Fighting Cybercrime:
A Joint Task Force of Real-Time Data and Human Analytics

William Callaghan
eSentire Inc.

About eSentire

- Founded in 2001
- Located in Cambridge, Ontario, Canada
- Managed Detection and Response for Mid-Sized Enterprises
  - Real-Time Analytics
  - 24/7 Security Operations Centre
  - Threat Intelligence Team

CYBERCRIME
IS BIG BUSINESS

$70B

spent on cybersecurity

$375-575B

in estimated losses

MEANS | MOTIVE |OPPORTUNITY

Easy Access to Cyber Weaponry

Minimal Cyber Skills Required

Motivation is High

No Negative Repercussions

THREAT ACTORS

CRIMINAL

ORGANIZED CRIME

HACTIVIST

INSIDER

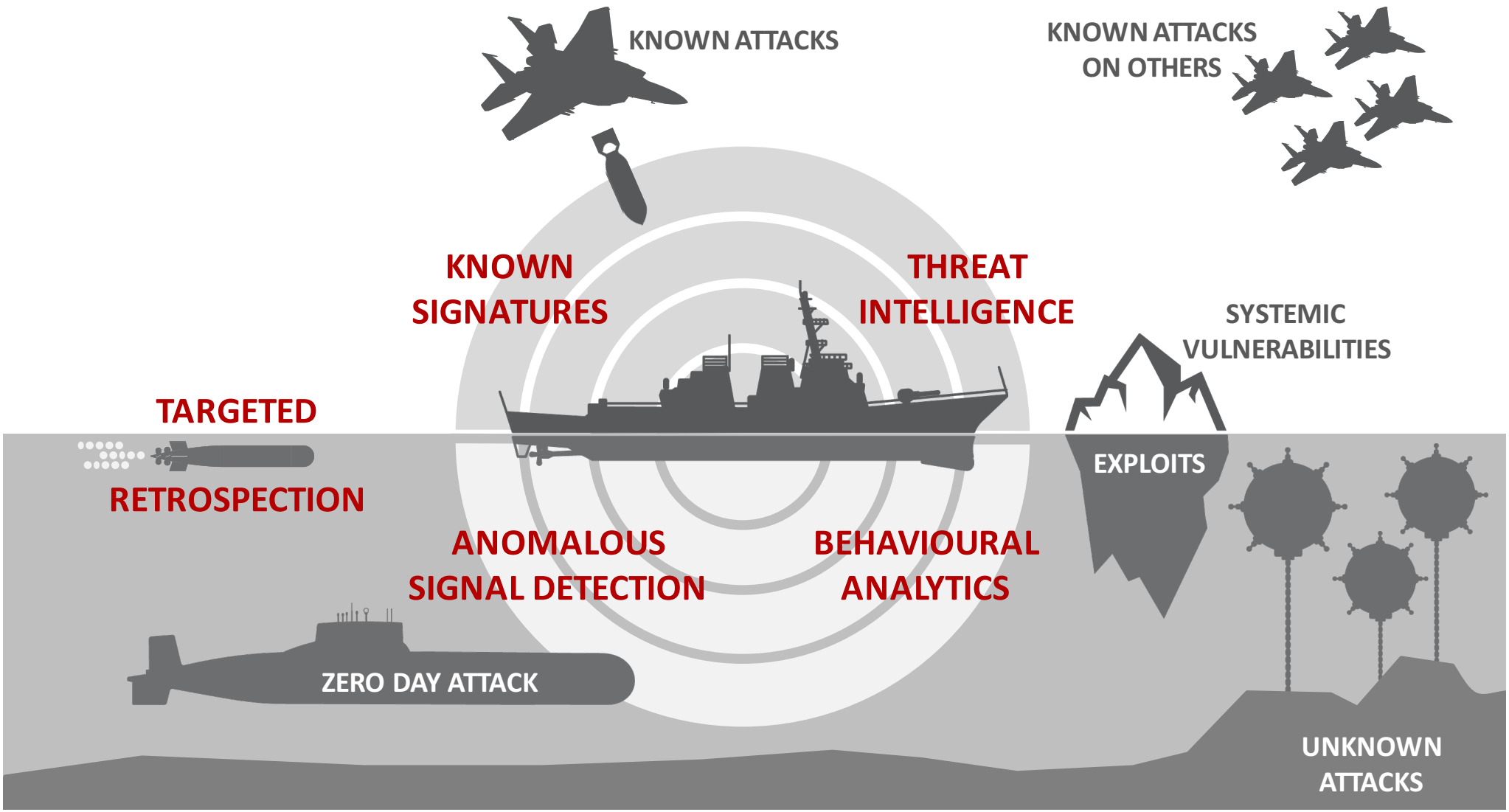
NATION STATE ACTOR

TERRORIST

The majority of cyber defenses protect against **KNOWN** threats.

LAYERS OF SECURITY

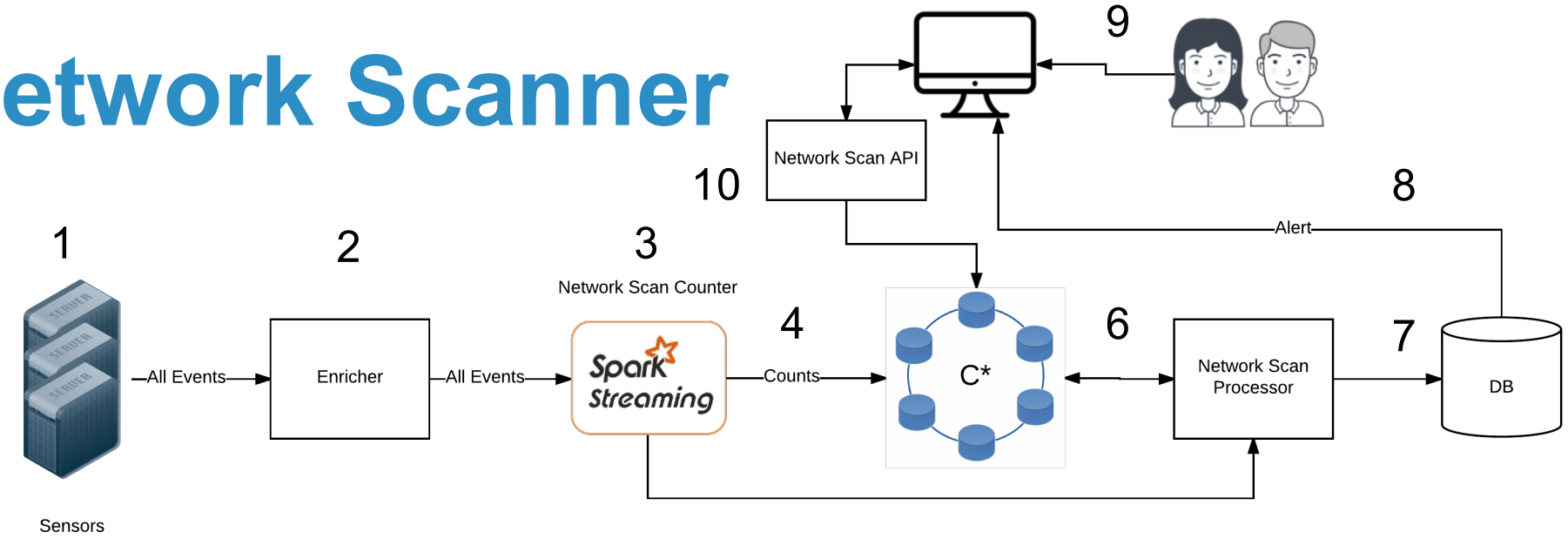
NAC | WEB PROXY | MALWARE SANDBOXING | ROUTER | FIREWALL | ANTI VIRUS | USER ID AND PASSWORD UTM | IPS/IDS | AUTHENTICATION TOKENS | HIPS | DLP | WAF | SPAM FILTER


KNOWN ATTACKS
KNOWN ATTACKS ON OTHERS
KNOWN SIGNATURES
THREAT INTELLIGENCE
SYSTEMIC VULNERABILITIES
TARGETED
RETROSPECTION
EXPLOITS
ANOMALOUS SIGNAL DETECTION
BEHAVIOURAL ANALYTICS
ZERO DAY ATTACK
UNKNOWN ATTACKS

Network Scanner

- A tool for exploring connection information from a given IP Address.
- Analysts can get breakdown of traffic by:
  - IP, Port, Minute or any permutation of the 3.
- Cassandra table for each type of breakdown.
  - Data is modeled based on queries, not relations.
  - Therefore, there is data duplication.

Network Scanner



Important Event fields:

- Company
- Site
- Host (IP)
- Event bucket (Minute of event)
- Event time (when the event occurred on the sensor)
- Received time (when the event was processed by the counter)

Network Scan Counter:

- Counts # of connections for each (company,site,host) in every minute.

Drawbacks

With 1000+ sensors in the field, some sending 100,000+ events/hour:

- Scalability of having a C* table for each method of traffic breakdown?
- Limited to the use cases (tables created) at the time.

Goals

Reduce data duplication and load on C* while being able to:

- Perform flexible queries over large datasets.
- Have minimal latency.

Solution

When an alert is generated for a (Company,Site,Host) for a given hour window:

- Load the partition(s) into Spark.
  - (Company, Eventbucket)
  - Restrict by “Receive Time” to get a snapshot of the data that led to the alert being triggered.
- Perform the query in-memory and return the results.

Overhead

- Overhead in starting Spark applications
  - Creation of Spark Context
  - Connecting to Cassandra (or other sources)
  - Acquiring Executors
  - Distributing application code to workers

Spark Job Server

- Open-source project originally created by Ooyala.
- Provides a persistent connection to Spark, relieving the overhead.
- HTTP API to pass in arguments to and run a Spark application.
- Can now run successive Spark SQL queries without the overhead.

```
import spark.jobserver.api.{SparkJob -> NewSparkJob, _}

object MySparkJob extends NewSparkJob {
    type JobData = Config
    type JobOutput = Any

    def validate(sc:SparkContext, runtime:JobEnvironment, data:JobData): JobData or
        Every[ValidationProblem] = {
        // Data validation steps go here
    }

    def runJob(sc:SparkContext, runtime: JobEnvironment, data:JobData): JobOutput {
        // Extracting arguments
        val queryString = data.getString("sql");
    }
}
```

Querying The Same Dataset

What if we want to make different queries on the same dataset?

- We shouldn't have to load the data in each time.

Caching in Spark Job Server

- Can cache RDDs, DataFrames in memory.
- They are then available to any Spark application running on the Job Server
  - Methods: Get, Update, Forget.
- Naming Scheme:
  - “COMPANY_START_END_RECVTIME”

```
object MySparkJob extends NewSparkJob with NamedObjectSupport {
    …
```

Caching in Spark Job Server

- Managing a large number of datasets using Spark Job Server can be tedious.
  - Datasets had to be managed within a Spark application.

Alluxio: In-Memory Distributed Storage

- An in-memory, distributed file system.
- Filesystem API supports frameworks such as: Spark, MapReduce.
- Can query Parquet files from Alluxio.
- Can set a TTL on datasets.
- Fault-tolerance mode for HA.
- Can promote datasets to HDFS.

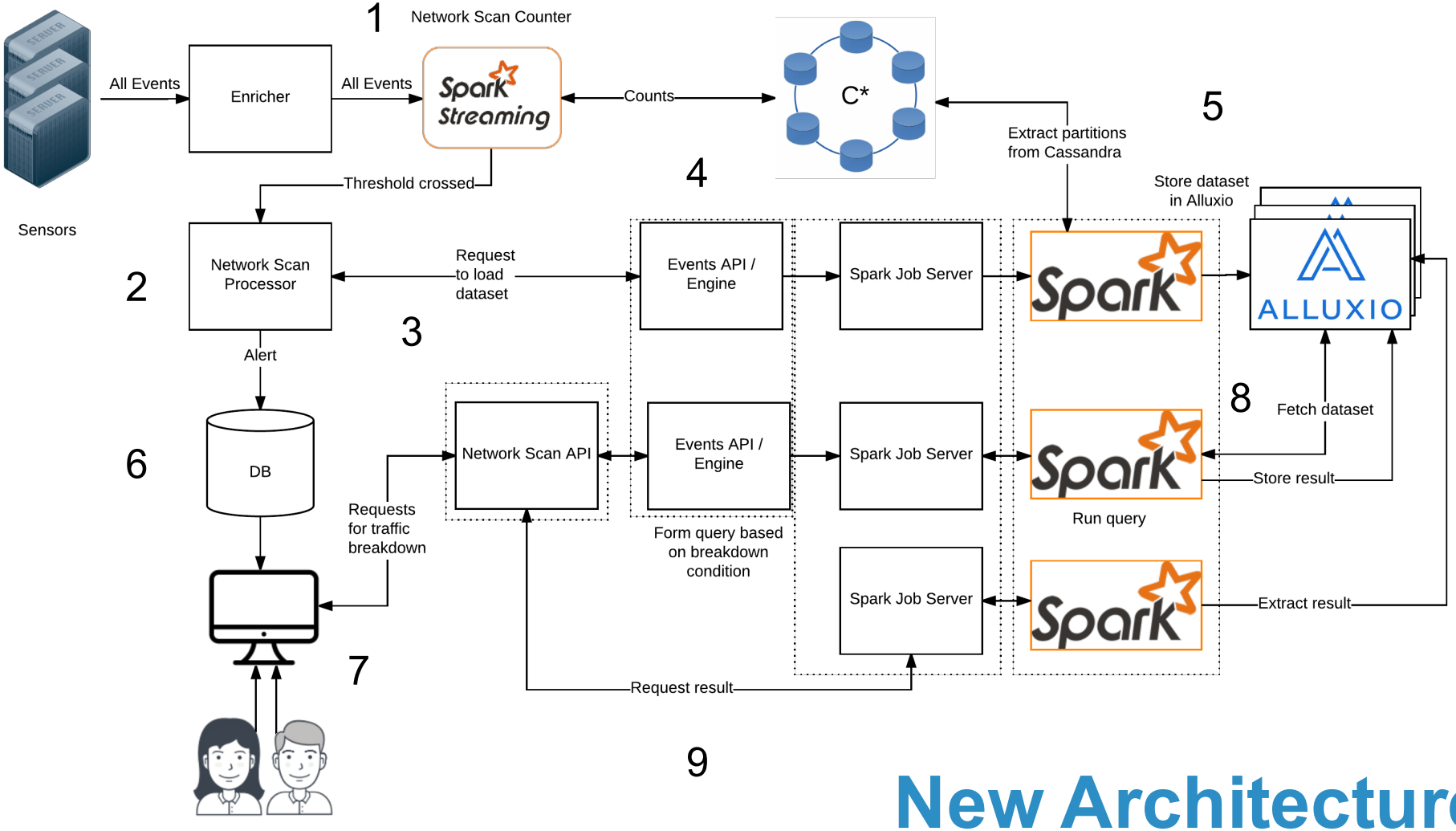
```
import spark.jobserver.api.{SparkJob -> NewSparkJob, _}
import alluxio.client.file._
import alluxio._

…

def runJob(sc:SparkContext, runtime: JobEnvironment, data:JobData): JobOutput {
        val sqlContext = new SQLContext(sc);
        val fileSystemMaster = "alluxio-ft://<IP>:19998"
        val fileSystem = BaseFileSystem.get();
        val uri = new AlluxioURI(fileSystemMaster+"/data/COMPANY_START_END_RECV")
        val query = "SELECT COUNT(*) FROM <uri> WHERE ..."

        if (fileSystem.exists(uri) && fileSystem.getStatus(uri).isCompleted())
        {

                val result = sqlContext.sql(query)
                result.write.parquet(fileSystemMaster+"/results/"+runtime.jobId)
        }
}
```



New Architecture

Conclusions

- Expanded capabilities of our Network Scanner tool to support flexible queries with response times of ~1-2 seconds.

- Analysts now have a more powerful tool for exploratory analysis of network activity.

Thank You.

Questions?

William Callaghan
william.callaghan@esentire.com