Hadoop and Kerberos: The madness beyond the gate

Authors:

S. A. Loughran

Word count: 2,253

Introduction

When HP Lovecraft wrote his books about forbidden knowledge which would reduce the reader to insanity, of "Elder Gods" to whom all of humanity were a passing inconvenience, most people assumed that he was making up a fantasy world.

In fact he was documenting Kerberos.

What is remarkable is that he did this fifty years before kerberos was developed. This makes him less of an author, instead: a prophet.

What he wrote was true: there are some things humanity was not meant to know. Most people are better off living lives of naive innocence, never having to see an error message about SASL or GSS, never fear building up scripts of incantations to "kadmin.local", incantations which you hope to keep evil and chaos away. To never stare in dismay at the code whose true name must never be spoken, but instead it's initials whispered, "UGI". For those of us who have done all this, our lives are forever ruined. From now on we will cherish any interaction with a secure Hadoop cluster -from a client application to HDFS, or application launch on a YARN cluster, and simply viewing a web page in a locked down web UI -all as a miracle against the odds, against the forces of chaos struggling to destroy order.

And forever more, we shall fear those voices calling out to us in the night, the machines by our bed talking to us, saying things like "we have an urgent support call related to REST clients on a remote kerberos cluster -can you help?"

This documents contains the notes from previous people who have delved too deep into the mysteries of Hadoop and Kerberos, who have read the forbidden source code, maybe who have even contributed to it. If you wish to preserve your innocence, to view the world as a place of happiness: stop now.

Foundational Concepts

What is the problem that Hadoop security is trying to address?

Apache Hadoop is "an OS for data". A Hadoop cluster can rapidly become the largest stores of data in an organisation. That data can explicitly include sensitive information: financial, personal, business, and can often implicitly contain data which needs to be sensitive about the privacy of individuals (for example, log data of web accesses alone). Much of this data is protected by laws of different countries. This means that access to the data needs to be strictly controlled, and accesses made of that data potentially logged to provide an audit trail of use.

You have to also consider, "why do people have Hadoop clusters?". It's not just because they have lots of data --its because they want to make use of it. A data-driven organisation needs to trust that data, or at least be confident of its origins. Allowing entities to tamper with that data is dangerous.

For the protection of data, then, read and write access to data stored directly in the HDFS filesystem needs to be protected. Applications which work with their data in HDFS also need to have their accesses restricted: Apache HBase and Apache Accumulo store their data in HDFS, Apache Hive submits SQL queries to HDFS-stored data, etc. All these accesses need to be secured; applications like HBase and Accumulo granted restricted access to their data, and themselves securing and authenticating communications with their clients.

YARN allows arbitrary applications to be deployed within a Hadoop cluster. This needs to be done without granting open access to the entire cluster from those user-launched applications, while isolating different users' work. A YARN application started by user Alice should not be able to directly manipulate an application launched by user "Bob", even if they are running on the same host. This means that not only do they need to run as different users on the same host (or in some isolated virtual/container), the applications written by Alice and Bob themselves need to be secure. In particular, any web UI or IPC service they instantiate needs to have its access restricted to trusted users. here Alice and Bob

Authentication

Authorization

Encryption

Auditing

The Limits of Hadoop Security

What are the limits of Hadoop security? Even with Kerberos enabled, what vulnerabilities exist?

Unpatched and 0-day holes in the layers underneath.

The underlying OS in a Hadoop cluster may have known or 0-day security holes, allowing a malicious (YARN?) application to gain root access to a host in the cluster. Once this is done it would have direct access to blocks stored by the datanode, and to secrets held in the various processes, including keytabs in the local filesystems.

Defences

1. Keep up to date with security issues. (SANS is worth tracking), and keep servers up to date.

2. Isolate the Hadoop cluster from the rest of your network infrastructure, apart from some "edge" nodes, so that only processes running in the cluster.

3. Developers: ensure that your code works with the more up to date versions of operating systems, JDKs and dependent libraries, so that you not holding back the upgrades. Do not increase the risk for the operations team.

Failure of users to keep their machines secure

The perennial problem. Securing end-user machines is beyond the scope of the Hadoop project.

However, one area where Hadoop may impose risk on the end-user systems is the use of Java as the runtime for client-side code, so mandating an installation of the JVM on those users who need to directly talk to the Hadoop services.

Ops teams should

* Make sure that an up to date JVM/JRE is installed, out of date ones are uninstalled, and that Java Applets in browsers are completely disabled.

* Control access to those Hadoop clusters and the services deployed on them.

* Use HDFS Quotas and YARN Queues to limit the resources malicious code can do.

* Collect the HDFS audit logs and learn how to use them to see if, after any possible security breach, you are in a position to even state what data was accessed by a specific user in a given time period.

We Hadoop developers need to

1. Make sure that our code works with current versions of Java, and test against forthcoming releases (a permanent trouble spot).

2. Make sure that our own systems are not vulnerable due to the tools installed locally.

3. Work to enable thin-client access to services, through REST APIs over Hadoop IPC and other Java protocols, and by helping the native-client work.

4. Ensure our applications do not blindly trust users -and do as much as possible to prevent privilege escalation.

5. Log information for ops teams to use in security audits.

Denial of service attacks.

Hadoop is its own Distributed Denial of Service platform. A misconfiguration could easily trigger all datanodes to attempt to report in so frequently that the namenode gets overloaded, triggering apparent timeouts of some DN heartbeats, leading to the namenode assuming it has failed and starting block transfers of under-replicated blocks, so impacting network load and reporting even more. This is not a hypothetical example: Facebook had a cluster outage from precisely such an event, a failing switch partitioning the cluster and triggering a cascade failure. Nowadays IPC throttling (from Twitter) and the use of different ports on the namenode for heartbeating and filesystem operations (from Facebook) try to keep this under control.

We're not aware of any reported deliberate attempts to use a Hadoop cluster to overload local/remote services, though there are some anecdotes of the Yahoo! search engines having be written so as to deliberately stripe searches not just across hosts, but domains and countries, so as not to overload the DNS infrastructure of small countries. If you have some network service in your organisation which is considered critical (Examples: sharepoint, exchange), then configure the firewall rules to block access to those hosts and service ports from the Hadoop cluster.

Other examples of risk points and mitigation strategies

YARN resource overload

Too many applications asking for small numbers of containers, consuming resources in the Node Managers and RM. There are minimum size values for YARN container allocations for a reason: it's good to set them low on a single node development VM, but in production, they are needed

DNS overload.

This is easily done by accident. Many of the large clusters have

local caching DNS servers for this reason, especially those doing any form of search.

CPU, network IO, disk IO, memory

YARN applications can consume so much local resources that they hurt the performance of other applications running on the same nodes.

In Linux and Windows, CPU can be throttled, the amount of physical and virtual memory limited. We could restrict disk and network IO (see relevant JIRAs), but that won't limit HDFS IO, which takes place in a different process.

YARN labels do let you isolate parts of the cluster, so that low-latency YARN applications have access to machines across the racks which IO-heavy batch/background applications do not.

Deliberate insertion of malicious code into the Hadoop stack, dependent components or underlying OS.

We haven't encountered this yet. Is it conceivable? Yes: in the security interfaces and protocols themselves. Anything involving encryption protocols, random number generation and authentication checks would be the areas most appealing as targets: break the authentication or weaken the encryption and data in a Hadoop cluster becomes more accessible. As stated, we've not seen this. As Hadoop relies on external libraries for encryption, we have to trust them (and any hardware implementations), leaving random number generation and authentication code as targets. Given that few committers understand Hadoop Kerberos, especially at the REST/SPNEGO layer, it is hard for new code submissions in this area to be audited well.

One risk we have to consider is: if someone malicious had access to the committer credentials of a developer, could they insert malicious code? Everyone in the Hadoop team would notice changes in the code appearing without associated JIRA entries, though it's not clear how well reviewed the code is.

Mitigation strategies. A key one has to be "identify those areas which would be vulnerable to deliberate weakening, and audit patch submissions extra rigorously there", "reject anything which appears to weaken security -even something as simple as allowing IP addresses instead of Hostnames in kerberos binding (cite: JIRA) could be dangerous. And while the submitters are probably well-meaning, we should assume maliciousness or incompetence in the high-risk areas. (* yes, this applies to my own patches too. The accusation of incompetence is defendable based on past submissions anyway).

Insecure applications

SQL injection attacks are the classic example here. It doesn't matter how secure the layers are underneath if the front end application isn't handling untrusted data. Then there are things like emergency patches to apple watches because of a binary parse error in fonts.

Mitigation strategies

1. assume all incoming data is untrusted. In particular, all strings

used in queries, while all documents (XML, HTML, binary) should be treated as potentially malformed, if not actually malicious.

2. Use source code auditing tools such as Coverity Scan to audit the code. Apache projects have free access to some of these tools.

3. Never have your programs ask for more rights than they need, to data, to database tables (and in HBase and Accumulo: columns)

4. Log data in a form which can be used for audit logs. (Issue: what is our story here? Logging to local/remote filesystems isn't it, not if malware could overwrite the logs)

Hadoop IPC Security

Adding a new IPC interface to a Hadoop Service/ Application

This is "fiddly". It's not impossible, it just involves effort. In its favour: it's a lot easier than SPNEGO.

SPNEGO

SPNEGO is the acronym of the protocol by which HTTP clients can authenticate with a web site using Kerberos. This allows the client to identify and authenticate itself to a web site or a web service.

SPNEGO is supported by

* the standard browsers, to different levels of pain of use

* `curl` on the command line

* `java.net.URL` in Java7+

The final point is key: it can be used programmatically in Java, so used by REST client applications to authenticate with a remote Web Service.

CAUTION

Apache Http Components do not support SPNEGO. As the documentation says "try it and see" {cite}.

Exactly how the Java runtime implements its SPNEGO authentication is a mystery to all. Unlike, say Hadoop IPC, where the entire authentication code has been implemented by people whose email addresses you can identify from the change log and so ask hard questions, what the JDK does is a black hole.

ZOOKEEPER and SASL

Apache Zookeeper uses SASL to authenticate callers.

Other than SASL, its access control is all based around secrets which are shared between client and server, and sent over the (unencrypted) channel. This means they cannot be relied upon to securely identify callers.

Enabling SASL in ZK

SASL is enabled in ZK by setting a system property. While adequate for a server, it's less than convenient when using ZK in an application as it means something very important: you cannot have a non-SASL and a SASL ZK connection at the same time. (you could create one connection, change the system properties and then create the next, but as Apache Curator doesn't do this, and everyone sensible uses Curator to handle transient disconnections and ZK node failover, this isn't practicable). Someone needs to fix this.