked on the backend for the same valid schema otherwise no data is returned.
- The download link is a one time generated link by the server using a salted and complex hash for the specific user and it also has an expiration time of 3 minutes. This ensures that only the requested data is exposed to only a specific user and this is possible only for a very small period of time.
- We take specific steps to protect against DOM-based XSS.

8. How does Panalyt protect against injection attacks such as SQL injection?

- Our application uses an object-relational mapping (ORM) framework. When we need to manually construct queries or conditions, we pass user input to the database via stored procedures.
- We use JOOQ to build the SQL queries using prepared statements. See https://www.jooq.org/doc/latest/manual/sql-building/bind-values/sql-injection/

9. Where does Panalyt store data from file uploads and what restrictions does it impose on the upload file type?

- Panalyt stores data from file uploads in its Google Cloud Platform instance.
- We only allow .csv, .png and .jpeg uploads.
- We verify the file type by checking the file extension on the server side and the content type that is sent by the user.

10. How does Panalyt use cryptography to ensure the confidentiality and/or integrity of information?

- **Authentication** : OAuth2 with SHA-256 provided by Firebase with robust and configurable rules.
- **Files and Storage encryption**: Files are encrypted with the option of a customer-managed encryption key created and managed through Cloud Key Management Service or customer-supplied encryption keys. (https://cloud.google.com/storage/docs/encryption/)
- **Persistent Data Objects on Google Datastore**: Object's data and metadata is encrypted under the Advanced Encryption Standard (AES), and each encryption key is itself encrypted with a regularly rotated set of master keys. (https://cloud.google.com/datastore/docs/concepts/encryption-at-rest) . All our namespaces are also well defined and abstracted so that no two datasets mix with each other.
- **SQL cache Encryption and Abstraction**: All the data cache is also encrypted with again clearly abstracted namespaces along and are transmitted with AES-256 encryption.
- **Overall REST application** : The application using Google Tink library to cater to all the cryptographic needs. All the signatures are using a Symmetric key provided by this library and the configuration is stored securely by Google. (https://cloud.google.com/security/encryption-at-rest/default-encryption/)

11. How does Panalyt test the security of its code during releases?

- Panalyt engineers use code review for all changes to the production application and take particular care of reviewing for security issues following the OWASP Top 10 guidelines.
- Code changes are scanned for new security vulnerabilities on a continuous basis. PRs are blocked when new vulnerabilities are found.
- Docker containers are also scanned for security vulnerability on every new push to the registry.
- We use unit tests to verify that the application enforces access control.
- Our QA process explicitly includes testing for security issues that might have been introduced in new releases.

12. How does Panalyt handle post launch monitoring?

- We have procedures in place to log and monitor for unexpected crashes, exceptions, and other error conditions. If something looks suspicious, a security-conscious engineer evaluates it.

13. Whom and how should I contact Panalyt about any securities issues or concerns in the application and how?

- All in-app security issues or concerns can be brought to the notice of the Panalyt team using the Report Bug option in the left navigation tray and concerns from external parties can be emailed to contact@panalyt.com or sent using the Intercom chat widget on our website and our team will immediately investigate the concerns.