Python

unknown

Sep 12, 2019

MAIN DOCUMENTATION

CHAPTER

ONE

WHITE PAPER

1.1 Motivation

Almost every system needs to keep audit logs – information on "who did what and when". For example:

- Who changed the patient's medical history
- Which bank employee viewed the customer bank account
- Which CCTV camera has taken a particular video and when

It's not just important who did it, but also what exactly they changed or viewed and when. The whole set of details (actor, action, entity modified, details of the action) has to be stored. But simply storing it won't be sufficient, as the stored data can be modified – there's a need for securely storing it and guaranteeing its integrity.

Whenever confronted with the need to store an audit log (which is the case for almost all applications), companies go for custom home-grown solutions. In the best case scenario they use some plug-in to their database access technology (e.g. an ORM) that automatically handles modifications. Very rarely these solutions cover the following set of requirements:

- Business-logic level events as well as insert/update/delete/get events have to be stored – it's not sufficient to just see the database rows that were updated, sometimes high-level operations like "basket checkout", "bank transfer initiated" or "medical examination performed" need to be stored as well, as they carry more meaning to the business than a bunch of database table updates.
- The audit-log has to be tamper-proof, i.e. nobody, even system administrators, can alter it without being detected. That way the audit log has a more significant legal strength in courts. And the business can make certain guarantees to their clients.
- The audit-log has to be easily searchable and navigable – anyone with proper access (a high level manager, line manager or even external auditor) can easily see sequences of events that lead to a particular issue

And when speaking of issues – data manipulation attacks, both from insiders and outside attackers, are a serious threat nowadays. Wired has put it in the top security threats prediction for 2016 . Without taking additional measures, no business is safe from having their data manipulated for the benefit of other parties. And often without even detecting that until it's too late (this is why NIST recommends as well as OWASP recommend using audit logs)

Last, but not least, public sector entities, as well as regulated businesses, need to comply with a set of security regulations. Ticking the "compliance" box may not be that hard, but the liabilities for not protecting customers'/citizens' data are still a serious risk to be taken into account.

Speaking of regulations, the General Data Protection Regulation in the EU mandates that every system that processes personal data (which is true for most systems) must have an audit log and not complying with the regulation may lead to significant fines for any business.

While many companies think they have audit logs, they are almost never properly protected. And as one book on ISO 27001 warns:

System logs need to be protected, because if the data can be modified or data in them deleted, their existence may create a false sense of security.There needs to be a product that addresses all of the above concerns and observations, and LogSentinel aims to be such a product.

1.2 Features

LogSentinel offers the following core set of features

- Easy integration – the RESTful API is very simple and can be plugged in new and existing systems painlessly. Using one of the provided client libraries simplifies this process even further. No need for complex setups or lengthy integration processes.
- LogSentinel runs in the cloud, but can also be deployed on-premise. This gives sufficient flexibility, based on the particular buainess case – small businesses may not need or be able to manage their own additional servers (even though the setup is easy), whereas big enterprises may not wish to trust their sensitive audit logs to a cloud solution.
- The audit log is protected by a technology similar to the way the integrity of the blockchain is protected. No modification to the log can be made without becoming evident almost immediately. This also bears legal strength – e.g. if a given event data is not tampered with, it is guaranteed that the event really happened.
- The authentication-specific API endpoints provide a way for actors to digitally sign their logins, for example, which according to the relevant legislation may give additional legal strength in case an issue is brought to court.
- A user-friendly dashboard is provided for searching and navigating through the audit events – for example "who modified that user account at 6 pm yesterday" or "what exactly did employee X do on his last shift"
- Protecting the integrity of all the data – while the integrity of the data itself is not the goal of LogSentinel (as opposed to protecting the integrity of the audit log), having a full secure audit log on all data changes means that the audited data is also protected – you can always trace the data modifications in the log and verify whether a given change occurred "naturally", or was altered inappropriately.

1.3 Technology

The technology used is based on a number of peer-reviewed papers, e.g. Audit Logs to Support Computer Forensics [1] by Bruce Schneier and John Kelsey and Efficient Record-Level Keyless Signatures for Audit Logs [2] by Ahto Buldas et al.

The implementations is customized to account for the various business cases that should be addressed. Some properties have been dropped or moved to the client/client libraries, and the need for scaling the product and providing multi-tenancy has lead to a bit more complicated processing logic.

Notably, the technology relies on consecutively hashing all incoming audit log events, where each subsequent hash is formed by the data of the current entry combined with the hash of the previous entry. That way the audit log entries form a hash chain that cannot be "broken" – i.e. any manipulation to any of the entries will result in invalid hashes from that moment on.

In addition to forming a hash chain, entries form groups, which are timestamped using a client-provided timestamping authority. The group is represented by a single value, which is the root of the merkle tree of the hashes of all the entries in a group. A merkle tree is a data structure used in the blockchain to guarnatee the integrity of each block. The timestamping provides additional integrity guarantees and legal strength (EU Regulation 910/2014 defines the rules for trusted timestamping)

The integrity of the log is guaranteed even if a malicious actor gains access to the log database. So you don't have to trust the LogSentiel cloud service, or even your own employees who manage the hosted solution. In order to achieve that, there is one "catch" – the latest hash in the hash chain at a given moment has to be kept in an unmodifiable way.

When you have a given hash, the fact that it is present in the audit log guarantees that it hasn't been tampered with. And vice-versa – if a hash that was previously stored is missing from the audit log, it means the log has been tampered with. There are multiple ways to store these latest hashes in an unmodifiable way, and LogSentinel supports:

- Store it on the Ethereum blockchain. Public blockchains are the perfect candidate, as they are immutable – once an entry is stored there, it cannot be removed. Transaction fees are relatively cheap, so LogSentinel can regularly push the last known hashes to the Ethereum blockchain
- Print it on paper – it can be published in newspapers (which is less practical), or printed on a blank paper and stored in physically protected cases, or snail-mailed to multiple stakeholders, including auditors.
- Store it on a write-only medium. Be it a CD-R, or more generally – any WORM storage
- LogSentienl returns the latest known hashes which you can decide when and how to store.
- Email it to multiple stakeholders – while email can be manipulated as well, having it distributed to multiple people, potentially with different email servers, increases the complexity of changing the hash in all places.

1.4 Conclusion

LogSentinel is a secure audit log service that is simple to integrate and guarantees the integrity of all your audit data. It can be integrated in any system with minimal effort. The use of state-of-the-art cryptography and original research makes sure the data is verifiably protected and this can be proven to both customers and law enforcement.

CHAPTER

TWO

GETTING STARTED

2.1 Overview

LogSentinel is a service that lets you log all the business logic events in your application and allows searching through them. The main feature, however, is the guaranteed integrity of the log – once it enters the system, it cannot be modified without being detected. That way it follows the OWASP recommendations on audit trails.

LogSentinel can be used either directly through its RESTful API (e.g. when your project is entirely under your control), or via one of the plugins for 3rd party software.

2.2 RESTful API

In order to get started, you can take a look at the simple RESTful API for sending audit log events . There are four methods, all of which accept an arbitrary request body – it can be JSON, base64-encoded binary format, or (again encoded) encrypted data.

- `/api/log/simple` – this endpoint accepts just any request body (using POST) and stores it as a new event entry. It is the recommended endpoint if you want full secrecy of the data – you should encrypt it at your end and just send us the encrypted payload. Note that you won't be able to use the search capabilities that way.
- `/api/log/{actorId}/{action}` - this endpoint accepts an actorId and an action in addition to the request body. The actorId is normally the userId that performed the action, and the "action" parameter is an action specific to your system that is not about a particular entity – e.g. "PERFORM_SEARCH", "START_BACKGROUND_PROCESS", etc.
- `/api/log/{actorId}/{action}/{entityType}/{entityId}` – this endpoint accepts the actorId, an action (any action, including the recommended INSERT/UPDATE/DELETE/GET and custom ones like CHECKOUT_BASKET, DISCARD_ITEMS, etc) and the entity type and ID. This is intended to store database-related, entity-oriented events. This is expected to be the most often used endpoint in your application. It is also the most transparent one, since generic functionality can be plugged in to automatically send the events to LogSentinel. Some of the client libraries provide such features. It is recommended to pass "old" and "new" values in the body of UPDATE events in order to be able to reconstruct the whole data modification process, similarly to how event sourcing works.
- `/api/log/{actorId}/auth/{action}` – this endpoint is specifically intended for authentication events – it only takes an actorId and authentication action (LOGIN, LOGIN_FAILED, LOGOUT, SIGNUP, LOGIN_AS (for staff acting as user), AUTO_LOGIN (in case of remember-me functionality)). In order to have some additional legal strength you can have the user sign their login event with their password (e.g. like described here) and pass the result of the signing using the custom headers Signature and User-Public-Key

If the metadata isn't critical and doesn't need to be encrypted, you can only send an encrypted body and still use the more specific endpoints. If bandwidth is an issue, e.g. in IoT context, the body itself can be a hash of the original resource (e.g. a photo)

The response contains the ID of the inserted log entry and the last known entry hash. Since forming the hash chain is sequential, we cannot return the hash of the entry you just inserted, so instead we return the last computed hash. You can choose to store that hash somewhere (or at least log it).

The hash needs to be published/stored in a place other than the audit log server, because if an attacker gets hold of the audit log and tries to re-write it, he won't be able to do that without being detected – the hash that has been published/stored elsewhere will not be found, which will indicate tampering with the log. LogSentinel has better ways of guaranteeing that a given hash was indeed computed, thus proving the integrity of the whole audit log, but it will be easier to do periodic checks if you store some of the hashes returned.

2.3 Authentication

In order to use the API, you have to authenticate your calls. For that you need to register , login to the dashboard and obtain the organization key and secret and an application id from the "API credentials" menu. Then you should pass two headers for each request:

- `Authorization: Basic<base64(organizationId:secret)>`
- `Application-Id:<applicationId>`

2.4 Libraries and plugins

Here you can find a *Libraries & Plugins* for various languages and frameworks.

In addition to the libraries, we support agents and plugins for various systems. The most basic integration can be done at the database level, using the LogSentinel database agent .

2.5 Why not . . . ?

The problem of securely storing audit logs is not a strictly defined one. The reasons why certain aspects were not implemented in a particular way are discussed below

- Why not provide built-in anonymization of actors and privacy mechanisms for the data? One of the features of LogSentinel is the ability to easily search, visualize and analyze audited events. Using actor pseudonyms, encrypting the data or using bit masks when computing the trees and hashes (all as suggested in the literature) would yield the search features useless. The premise of most of the papers includes only preserving the integrity of the logs, not analyzing them. Not having the entries fully encrypted would allow machine-learning based risk analysis and alerting on malicious activities. However, the privacy features are not ignored – they can easily be achieved on the client-side, before sending. The RESTful API acknowledges that opportunity and provides an endpoint for that.
- Why not provide binary serialization support, in addition to the HTTP API? LogSentinel uses HTTP2, which is a binary protocol. Adding gzip ontop of that reduces much of the overhead of typical RESTful APIs and allows for high performance
- Why not use merkle tree for everything? The data model chosen wouldn't benefit much from merkle trees over simple hash chains – we need to store all the data for all the entries anyway, and looking up each hash is pretty quick due to the performed indexing. We also don't use merkle proofs, as in the typical use case there is no need for a thin (verification) client. Additionally, representing all the data as one big append-only merkle tree would

introduce complexity in terms of storage and retrieval (we must not assume we will be able to fit the whole tree in memory), We use merkle trees for the timestamping process, as additional integrity guarantee. Currently it doesn't serve as much more than a means to concatenate elements, but timestamp group merkle trees can be useful in the future. Note that even blockchains do not represent the whole chain as a single merkle tree. Instead, each block is represented by a merkle tree, whose roots represent a hash chain.

- Why not use a blockchain? The blockchain has features that are not needed in the "tamper-evident audit log" scenario. The distributed consensus algorithm (with proof or work, for example) and the distributed storage are not necessary in order to achieve the goal of ensuring the integrity of audit logs, therefore using a full-blown blockchain would be an overdesign (even though it would sound cool) and would make setting up and maintaining the system more complex. That's why we use only the relevant bits – the hash chaining / merkle trees.
- Why not use a custom solution or syslog instead of LogSentinel? Custom solutions rarely cover the necessary features and take time and resources to implement. Using syslog or something like splunk or logstash again doesn't cover the security requirements. One can get hash chaining ontop of syslog, as shown by one of the cited papers, but it requires additional development and knowledge on the syslog server internals. LogSentinel is a "drop-in" solution, which is used by a very simple and straightforward API.
- Why not use just timestamping? Timestamping guarantees the integrity of the timestamped groups (blocks) of entries, but does not guarantee that no record was inserted with a date in the past or that no group was deleted. The hash chain provides a strong guarantee that there were no modifications on the entire log

For more details, read the *Advanced Documentation*

CHAPTER

THREE

CODE EXAMPLES

Below are some code example for basic SentinelTrails functionality:

3.1 Inserting a single entry

Java

The Java example uses the logsentinel-java-client

```
//credentials obtained after registration
LogSentinelClientBuilder builder = LogSentinelClientBuilder
    .create(applicationId, organizationId, secret);
LogSentinelClient client = builder.build();

try {
    var result = client.getAuditLogActions().log(
        new ActorData(actorId).setActorDisplayName(username).
↪setActorRoles(roles),
        new ActionData(details).setAction(action)
    );
    System.out.println(result);
} catch (ApiException e) {
    // handle exception
}
```

C#

The C# example uses the logsentinel-dotnet-core-client

```
LogSentinelClientBuilder builder = LogSentinelClientBuilder
        .create(applicationId, organizationId, secret);

builder.setEncryptionKey(encryptionKey); // Optional

LogSentinelClient client = builder.build();

try
{
        var result = client.getAuditLogActions().LogUsingPOST(
                new ActorData().setActorDisplayName(actorName).
↪setActorRoles(actorRoles)
                        .setActorId(actorId),
                new ActionData().setDetails(details).setAction(act)
```

(continues on next page)

(continued from previous page)

```
                        .setEntryType(entryType),
                applicationId);
        Console.WriteLine(result.LogEntryId);
}
catch (ApiException e)
{
        Console.WriteLine("Exception when calling AuditLogControllerApi
↪#logAuthAction");
}
```

PHP

```
$data = <<<EOT
{
  "detail1": "detail 1",
  "detail2": "detail 2"
}
EOT;

$curl = curl_init();
curl_setopt($curl, CURLOPT_POST, 1);
curl_setopt($curl, CURLOPT_POSTFIELDS, $data);

curl_setopt($curl, CURLOPT_URL, 'https://app.logsentinel.com/api/log/' .
↪$actorId . '/' . $action . '/' . $entityType . '/' . $entityId);
curl_setopt($curl, CURLOPT_HTTPHEADER, array(
        'Content-Type: application/json',
        'Application-Id: ' . $applicationId;
));
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($curl, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($curl, CURLOPT_USERPWD, $ORG_ID . ":" . $SECRET);

// EXECUTE:
$result = curl_exec($curl);
```

Python

```
import requests
url = 'https://app.logsentinel.com/api/log/' + actorId + '/' + action + '/'␣
↪+ entityType + '/' + entityId;
data = '''{
  "detail1": "detail 1",
  "detail2": "detail 2"
}'''

response = requests.post(url, auth = HTTPBasicAuth(orgId, secret), data =␣
↪data, headers = {"Content-Type": "application/json", "Application-Id":
↪"applicationId"})
```

Node.js

```
var https = require('https');
var data = JSON.stringify({
  "detail1": "detail 1",
  "detail2": "detail 2"
});
```

(continues on next page)

(continued from previous page)

```
var auth = 'Basic ' + Buffer.from(ORG_ID + ':' + ORG_SECRET).toString('base64')

var options = {
  host: 'app.logsentinel.com',
  path: '/api/log/' + actorId + '/' + action + '/' + entityType + '/' + entityId,
  method: 'POST',
  headers: {
        'Content-Type': 'application/json; charset = utf-8',
        'Application-Id': applicationId,
        'Authorization': auth;
  }
};

var req = https.request(options, function(res) {
  var res = JSON.parse(response.body)
  //...
});

req.write(data);
req.end();
```

3.2 Inserting batch entries

Java

The Java example uses the logsentinel-java-client

```
//credentials obtained after registration
LogSentinelClientBuilder builder = LogSentinelClientBuilder
    .create(applicationId, organizationId, secret);
LogSentinelClient client = builder.build();

List<BatchLogRequestEntry> batch = new ArrayList<>();
for (int i = 0; i < COUNT; i++) {
    String details = "details" + i;

    BatchLogRequestEntry entry = new BatchLogRequestEntry();
    entry.setActionData(new ActionData(details).setAction(action).
↪setBinaryContent(false));
    entry.setActorData(new ActorData(actorId).setActorDisplayName(username).
↪setActorRoles(roles).setDepartment("IT"));
    entry.setAdditionalParams(new HashMap<>());

    batch.add(entry);
}

try {
    client.getAuditLogActions().logBatch(batch);
} catch (ApiException e) {
    // handle exception
}
```

C#

The C# example uses the logsentinel-dotnet-core-client

```
    LogSentinelClientBuilder builder = LogSentinelClientBuilder
            .create(applicationId, organizationId, secret);

    builder.setEncryptionKey(encryptionKey); // Optional

    LogSentinelClient client = builder.build();

    try
    {
            List<BatchLogRequestEntry> batch = new List<BatchLogRequestEntry>();
            for (int i = 0; i < COUNT; i++) {
                string details = "details" + i;

                BatchLogRequestEntry entry = new BatchLogRequestEntry(
new ActorData().setActorDisplayName(actorName).setActorRoles(actorRoles).
↪setActorId(actorId),
                    new ActionData().setDetails(details).setAction(act).
↪setEntryType(entryType));

batch.Add(entry);
            }
            var result = client.getAuditLogActions().LogBatchUsingPOST(batch,␣
↪applicationId);
            Console.WriteLine(result.LogEntryId);
    }
    catch (ApiException e)
    {
            Console.WriteLine("Exception when calling AuditLogControllerApi
↪#logAuthAction");
    }
```

PHP

```
$data = <<<EOT
[{
  "actorData": {
    "actorId":"actor1",
    "actorDisplayName":"actor 1",
    "department":"IT"
  },
  "actionData": {
    "action":"VIEW",
    "entityId":"123",
    "entityType":"Deposit",
    "details":{
          "detail1": "detail 1",
          "detail2": "detail 2"
    }
},{
  "actorData": {
    "actorId":"actor2",
    "actorDisplayName":"actor 2",
    "department":"IT"
  },
  "actionData": {
    "action":"WITHDRAW",
    "entityId":"123",
```

(continues on next page)

(continued from previous page)

```
    "entityType":"Deposit",
    "details":{
          "detail1": "detail 1",
          "detail2": "detail 2"
    }]
EOT;

$curl = curl_init();
curl_setopt($curl, CURLOPT_POST, 1);
curl_setopt($curl, CURLOPT_POSTFIELDS, $data);

curl_setopt($curl, CURLOPT_URL, 'https://app.logsentinel.com/api/log/batch');
curl_setopt($curl, CURLOPT_HTTPHEADER, array(
        'Content-Type: application/json',
        'Application-Id: ' . $applicationId
));

curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($curl, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($curl, CURLOPT_USERPWD, $ORG_ID . ":" . $SECRET);

// EXECUTE:
$result = curl_exec($curl);
```

Python

```
import requests
url = 'https://app.logsentinel.com/api/log/batch';
data = '''[{
  "actorData": {
    "actorId":"actor1",
    "actorDisplayName":"actor 1",
    "department":"IT"
  },
  "actionData": {
    "action":"VIEW",
    "entityId":"123",
    "entityType":"Deposit",
    "details":{
          "detail1": "detail 1",
          "detail2": "detail 2"
    }
},{
  "actorData": {
    "actorId":"actor2",
    "actorDisplayName":"actor 2",
    "department":"IT"
  },
  "actionData": {
    "action":"WITHDRAW",
    "entityId":"123",
    "entityType":"Deposit",
    "details":{
          "detail1": "detail 1",
          "detail2": "detail 2"
    }]'''
```

(continues on next page)

(continued from previous page)

```
response = requests.post(url, auth = HTTPBasicAuth(orgId, secret), data = data,␣
↪headers = {"Content-Type": "application/json", "Application-Id": "applicationId"})
```

Node.js

```
var https = require('https');
var data = JSON.stringify([{
  "actorData": {
    "actorId":"actor1",
    "actorDisplayName":"actor 1",
    "department":"IT"
  },
  "actionData": {
    "action":"VIEW",
    "entityId":"123",
    "entityType":"Deposit",
    "details":{
          "detail1": "detail 1",
          "detail2": "detail 2"
    }
},{
  "actorData": {
    "actorId":"actor2",
    "actorDisplayName":"actor 2",
    "department":"IT"
  },
  "actionData": {
    "action":"WITHDRAW",
    "entityId":"123",
    "entityType":"Deposit",
    "details":{
          "detail1": "detail 1",
          "detail2": "detail 2"
    }]);

var auth = 'Basic ' + (Buffer.from(ORG_ID + ':' + ORG_SECRET).toString('base64'))

var options = {
  host: 'app.logsentinel.com',
  path: '/api/log/batch',
  method: 'POST',
  headers: {
        'Content-Type': 'application/json; charset = utf-8',
        'Application-Id': applicationId,
        'Authorization': auth;
  }
};

var req = https.request(options, function(res) {
  var res = JSON.parse(response.body)
  //...
});

req.write(data);
req.end();
```

3.3 Merkle proofs

For consistency and inclusion proofs, see our example verification application: logsentinel-java-client-verification-ui

CHAPTER

FOUR

ADVANCED DOCUMENTATION

4.1 Documentation

First make sure you've read the getting started guide and the white paper

In this section you'll find a more detailed description of each of the endpoints and the related use-cases

4.2 General use cases

You would normally use LogSentinel as a way to securely store your business-level audit log entries. "Business level" means events that are important to the business logic – all CRUD operations, as well as operations like checking out a basket, completing a transaction, etc. Additionally, you can log authentication events, like registration, login, logout, auto-login and failed login.

That way you'll get a searchable list of business logic events. Regular program logging, where information about the program flow is logged at different levels (error, info, debug), is a separate concern, as it is more useful for identifying problems with the program flow, rather than creating a log of what the users of the system did. The latter needs to have legal strength and protect again malicious internal actors, while the former is useful for engineers. In that sense, LogSentinel is more than a log aggregation tool.

4.3 General client setup

Applications send events to LogSentinel using a RESTful API. That can be done using a native RESTful call mechanism (in dynamic languages it's pretty straightforward) or via one of the client libraries

When configuring the client, a few mandatory parameters must be specified (explained in the Getting started guide):

- `ApplicationId` – obtained from the "API credentials" section in the admin panel and passed with a `Application-Id` header
- `OrganizationId` – obtained from the "API credentials" section and passed, alongside with the Secret parameter, in `Authorization` header
- `Secret` – obtained again from the "API credentials"

You'd also have to make a choice whether you want to send the full details of events, their hashes, or an encrypted version of the details. The latter two would practically disable parts of the search functionality, but will mean that the log server does not have any privacy related details

4.4 Additional headers

- `Signature` - the `Signature` contains a digital signature of the whole request body. The signature is an RSA "signature" using a locally (on your application end) generated key that the LogSentinel server does not have. The premise for LogSentinel is that even if a log server is compromised, modification of the records will be detectable. In the unlikely event of a LogSentinel server being compromised, an attacker could insert fake records, but if they don't possess the private key to sign the records, it will be detectable upon inspection. The algorithm used for signing should be `SHA256withRSA` and the result should be Base64 encoded and set in the `Signature` header. You can configure your public key in the application configuration so that verification is performed automatically.
- `Audit-Log-Entry-Type` – the type of the audit log entry. Allowed values are `BUSINESS_LOGIC_ENTRY`, `DATABASE_QUERY`, `SYSTEM_EVENT`, `NETWORK_EVENT`, `DOCUMENT`. The header is optional and the default value is `BUSINESS_LOGIC_ENTRY` as this is the default use case for LogSentinel. However, database monitor agents and other system or network logging solutions can be attached to LogSentinel as well, and this header allows for that.

4.5 Additional parameters

You can specify any number of additional query parameters (after ? in the URL) for all endpoints below. There are are a few standard ones

- `actorRoles`, `actorDisplayName` – with them you can assign a search-friendly and dashboard-friendly roles of the actor as well as a human-readable alias (in addition to the ID passed in the path)
- `actorDepartment` - the department where the actor belongs
- `gdprCorrelationKey` - a key to correlate the entry with a certain GDPR process from the Article 30 register we provide
- `process` - the name of the business process from which the event originates
- `directExternalPush` - you can designate certain events to be directly pushed to either Ethereum, a qualified electronic timestamp provider, email or twitter (respective values being `ETHEREUM`, `QTSA`, `EMAIL`, `TWITTER`). You can find more details in the our *On-premise security* page.
- `encryptedKeywords` – with it you can enable search in encrypted payload. See more details in the next section

You can use your custom parameters to perform structured searches by prefixing `additionalParams.`. For example you can pass

4.6 Response

All of the methods below return a JSON or XML response (depending on the supplied Accept header) which contains two fields:

- `logEntryId` – the internal ID of the inserted log record
- `lastKnownHash` - the last known computed hash in the hash chain. Note that this is not the hash corresponding to the current item, as the hashes are computed asynchronously. You can store this value and call the `/log/verify` endpoint, but this is not necessary, as automatic verification is performed by the LogSentinel service on regular intervals

4.7 /api/log/simple

The simple logging endpoint requires no specific parameters to be passed (apart from the authorization headers). You are free to pass anything in the body of the POST request, including encrypted or hashed versions of the event details.

4.8 /api/log/{actorId}/{action}/{entityType}/{entityId}

The full logging endpoint passing the following parameters as part of the path:

- `actorId` – the ID of the actor who performed the action leading to this event. Normally this is a userId, but in some cases it can be a system process name or plugin name (e.g. in the case of WordPress), in case the action is performed in the background
- `action` - you can use any action name that makes sense for your application. The regular can be INSERT/UPDATE/DELETE/GET, but there is no limitation.
- `entityType` - the type of the entity that is modified. If there is no entity, use the endpoint below. Usually that would correspond to a database table name or an ORM-mapped class name
- `entityId` - the ID of the entity that this event is about

4.9 /api/log/{actorId}/{action}

Same as the above endpoint, but used in case there is no particular entity (for example a user kicks-off a background process, or performs a search)

4.10 /api/log/document/{actorId}/{documentAction}/{documentId}

Useful when working with documents, rather than audit log events. Each time a document is created, modified or deleted, this can be logged

- `documentAction` - `CREATE_DOCUMENT`, `UPDATE_DOCUMENT`, `DELETE_DOCUMENT`, `RETRIEVE_DOCUMENT`,
- `documentId` - can be the document name or another identifier.
- Document type can be specified via a query param (e.g. `?documentType=PDF`)

When documents are logged, you can perform regular verifications on the integrity of your documents – do a search for particular document names and check if the hashes that you've originally passed match the ones stored at LogSentinel.

4.11 /api/log/{actorId}/auth/{action}

This endpoint is about authentication-related actions. The allowed values for the `action` parameter are: `LOGIN`, `LOGOUT`, `SIGNUP`, `AUTO_LOGIN`, `LOGIN_AS`, `LOGIN_FAILED`

`LOGIN_AS` is used when a staff member logs in on behalf of a user and the `AUTO_LOGIN` can be used to distinguish regular login from remember-me functionality.

This endpoint allows two additional optional headers – `Signed-Login-Challenge` and.. `User-Public-Key`. In case your users are authenticating using a private key (or a password-derived private key, e.g. using WebCrypto API), you can have them sign a login challenge with their private key and provide the signature and the public key. The login challenge can be the login event details, or a custom challenge that you can pass as an additional parameter. That way their authentication bears more legal strength, as they cannot deny having logged in (the signature has the non-repudiation property).

4.12 /api/log/batch

This method is used for batch inserts. It is generally recommended to insert events as soon as they occur, to avoid any intermediate tampering on the client side. But in some cases it makes sense to group requests (e.g. an agent that listens to the database audit log / query log – making a request for each query might mean excessive number of requests) The method accepts only a request body in the following format (all the fields are optional, but you should specify at least one for the entry to make sense):

```
[
  {
    "actionData": {
      "action": "ACTION",
      "details": "",
      "entityId": "123",
      "entityType": "BASKET",
      "entryType": "BUSINESS_LOGIC_ENTRY"
    },
    "actorData": {
      "actorDisplayName": "John Doe",
      "actorId": "123",
      "actorRoles": [
        "manager"
      ]
    },
    "additionalParams": {}
  }
]
```

Note that if you want to provide a signature, you have to provide it in `additionalParams` with a field `signature` per entry, rather than one signature for the whole request.

4.13 /api/search

With that endpoint you can perform programmatic search on your stored events. The parameters are:

- `query` – the Lucene query to perform against the search engine. You can read more about the query syntax here.
- `startTime` - epoch millis of the start of the period you want to limit your search to
- `endTime` - epoch millis of the end of the period you want to limit your search to
- `page` - the page number for the search results
- `pageSize` - the page size for the search results

The response is a list of audit log entry details:

```
[
   {
      "id":"89b71f20-512c-11e7-815e-c3cda8182be4",
      "timestamp" :1497463738898,
      "actorId":"1",
      "actorRoles":[
         "administrator",
      ],
      "action":"UPDATE",
      "entityId":"195",
      "entityType":"Post",
      "details":{
         "PostID":195,
         "PostType":"post",
         "PostTitle":"...",
         "CurrentUserID": 1
      },
      "applicationId":"07d7ed50-5040-11e7-863a-6bd5280da4f2",
      "ipAddress":"172.31.15.212",
      "actorDisplayName":"admin",
      "previousEntryId":"3f36b0e0-5128-11e7-815e-c3cda8182be4",
      "hash":"cvpyp98p7pjg8GZjXpQ-kpFH8hqnUq9IGArzrUBhk_KsgOy2-9ZZSvr-
↪g4bJOWiXeqsbFvQCNRXqHNMoWK6x7g==",
      "timestampGroupSize": 1
   }
]
```

4.14 /api/verify?hash={hash}

An endpoint for manual verification whether the supplied hash is present in the hash chain. A missing hash would indicate tampering. Note that it is not necessary to use this endpoint, ad automatic log verification is performed by LogSentinel on regular time intervals.

4.15 Hashable content endpoints

There are endpoints that are equivalent to the above (in terms of path variables, headers and parameters), but instead of `/log` begin with `/getHashableContent`.

These endpoints return the content that is hashed given a particular logging request. This introduces transparency, as you can manually apply the SHA-512 hash function to the returned hashable content and compare it with the hash that LogSentinel computes for each event.

4.16 curl example

Below is a `curl` example to get you started with the API

```
curl -X POST -u $ORGID:$SECRET --header 'Content-Type: application/json' \
    --header 'Accept: application/json' --header 'Application-Id: 123e4567-e89b-12d3-
↪a456-426655440000' \
    -d '{"details": 1}''https://api.logsentinel.com/api/log/actor-1/ACTION'
```

For more experiments, obtain API credentials and experiment on our API page

CHAPTER

FIVE

API REFERENCE

The full API reference is available here

CHAPTER

SIX

LOG VERIFICATION

Even though our technology is heavily influenced by blockchain, verification of the chain is not a given. Even in popular blockchains, it is done only in certain scenarios and only partially. Public blockchains do not perform full chain verification by default because that is not their core usecase. It is ours, however, and we need to make sure that the logs have not been tampered with. That's why there are multiple ways to verify that the chain of logs is intact.

6.1 Automatic background verification

We perform regular verification of the log, going from the newest to the oldest log entry and matching their hashes, their timestamps, optionally their signatures as well as the hashes of the blocks they belong to. Full verification is time-consuming, so it can be turned off for very large logs (hundreds of millions of retained entries). The period for full verification is configurable through the dashboard and usually goes from 12 to 36 hours depending on the subscription plan and perceived risks.

There's also the partial verification which is run every 15-30 minutes and it verifies only the latest inserted logs.

In case of failure in any of these automated verification processes, a configured list of recipients is notified via email or *via Telegram*.

6.2 Merkle tree proofs

Merkle trees are at the core of many blockchain implementations. We support multiple Merkle Tree endpoints that allow performing consistency and inclusion proofs. What those proofs mean and how they work is explained here and we recommend reading that section if you want to perform independent verification of a SentinelTrails chain. In short:

- A consistency proof guarantees that the chain has not been tampered with between two publicly known root hashes. Normally you obtain the two roots (e.g. from those pushed to Ethereum or a third party service), you get the consistency proof and validate it.
- An inclusion proof guarantees that a particular entry is in the chain. You pass the expected standalone hash of the entry (as per the Hashable content endpoints result) and then verify whether the proof is valid.

You can also read our paper on our use of Merkle trees.

Note: We have built a sample verification application that uses merkle proofs and hashes, published to Ethereum in order to verify the chain. You can look through its code here

6.3 Incremental verification

When you know a certain set of hashes, e.g. those published to Ethereum, pushed to Twitter, sent via Email, or stored in Glacier, you can request all entires that lie between two known hashes in the chain by using the /api/verification/entries endpoint.

Note: Because we form hashes by concatenating the log entry data in specific order, you can use the hashable content endpoints to obtain the content for a given entry would look like and then apply SHA-512 on it to obtain the hash.

When you have all entries between the two known hashes, you can make sure that the chain is properly computed or whether some log entry has been tampered with.

6.4 Manual verification

We offer a simple UI to verify if a given hash is present in the chain. It does verify its basic integrity properties but it doesn't guarantee that the whole chain is intact, only the particular entry.

CHAPTER

SEVEN

SEARCHABLE ENCRYPTION

LogSentinel's API provides mechanism of searching in encrypted details. To be able to do this, you must perform the following 3 steps before sending data to the API:

1. Extract parameters and keywords from the payload, by which events will be searchable
2. Encrypt payload details with symmetric key. The algorithm must be AES (128 or 256). Important: for better security you should put a random block of 16 bytes in front of plain message before encrypting.
3. Encrypt each keyword with the same key as in step 2 and hash it with SHA-256

Make a request to any of the `/api/log/` endpoints with the encrypted payload and parameters and add additional request param `encryptedKeywords` with value=comma separated list of all encrypted and hashed keywords.

On the LogSentinel application dashboard page you can provide your key (Base64 encoded), thus enabling both decrypting encrypted details and searching by keywords in it. Note that **the key is not sent to the server**, so your encrypted details remain secret.

CHAPTER

EIGHT

LIBRARIES & PLUGINS

8.1 Client libraries

Ready to use client libraries can be found on GitHub.com/LogSentinel Currently the following are available:

- Java – logsentinel-java-client
- PHP – logsentinel-php-client
- .NET core - logsentinel-dotnet-core-client

8.2 Plugins

The following plugins and agents are also available:

- Zapier integration - you can integrate 1000+ applications with LogSentinel through Zapier
- WordPress plugin (source) – the plugin relies on an existing audit log plugin
- LogSentinel Agent – command-line tool for monitoring files and databases and sending the changes to be stored in LogSentniel. *More details can be found here*

8.3 Community contributions

Client libraries and other tools contributed by the community

- PowerShell integration by Snagler

CHAPTER

NINE

LOG COLLECTOR INTEGRATION

9.1 Integration with Fluentd

Fluentd quickstart

- install third party plugin http (requires basic knowledge of ruby gems). Info for fluentd custom plugins example configuration for the plugin to communicate with logsentinel

```
<source>
  @type tail
  path /opt/log.txt
  refresh_interval 10
  tag logsentinel.file
  <parse>
    @type regexp
    expression /(?<actorId>[^ ]*) (?<action>[^ ]*) (?<entityType>[^ ]*) (?<entityId>[^
↪ ]*) (?<param1>[^ ]*)$/
  </parse>
</source>

<match logsentinel.**>
  @type http
  endpoint_url     https://api.logsentinel.com/api/log/<actorId>/<action>/<entityType>
↪/<entityId>?param1=<param1>
  serializer    json
  custom_headers {"Application-Id": "b1fgt7a0-5rc5-11e8-8230-0db3d3bfb10d"}
  username <organizationId>
  password <secret>
</match>
```

- `<source>` configuration is only for testing purposes. It shows how to use regex to format data properly It gets lines from log file with path <path> every <refresh_interval> seconds and parses it with <expression> regex, so data can be extracted easy. This specific regex transforms: `"actor1 action2 entityType3 entityId4 urlParam" -> {"actorId":"actor1","action":"action2", "entityType":"entityType3","entityId":"entityId4","param1":"urlParam"}`
- `<match>` config is with type http which is the plugin that is already installed.
- `endpoint_url` is Logsentinel API url. Path variables and url params can be extracted from input (properly parsed). Params in <> are replaced with their values. Nested params also can be used (example: `<data.id>` extracts `444` from `{"data" :{"id":444}}`)
- `custom_headers`, `username` and `password` contain mandatory headers for authentication and authorization. Values of Application-Id, username and password should be obtained from the API credentials page on your dashboard

Additional configuration params are available - see http plugin configuration options

9.2 Integration with Logstash

- Logstash http plugin documentation: https://www.elastic.co/guide/en/logstash/current/plugins-outputs-http.html
- sample configuration for integration with logsentinel

logstash.conf

```
input {
    file {
        path => "/opt/log.txt"
        start_position => "beginning"
    }
}
filter {
    grok {
        match => { "message" => "actorId=%{WORD:actorId} action=%{WORD:action}␣
↪entityType=%{WORD:entityType} entityId=%{WORD:entityId}" }
    }
}
output{

   http {
   format=>"json"
   http_method=>"post"
   url=>"https://api.logsentinel.com/api/log/%{[actorId]}/%{[action]}/%{[entityType]}/
↪%{[entityId]}"
   headers => ["Application-Id", "b1f8b7a0-5cc6-11e8-8230-0dr3d3brb12d"]
   headers => ["Authorization",
↪"BasicYjFmNjQ2YTAtNWNjNS0xMWU4LTgyMrEtMGRiM1QzYmDiMTBkOmM0YjA4OWViMDg1MmJmNmI0ZGJhNjMwMTJmN2Y2Y2RjN
↪"]

    }
}
```

grok filter parses mandatory fields from a sample log file in key=value format. This is just an example, you can use any logstash functionality you wish.

Authorization and Application-Id headers contain mandatory headers for authentication and authorization. Values of Application-Id and Authorization are just an example. Your organization real values must be provided. Authorization header consists of "Basic" string + base64_encode(<your organization id>:<your secret>)

9.3 Integration with Nxlog

- Nxlog http module documentation https://nxlog.co/documentation/nxlog-user-guide#om_http
- sample configuration for integration with logsentinel

nxlog.conf

```
<Input file>
    Module              im_file
```

(continues on next page)

(continued from previous page)

```
    File                '/opt/log.txt'
</Input>

<Output http>
    Module              om_http
    URL                 https://api.logsentinel.com
    ContentType application/json
        AddHeader   Authorization :␣
↪BasicYjFmNjQ2YTAtNWNuNS0xMeU4LTgyMzAtMGRiM1QzYmZiMTBkOmM0YjA4OWViNDg1MmJ
        mNmI0ZGJhNjMwMTJmN2Y2Y2RjMjk3ZWY3ODg4NmRiM2E5YjViODhiNGUxZGZlMzZhOGM=
        AddHeader   Application-Id : b1f8b7a0-5cc5-11e8-8230-0db3d3bfb10d
    <Exec>
        $raw_event =~ /(\S+) (\S+) (\S+) (\S+)/ ;
        $actorId = $1;
        $action = $2;
        $entityType = $3;
        $entityId = $4;
        set_http_request_path('/api/log/'+ $actorId + '/' + $action +'/' +
↪$entityType +'/' +$entityId);
    </Exec>
</Output>
```

URL is Logsentinel API url (api.logsentinel.com)

Authorization and Application-Id headers contain mandatory headers for authentication and authorization. Values of Application-Id and Authorization are just an example. Your organization real values must be provided. Authorization header consists of “Basic” string + base64_encode(<your organization id>:<your secret>)

Extracting data from logs here is just simple regex that reads 4 words from log file and fills the mandatory url params (actorId, action , entityType, entityId). You can use all Nxlog functionality to parse and transform your logs as you wish.

Note: Sending custom http headers is only available in Enterprise edition of Nxlog. This feature is mandatory for integration with Logsentinel.

CHAPTER

TEN

SYSLOG INTEGRATION

In order to integrate with syslog, there are several options.

- We support syslog over TCP (plaintext and over TLS) as well as over UDP.
- We support both RFC 3164 and RFC 5424. In addition to that, we support SonicWall extended syslog messages

Below is a list of endpoints and ports for each supported variant. We recommend TCP over TLS for most installations. However, some setups lack the needed flexibility, so fallback to plaintext TCP or UDP may be needed. In such cases there's an option for a VPN tunnel (for enterprise customers), or a more complicated internal setup with a intermediate syslog forwarder.

- syslog.logsentinel.com:514 - plaintext TCP
- syslog.logsentinel.com:515 - TCP over TLS
- syslogudp.logsentinel.com:1516 - plaintext UDP

We have a syslog configuration script which you can download and run configure-syslog.sh. It configures a syslog template that allows authenticating against LogSentinel Trails. The important line for authentication is this:

```
\$template LogSentinelFormat,\"<%pri%>%protocol-version% %timestamp:::date-rfc3339%
↪%HOSTNAME% %app-name% %procid% %msgid% [logsentinel@$LOGSENTINEL_DISTRIBUTION_ID␣
↪organizationId=\\\"$LOGSENTINEL_ORG_ID\\\" secret=\\\"$LOGSENTINEL_ORG_SECRET\\\"␣
↪applicationId=\\\"$LOGSENTINEL_APP_ID\\\" tag=\\\"RsyslogTLS\\\"] %msg%\n\"
```

SonicWall and other devices can be authenticated using a syslogId that is configured per device. You can obtain the syslogId (which effectively comprises of an ID and secret) from the API credentials page.

CHAPTER

ELEVEN

GDPR FUNCTIONALITY

The General Data Protection Regulation requires systems to be upgraded to follow certain rules. Some of the requirements can't be handled by a drop-in solution, but some can. That's why LogSentinel supports a number of features – a GDPR register and GDPR-specific logging endpoints.

The GDPR endpoints are under `/api/log-gdpr/`. There you can log consent and all requests by data subjects in a way that you can prove to regulators the events that happened. More details can be found in the API console .

We support a GDPR Article 30 register, where a company should enlist all its processing activities (what types of data about what types of data subjects it processes). What does this have to do with audit logs? Since the regulation requires processing of data to be authorized, and the integrity of the data to be guaranteed, audit logs can be mapped to a particular processing activity in the register. That for each processing activity you'll be able to track the relevant actions. This is done simply by providing an extra GET parameter to the logging call – `gdprCorrelationKey`. Each processing activity can be assigned a unique correlation key to make it match the audit log records.

Additionally, you can use the GDPR register (via the `/gdpr` API endpoints) to fetch information about processing activities in order to display it to users for the purpose of collecting their consent. It's best to have the register and your website in sync, so that all consent-dependent activities are covered and the user explicitly agrees to each of them.

CHAPTER

TWELVE

COMPARING DATABASE STATE

In some cases you may want to verify that the state of your database is not tampered with by comparing it to the logs. In order to do that, you have to be able to match log entries to database records. You can do that in several ways:

- when sending an event, store its ID in your database in a designated column
- pass `entityId` for each request. The `entityId` is the ID of your record

Note that if you perform updates, multiple log entries will be linked to the same record. Note: you can fetch all entries regarding a single entity by using `/api/search/entityHistory`

When you want to perform verification, you can fetch records in bulk using the `/api/search/batch` endpoint by passing your IDs as `values`.

If you pass log entry IDs (the one LogSentinel returned), you should specify `field=ID`. If you want to pass `entityId`, then pass.. `field=ENTITY_ID` and `entityType=YourEntityType`.

You can pass up to 2000 IDs and get the resulting entities. Then you can compare the returned entities to the contents of your database. Again note that when searching by entityId you may get multiple log entries.

Since comparison may be expensive, you should not perform it excessively. Comparing your most recent data once every few days and your entire database every month should be sufficient in most cases.

In case of large amounts of logically immutable data, additional aggregation steps may be taken. After an initial verification using the above approach, a large chunk of records can be hashed together to form a single hash (of e.g. 100 000 records). Then that hash can be pushed to LogSentinel and stored in a separate column next to the other data of each record. Then, a scheduled job (e.g. once a month) could go and recalculate the hash for these records and search LogSentinel for that hash. That way if an attacker has tampered with the database, 1. the hash won't match even internally, or 2. the hash won't be found in LogSentinel if the attacker has also recalculated the hash. That way periodic calls over-the-wire to LogSentinel won't be as heavy, as only a limited amount of data will be transferred.

CHAPTER

THIRTEEN

ADVANCED FAQ

Q: What if someone deletes data in my system, how does LogSentinel help?

A: There are several complementary measures that can be taken. First, there's the entity history API endpoint which can be used to reconstruct the current state of an entity and compare it to what is stored in a database. Additionally, you can write your backups to WORM (write-only) storage so that nobody can tamper with the backups and then once you discover tampering has happened, you can restore the particular backup and restore the data.

Q: What if a system administrator is malicious?

A: Malicious administrators on our end cannot tamper with the log without being detected. If there's a malicious administrator on your end (or if LogSentinel is run on-premise), the only thing they can do is insert fake events and block access to the LogSentinel installation on a network level. They cannot do Man-in-the-middle attacks (communication is encrypted). A best practice would be to also log admin actions, e.g. commands they execute on systems, so that the activity of blocking network traffic or inserting fake events manually is visible. An additional good practice is to sign all events with a private key (potentially stored in a hardware security module) which can later be verified using the corresponding public key.

Q: Can I prove to 3rd parties that a log exists and is not tampered with?

A: Yes, there's an option to give limited auditor access in case of audits. There's also a user interface to verify individual hashes that you can give to your customers

Q: Should I store personal data in the logs if I have to follow GDPR (The EU General data protection regulation)?

A: It is discouraged, as deleting data from the log is not desirable (the whole purpose of the log is to make it impossible to delete entries). However, depending on the legal analysis, limited personal data can be stored, pursuant to Article 17(3)(e) of GDPR if it would be used "for the establishment, exercise or defence of legal claims."

Q: What if we don't want to risk sending sensitive data to a cloud solution?

A: LogSentinel supports *searchable encryption* – you encrypt all the data before sending it but you can still search in it through our dashboard without us knowing its contents.

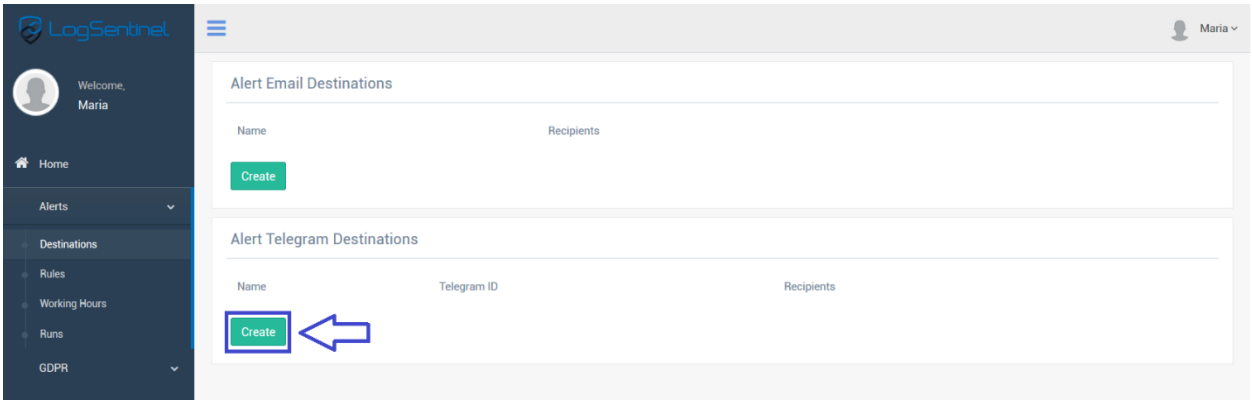
CHAPTER

FOURTEEN

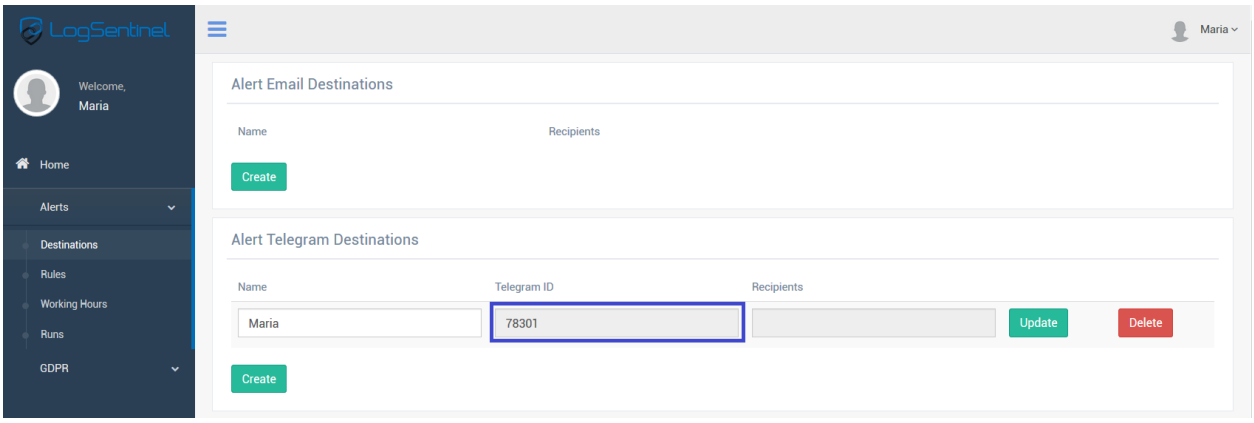
TELEGRAM MESSENGER NOTIFICATIONS

If you would like to set up alerts and get notified via phone for certain types of anomaly activities, follow the below instructions.

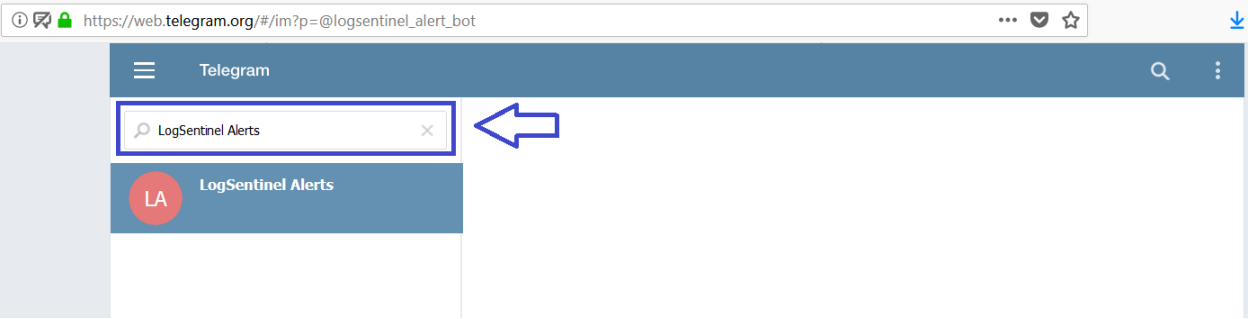
Log into your Sentinel Trails dashboard. Navigate to Alerts and click on *Destination* . In the section *Alert Telegram Destinations* click on the button *Create*.



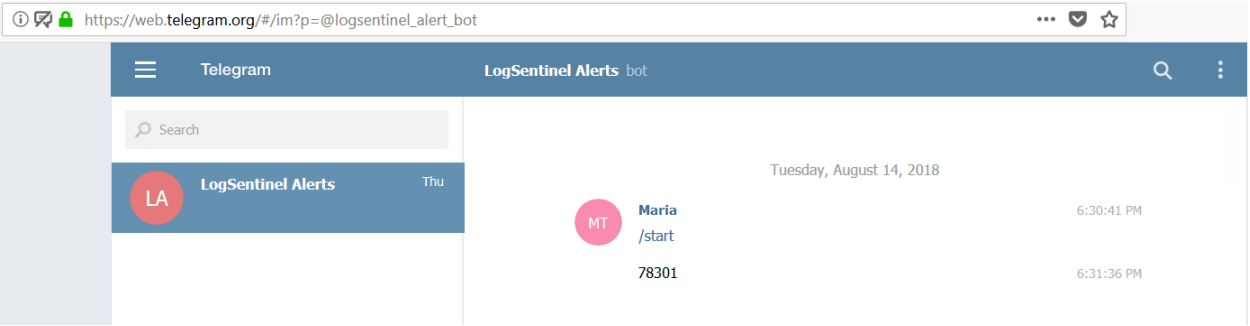
Type in your name and copy the *Telegram ID* number.



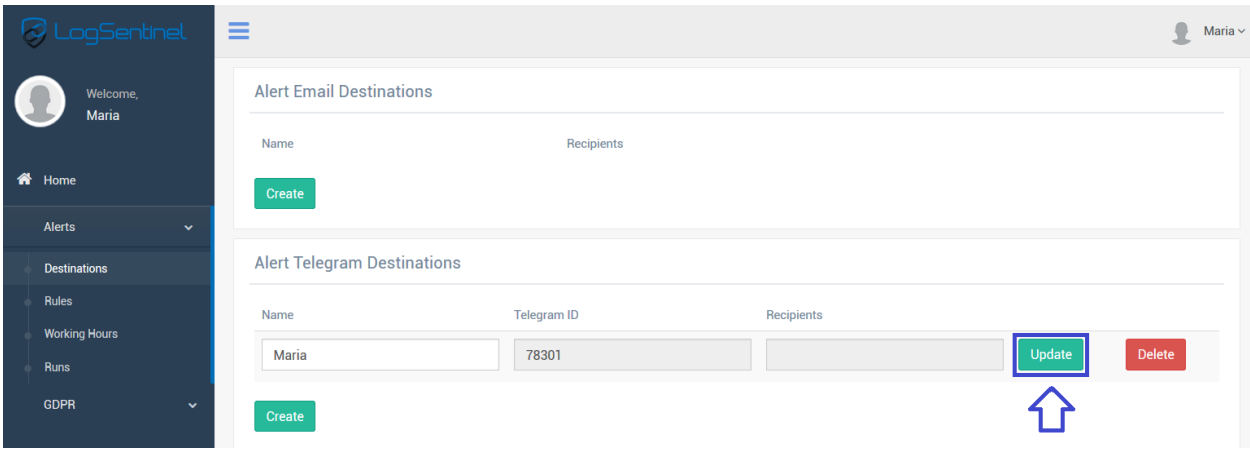
Open the Telegram Messenger (If you don't have it installed, download it from https://telegram.org /) and find **LogSentinel Alerts** .



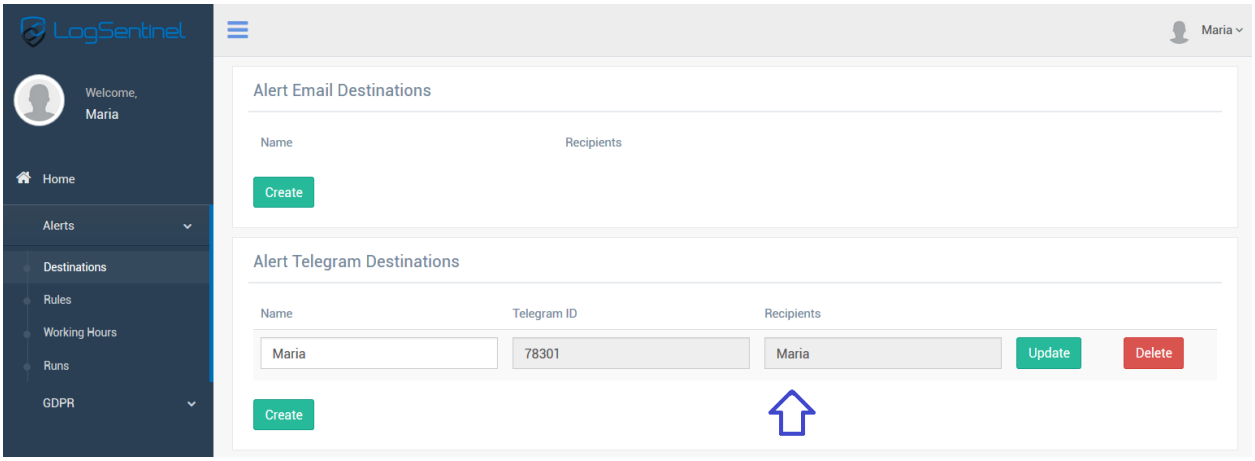
Send the *Telegram ID* to the LogSentinel Alerts bot.



In the LogSentinel control panel, update the *Alert Telegram Destinations* and your name will appear under *Recipients*



Your account is now ready to receive alerts! Keep an eye on them and make a few tests to ensure that everything is set up correctly.


LogSentinel
Maria
Welcome,
Maria
Home
Alerts
Destinations
Rules
Working Hours
Runs
GDPR
Alert Email Destinations
Name
Recipients
Create
Alert Telegram Destinations
Name
Telegram ID
Recipients
Maria
78301
Maria
Update
Delete
Create

CHAPTER

FIFTEEN

LOGSENTINEL AGENT OVERVIEW

The Sentinel Trails listening agent is an open source component that gets installed on target machines to listen to a configured set of log sources. It can be installed on Linux and Windows and supports the following types of sources:

- **Log files** an arbitrary text file can be collected and sent, line by line, to the Sentinel Trails service. These typically application logs and logs by any other non-standard software
- **Database logs files** - if database query logs are enabled, the agent listens to newly issued queries and sends them to the Sentinel Trails service
- **Database tables** - if you store audit trail inside relational database tables, you can configure queries that periodically fetch new entries and send them to the Sentinel Trails service
- **MS SQL audit trail** – if MS SQL audit trail is enabled, the agent can be configured to listen to it and forward the events
- **MS SQL change tracking** – if MS SQL change tracking is enabled, the agent can be configured to listen to it and forward the change events
- **Access logs** – the standard web server access log files can be parsed and sent to Sentinel Trails
- **Linux audit log** – the native linux audit log file can be tailed and forwarded to Sentinel Trails
- **Windows event logs** – the Windows event logs (including all categories – Application, Security, System) can be read continuously as sent to Sentinel Trails
- **Directory changes** - any changes in a directory (new files, removal of files, modification of files) can be tracked using this agent configuration.
- **AxonDB logs** – AxonDB is a special type of non-relational database. We support its custom log format.

Any combination of the above can be configured. Note that for some target types the agent can be installed on a different machine than the actual log source. For example, in case of database tables or database audit trail, the agent can be installed on another machine that connects to the database server via a database connection string and credentials.

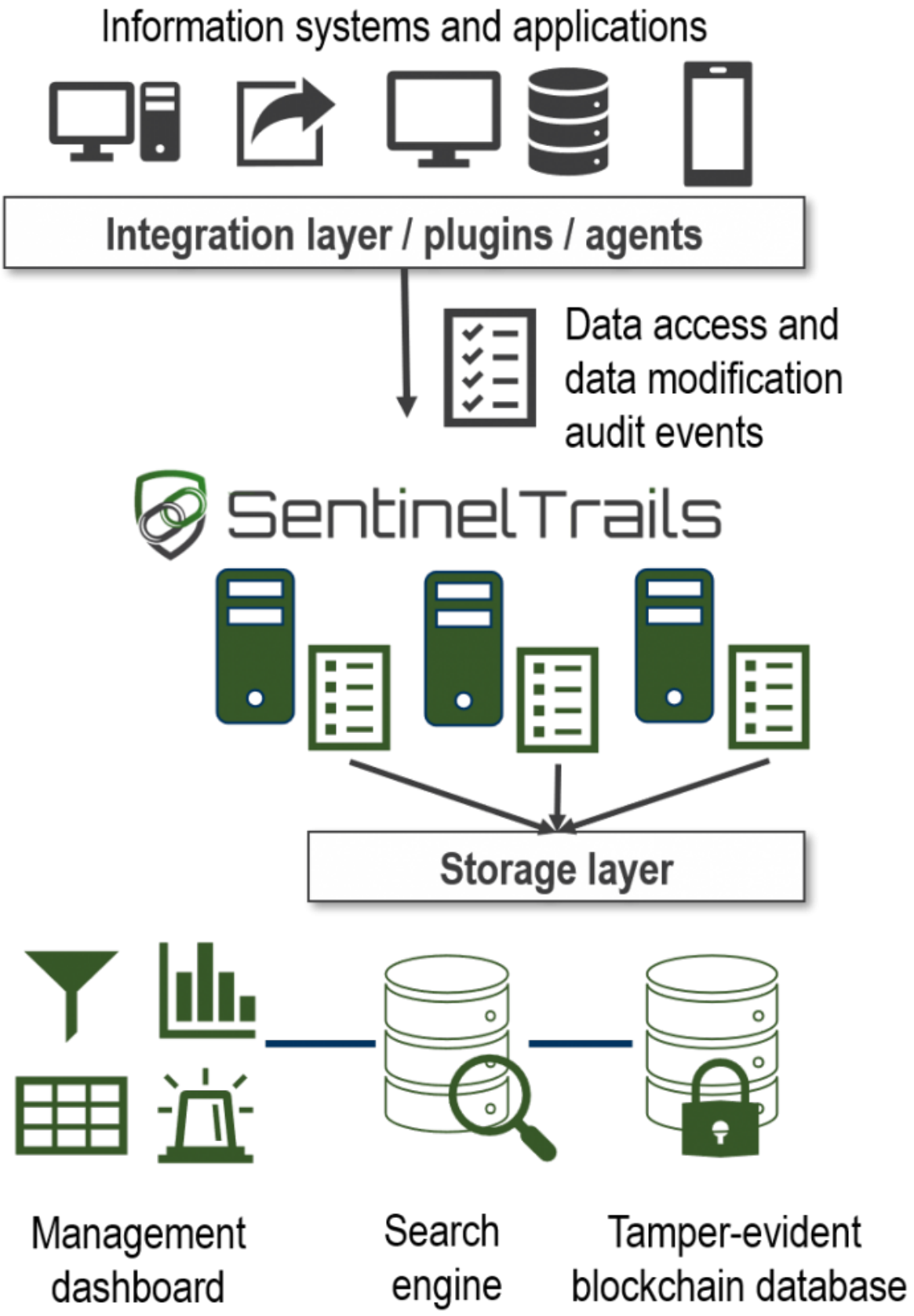
Full configuration details can be seen here .

All communication between the agent and the Sentinel Trails service is **encrypted** .

Configuration and installation is done through scripts provided by us and you *can follow the steps here*:

In order to find existing logs and tables to monitor and send to SentinelTrails, you can use our free and open source scan-logs script

Below is an overview of how the agent fits into the architecture:


Information systems and applications
Integration layer / plugins / agents
Data access and
data modification
audit events
SentinelTrails
Storage layer
Management
dashboard
Search
engine
Tamper-evident
blockchain database

CHAPTER

SIXTEEN

CONFIGURING THE LOGSENTINEL AGENT

When audit logs are concerned, there are many ways to generate and collect them. Ideally, audit logs are generated in code, depending on the business logic of each application and sent for secure storage to SentinelTrails .

However, refactoring a system to include dedicated audit log functionality may not be feasible, as there are a lot of legacy systems out there. This is why we have built the LogSentinel Agent , an open-source tool that can be installed on any machine in order to collect logs that are relevant for forensic, audit and compliance purposes.

16.1 Supported log sources

The agent supports many types of log sources:

- Log file – reading a simple text log file and sending each line as a separate log event. You can configure a file to be watched and "tailed" and log records will be sent to SentinelTrails.
- Relational database – some applications already have some sort of audit trail stored in the database and only need to forward it to a more secure and tamper-protected service. You can configure multiple queries and map columns to particular audit log entry fields (e.g. actor, action, entityType). Queries rely on a datetime column to only fetch fresh records.
- Database log – a smarter log file extension – it looks for SQL queries in a database log file, parses them and sends each query as a separate log message
- Access log – web servers usually generate access logs and they may be seen as audit logs in some circumstances. The agent supports the standard way (supported by Apache, Nginx, etc.) to provide access log format.
- MS SQL Server audit log – SQL Server has a special audit log functionality. It is enabled
- as described here
- and then it can be queried by the agent in order to obtain and send each audit log entry to the SentienlTrails service
- Linux audit log – Linux distributions have a standard audit log file (/var/log/audit/audit.log). The agent supports the specific format of its contents, extracts the relevant fields and sends the events
- Directory – the agent can listen to directory changes (e.g. adding or removing files)
- Windows event log – windows applications normally use the windows event log and so the agent can tap into that log and forward its contents. the agent is customizable to include or exclude certain sources
- AxonDB – AxonDB is a special event-based database. Since every modification there is an event, the Agent can listen to all these events and convert them to audit log entries to be sent to SentinelTrails

16.2 Installing the agent

Installing the agent is simple. It depends on whether you install it on a Linux or Windows machine, but it usually involves a short script or a one-click installer.

16.2.1 Installing on Linux

1. Get the latest release of logsentinel-agent.jar and copy it to `/var/logsentinel/logsentinel-agent.jar`
2. Get the logsentinel-agent.conf file and copy it to ‘‘/var/logsentinel/logsentinel-agent.conf
3. Add a `logsentinel-agent.yaml` in the same directory. The file should contain the configuration of the agent (see the next section)
4. Get the setup-agent.sh script and run it (works on CentOS/RHEL; we'll soon add a similar script for Debian-based distros)

This should start the agent and configure it to run automatically on startup. You can start and stop it via `service logsentinel-agent start/stop`

16.2.2 Installing on Windows

1. Get the latest Windows installer
2. Extract it and run the `install.bat` (you need admin privileges)
3. Customize the `logsentinel-agent.yaml` file in the installation directory
4. Go to Services and start the LogSentinelAgent service

Note: if you are going to collect Windows event logs from other machines, you need a series of permissions configurations described in detail here

16.3 Configuring the agent

Configuring the agent is done via a straightforward YAML file. All properties are described in the documentation . Below is a sample setup that listens to a Windows log as well as a MS SQL Audit trail:

```
applicationId: ba2f0780-5424-11e8-b88d-6a2c1b6625c8
organizationId: ba2cdc90-5424-11e8-b88d-6a2c1b6625c8
secret: d8b63c3d82a6ded56b015a3b8617bf376b6aa6c181021abd0d37e5c5ac9941a1

# BUSINESS_LOGIC_ENTRY, DATABASE_QUERY, SYSTEM_EVENT
entryType: BUSINESS_LOGIC_ENTRY

# FILE, RELATIONAL_DATABASE, DATABASE_LOG, DIRECTORY, ACCESS_LOG, MSSQL_AUDIT_LOG,␣
↪LINUX_AUDIT_LOG, AXON_DB, WINDOWS_EVENT_LOG
targetTypes:
  - MSSQL_AUDIT_LOG
  - WINDOWS_EVENT_LOG

logsentinelBaseUrl: https://api.logsentinel.com

includeMacAddress: false
```

(continues on next page)

(continued from previous page)

```
includeLocalIp: false
timestampInitialUseCurrent: true

windowsEventLogAgent:
        sendLogsRate: 30000
        sourceTypes:
                - Application
                - Security

mssqlAuditLogAgent:
        jdbcConnectionString: jdbc:sqlserver://localhost:1434;integratedSecurity=true
        sendLogsRate: 30000
        mssqlLogsPath: c:\logs\mssqltrail\
```

16.4 Conclusion

The logsentinel-agent can be installed on any machine and will forward any of the supported log records to SentinelTrails. This allows for integrating SentinelTrails into any kind of organization, regardless of whether it relies on legacy systems or is building new ones. The agent can also work alongside existing log collection tools , so that you forward the most business critical events for secure storage and leave the rest of the logs in the existing, less secure solution.

Flexibility and integration-friendliness are key elements of an information security solution and we are happy to offer such a tool, bundled with support for our enterprise customers.

CHAPTER

SEVENTEEN

AGENT PROPERTIES

The full reference of agent properties can be found here.

CHAPTER

EIGHTEEN

ON-PREMISE INSTALLATION ON LINUX

To be completed. . .

CHAPTER

NINETEEN

ON-PREMISE INSTALLATION ON WINDOWS

SentinelTrails can be installed on-premise for enterprise customers. It can be done as part of a contract by our staff, or internally by the staff of your company. Below are the instructions for the latter scenario.

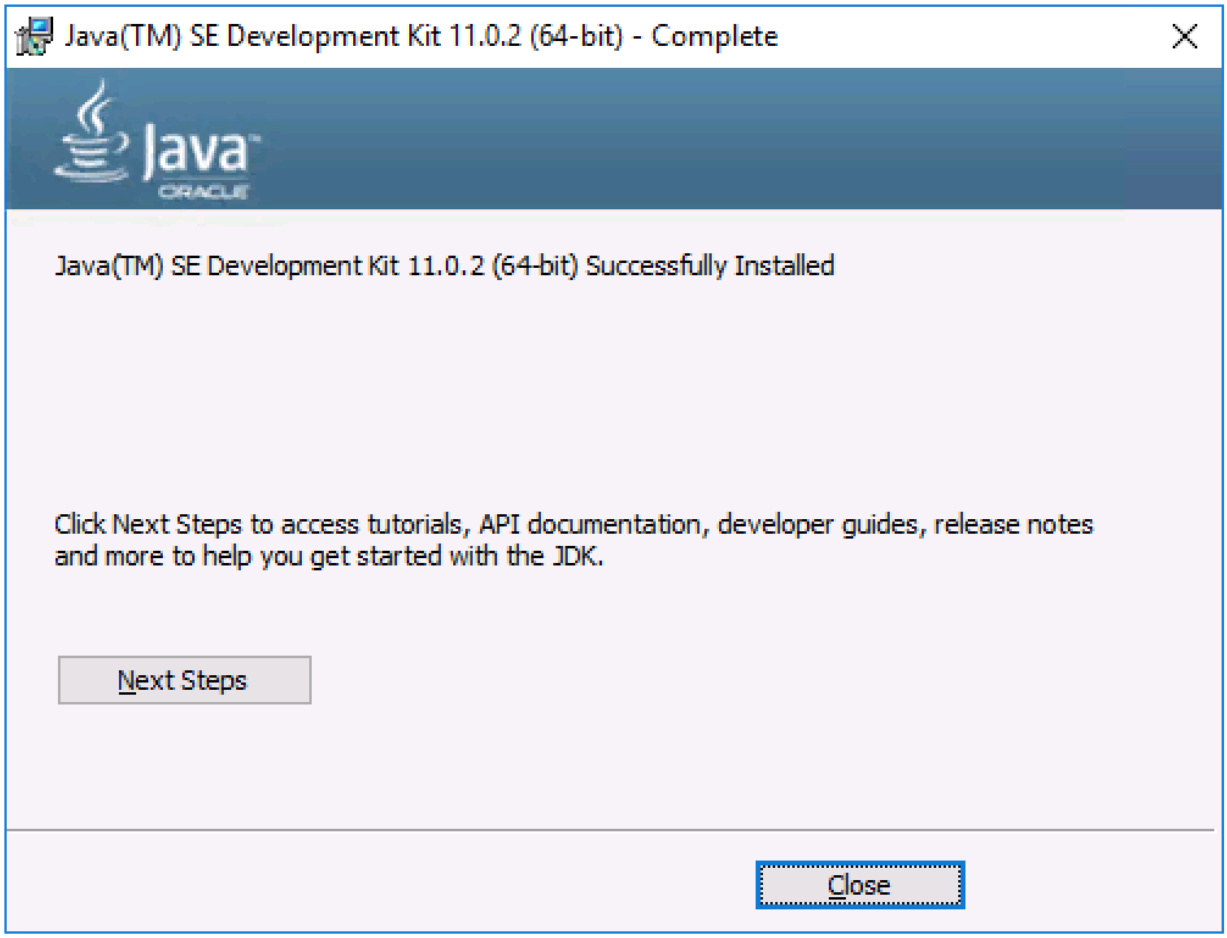
19.1 Prerequisites

Before you begin installation, you should make sure the following prerequisites are met:

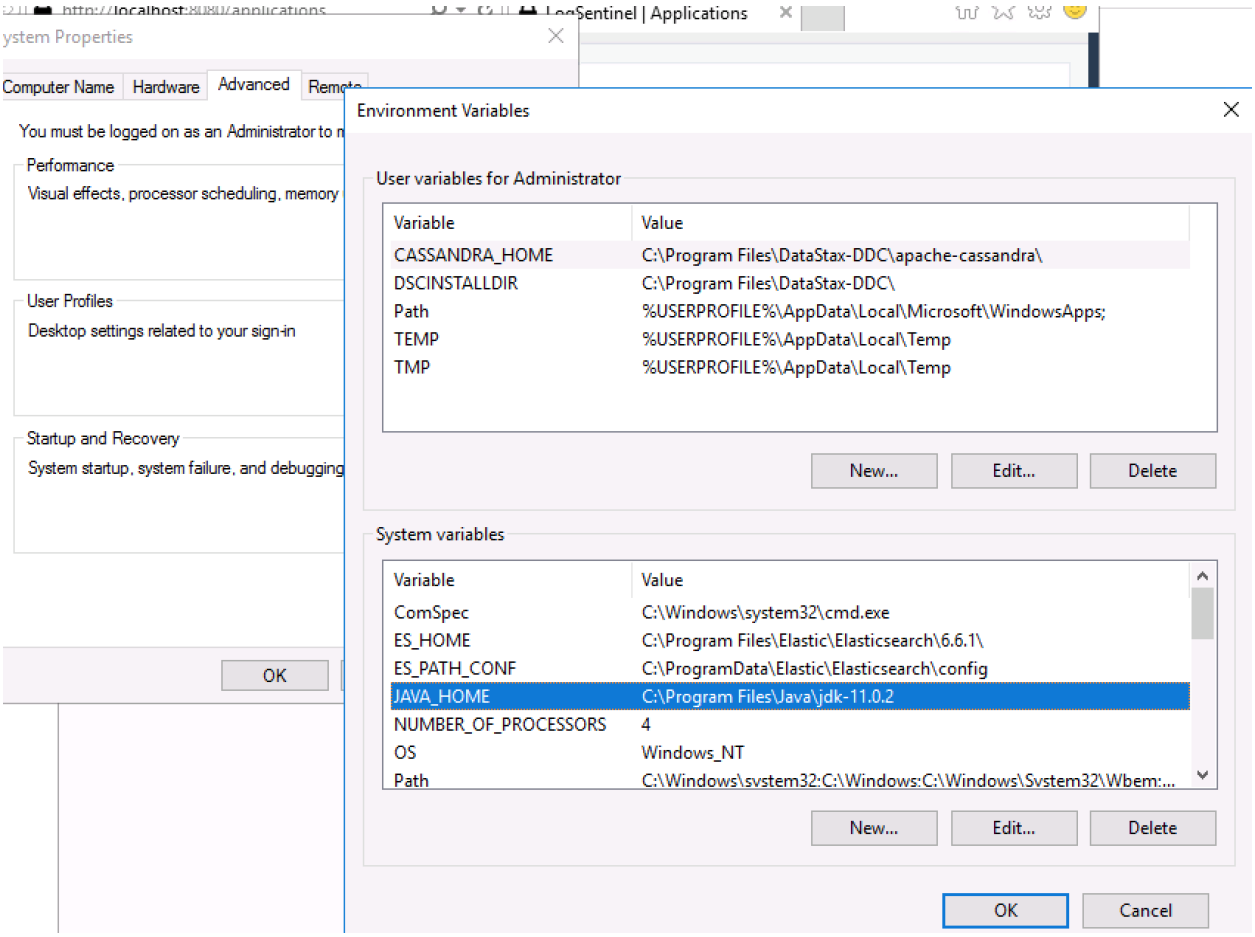
- You have downloaded the JDK installer https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html (jdk-11.0.2_windows-x64_bin.exe)
- You have downloaded the ElasticSearch installer https://s3-eu-west-1.amazonaws.com/logsentinel-public/elasticsearch-6.6.1.msi
- You have downloaded the Cassandra installer https://s3-eu-west-1.amazonaws.com/logsentinel-public/datastax-ddc-64bit-3.9.0-install.msi
- You have downloaded the logsentinel installer zip (obtained via your LogSentinel account manager)

19.2 Installing Java

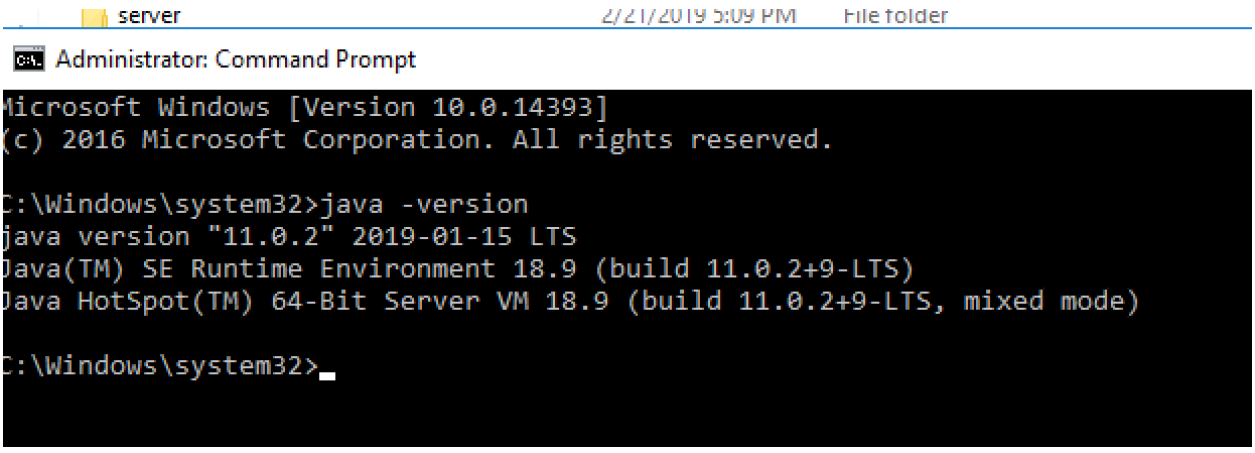
- Run the JDK installer (jdk-11.0.2_windows-x64_bin.exe) and follow the instructions.



- Make sure the PATH is configured properly



- To make sure Java is installed, run cmd and type “java -version”



19.3 Installing Cassandra

Cassandra for Windows is packaged in the Datastax DCC. Run the installer (datastax-ddc-64bit-3.9.0-install.msi) and follow the instructions. .. image:: cassandra-windows.png

To make sure Cassandra is installed and running, run cqlsh. .. image:: cqlsh-windows.png

Finally, create a keyspace: ‘‘ CREATE KEYSPACE IF NOT EXISTS logsentinel WITH REPLICATION = { ‘class’: ‘SimpleStrategy’, ‘replication_factor’: 1 } AND DURABLE_WRITES = true; ‘‘

19.4 Installing ElasticSearch

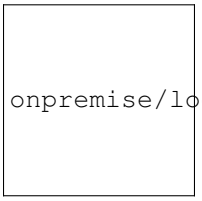
Run elasticsearch-6.6.1.msi and follow the instructions.

To make sure ElasticSearch is installed and running, open http://localhost:9200/ in a browser .. image:: elasticsearch-windows.png

You may need to create a `log` folder under `C:\Program Files\Elastic\Elasticsearch\6.6.1`

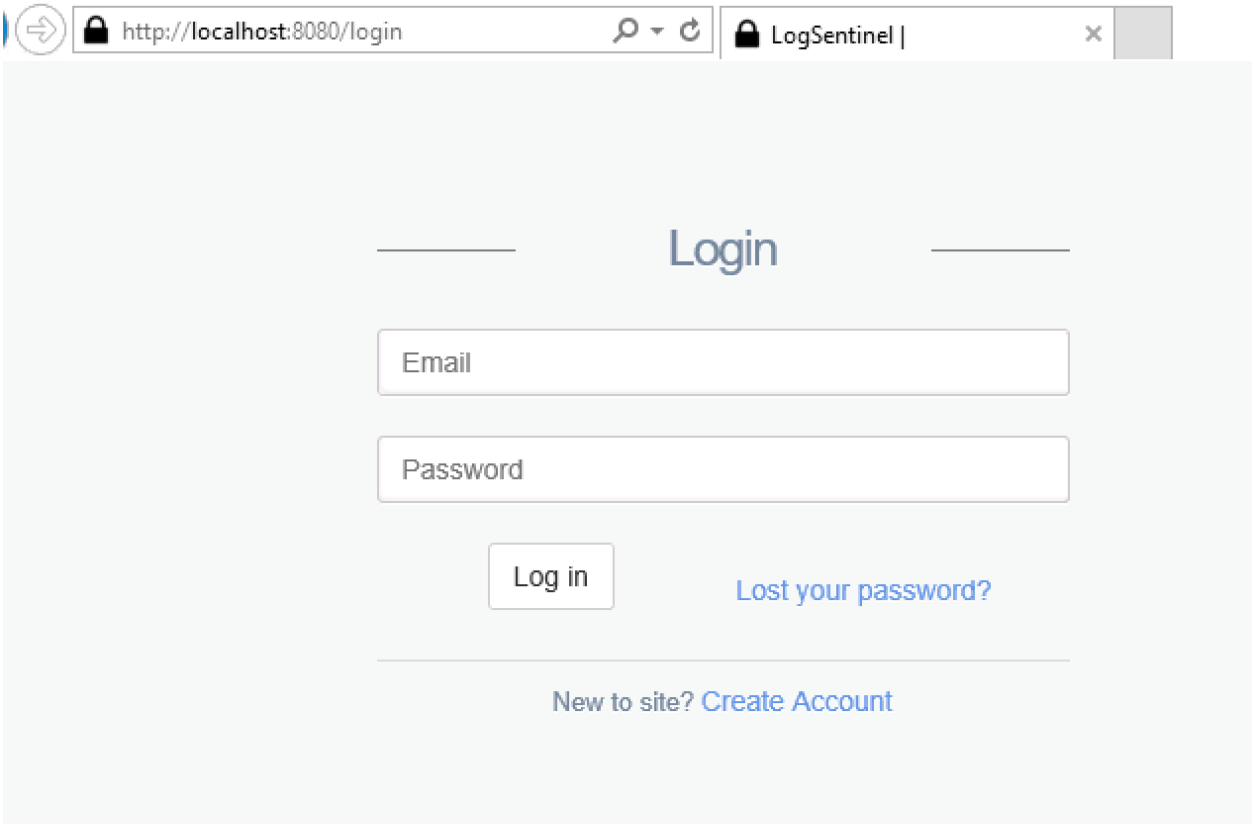
19.5 Installing LogSentinel’s SentinelTrails

Extract the logsentinel-installer.zip archive in c:logsentinel and run install.bat (you need to have administrative privileges). Then make sure the service has been installed.



Edit application.properties by setting: * `root.url` – http://{the domain name to be used for the service}. It can set to http://localhost:8080 initially and changed later * `admin.password` – password for the admin user * `spring.mail.*` properties to configure outgoing email settings (can be set at a later stage)

Go to “Services” and start the LogSentinel service. Open http://localhost:8080 (or the address that you configured)



Finally, use admin@logsentinel.com/{admin password} to log in to the administrative account.

CHAPTER

TWENTY

ON-PREMISE INSTALLATION USING DOCKER

If you want to use Docker for on-premise installation, our team will give you access to the latest sentinel trails docker image and a docker-compose script. Then you'd be able to run it using:

```
docker-compose up -d
```

CHAPTER

TWENTYONE

WHITELABELLING

SentinelTrails allows on-premise installations to be whitelabelled. To use whitelabeling:

- Place directory named `config` in the dir where you start the application from.
- Place file named `application.properties` inside config dir
- The following properties can be overridden:
  - `styling.dir` - path to the dir where logo and css files stay
  - `styling.footer` - can contain html (it's not escaped)
  - `styling.logo` - name of file inside <styling.dir>; if <styling.dir> is not defined this can be url
  - `styling.css` - name of file inside <styling.dir>; if <styling.dir> is not defined this can be url
  - `styling.title` - page title
  - `styling.login.name` - name on login page
  - `styling.login.footer` - footer on login page ;can contain html (it's not escaped)

Example:

```
styling.dir=file:config/whitelabel/
styling.footer=&copy; 2018
styling.logo=your_logo.jpg
styling.css=your_styles.css
styling.title=White Label
styling.login.name=Company name
styling.login.footer=&copy; 2019
```

CHAPTER

TWENTYTWO

LDAP & ACTIVE DIRECTORY

SentienlTrails supports integration with LDAP and ActiveDirectory for authentication. Below is a set of steps to be executed in order to configure the integration:

22.1 LDAP configuration

In order to authenticate with LDAP server (not Windows AD) the following steps must be done:

- in `application.properties` set `ldap.auth.enabled=true` and ``ad.auth.enabled=false`
- set ‘‘ldap.url‘ to pint to your LDAP server
- let’s assume the following ldap tree:

```
dn: dc=yourOrganization,dc=org
objectclass: top
objectclass: domain
objectclass: extensibleObject
dc: ourOrganization

dn: ou=groups,dc=springframework,dc=org
objectclass: top
objectclass: organizationalUnit
ou: groups

dn: ou=people,dc=springframework,dc=org
objectclass: top
objectclass: organizationalUnit
ou: people

dn: uid=ben@yourOrganization.org,ou=people,dc=yourOrganization,dc=org
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Ben Alexx
sn: Alex
organizationId: ba2cbc90-5424-11e8-b88d-6f2c1b6625e8
uid: ben@yourOrganization.org
userPassword: {SHA}nFCebWjxfaLbHHG1Qk5UU4trbvQ=

dn: cn=logsentinel_developer,ou=groups,dc=springframework,dc=org
objectclass: top
```

(continues on next page)

(continued from previous page)

```
objectclass: groupOfUniqueNames
cn: logsentinel_developer
ou: logsentinel_developer
uniqueMember: uid=ben@yourOrganization.org,ou=people,dc=yourOrganization,dc=org
```

- if ldap server doesn't give read access to annonymos users ldap.manager.DN and ldap.manager.password should be set
- user is searched with ldap.userDN.pattern=uid={0},ou=people -> ben@yourOrganization.org
- gropus are searched with ldap.group.searchbase=ou=groups
- encrypted password pattern : ldap.password.attribute=userPassword
- Names of the user are axtracted by ldap.names.param=cn
- Organization id of the user (in logsentinel) can be assigned with the default value `ldap.default.organizationId`. It can also be retrieved from LDAP if it is set to the user as an additional property `ldap.organizationId.param=organizationId`
- user role is retrieved from the groups that user is included. Only groups starting with `logsentinel_` are taken into account. For example here `ben@yourOrganization.org` is member of the group `logsentinel_developer`, so the retrieved role will be DEVELOPER. There are 5 acceptable roles: DEVELOPER, MANAGER, AUDITOR, ADMIN, IT

22.2 Active Directory configuration

In order to authenticate with Windows Active Directory:

- set `ad.auth.enabled=true` and `ldap.auth.enabled=false` in `application.properties`
- set `ldap.url` to the URL of the Active Directory (without the dc parts)
- set `ad.auth.domain` to the domain where users belong in AD
- roles are retrieved from the `memberOf` attributes of the user in AD. Same roles restrictions as in LDAP apply.

CHAPTER

TWENTYTHREE

ON-PREMISE SECURITY

In the cloud version of SentinelTrails we take all necessary operational security measures. However, in on-premise deployments this becomes the responsibility of the customer. Here are a few recommendations for keeping the installation secure.

23.1 Install TLS certificate

Note: If you are using an Nginx load balancer, you can configure it to terminate the TLS session and in that case you can leave your application node connections to be unencrypted. For more about that, see Load Balancing and High Availability.

The certificate generation process can be seen in this tutorial. The properties that need to be set in `application.properties` are:

```
server.ssl.enabled=true
server.ssl.key-store=/path/to/server.jks
server.ssl.key-store-type=JKS
server.ssl.key-store-password=secret
server.ssl.key-alias=server
server.ssl.key-password=secret
secure.headers=true
root.url=https://{your-root-url}
root.url.api=https://{{your-root-url}
```

23.2 Configure network restrictions

The Sentinel Trails (virtual) machines should only have the required ports opened. That includes:

- **application nodes:**
  - incoming: 8080 (for web requests), 1514, 1515, 1516 (for syslog), 5701 (for hazelcast) and 22 for SSH.
  - outgoing: 80, 443, 9200 (for elastic search), 9042 (for cassandra), 5701, 465 (for smtp)
- **database nodes:**
  - incoming: 7000, 7001, 7199, 9042, 9160 and 22
  - outgoing: 80, 443, 7000, 7001

- **search nodes:**
  - incoming: 9200, 9300, 22
  - outgoing: 80, 443, 9300
- **load balancer node:**
  - incoming: 80, 443, 22
  - outgoing: 80, 443, 8080

23.3 Configure administrator access restrictions

Ideally all nodes should be protected via 2-factor authentication. You can do that by executing the following setup-2fa.sh script:

```
#!/bin/sh
# Execute manually AFTER the host has been setup
# Based on https://aws.amazon.com/blogs/startups/securing-ssh-to-amazon-ec2-linux-
↪hosts/
sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.
↪noarch.rpm
sudo yum -y install google-authenticator
# Execute this line for each user manually, before they can login. Use > sudo su
↪<user> before that
google-authenticator --time-based --disallow-reuse --force --rate-time=30 --rate-
↪limit=3 --window-size=8
sudo echo "auth required pam_google_authenticator.so" >> /etc/pam.d/sshd
sudo sed -i -- 's/ChallengeResponseAuthentication no/ChallengeResponseAuthentication␣
↪yes/g' /etc/ssh/sshd_config
sudo sed -i -- 's/auth       substack     password-auth/#auth       substack     ␣
↪password-auth/g' /etc/pam.d/sshd
sudo echo "\nAuthenticationMethods publickey,keyboard-interactive" >> /etc/ssh/sshd_
↪config

sudo service sshd restart
```

23.4 Track administrative access through a custom PAM

We provide a custom PAM which you can get by following the instructions here. The PAM makes sure that each administrative access is pushed directly to either Ethereum or a qualified trust service provider. That way, in case an administrator tries to manipulate the logs that will not only be detected, but they won't be able to cover who they were. The PAM does some additional checks, e.g. if the network is not blocked, and does not let the administrator login if the log event can't be pushed properly.

CHAPTER

TWENTYFOUR

LOAD BALANCING AND HIGH AVAILABILITY

On-premise installations would often require multiple application nodes, database nodes and search nodes to be run alongside for high availability. And therefore load balancing is required.

24.1 Application node load balancing

One option of application node load balacing is to use round-robin DNS. That way a client will randomly connect to one of the listed nodes. Healtchecks will have to be implemented manually, however, because in case of one node going down, the DNS A-record will still contain its IP and send traffic to it.

Another opton is to configure an Nginx load-balancer and setup a Let's encrypt certificate using the following script:

24.2 Database load balancing

There are multiple ways to do load balancing with Cassandra in case you are running more than one node (which is highly recommended):

- Configuring a comma-separated list of cassandra nodes with the `cassandra.hosts` property
- TCP Nginx load balancing as shown above for the syslog backend
- Round-robin DNS

24.3 Elasticsearch load balancing

When running multiple Elasticsearch nodes (which recommended), you have similar options:

- Configuring a comma-separated list of elasticsearch nodes with the `elasticsearch.url` property
- Use HTTP Nginx load balancing for port 9200 (as shown above)
- Round-robin DNS

24.4 Other configurations

You have to set the `hazelcast.nodes` property to contain a comma-separated list of the IPs of application nodes