Fortress ClickHouse

Secure Your Database and Foil Evildoers in 15 Minutes or Less

Robert Hodges - Altinity

A brief message from our sponsor…

Robert Hodges

Database geek with 30+ years on DBMS. Kubernaut since 2018. Day job: Altinity CEO

Vitaliy Zakaznikov

15+ years of experience in software and hardware test. Day job: Altinity Head of QA

ClickHouse support and services including Altinity.Cloud
Authors of Altinity Kubernetes Operator for ClickHouse
and other open source projects

ClickHouse is a real-time analytic database

Understands SQL

Runs on bare metal to cloud

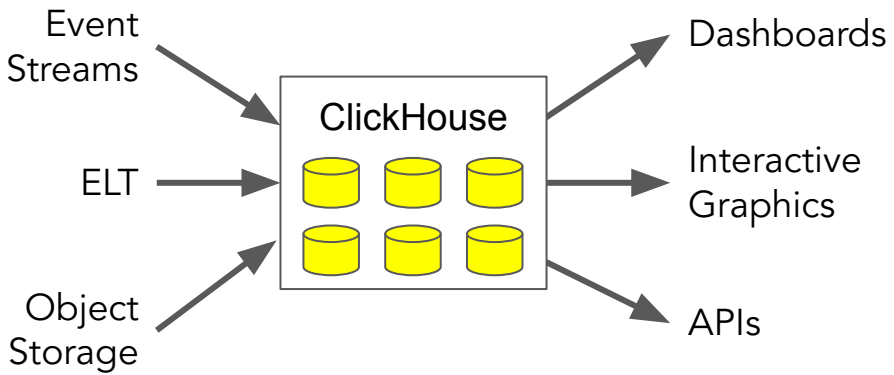
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

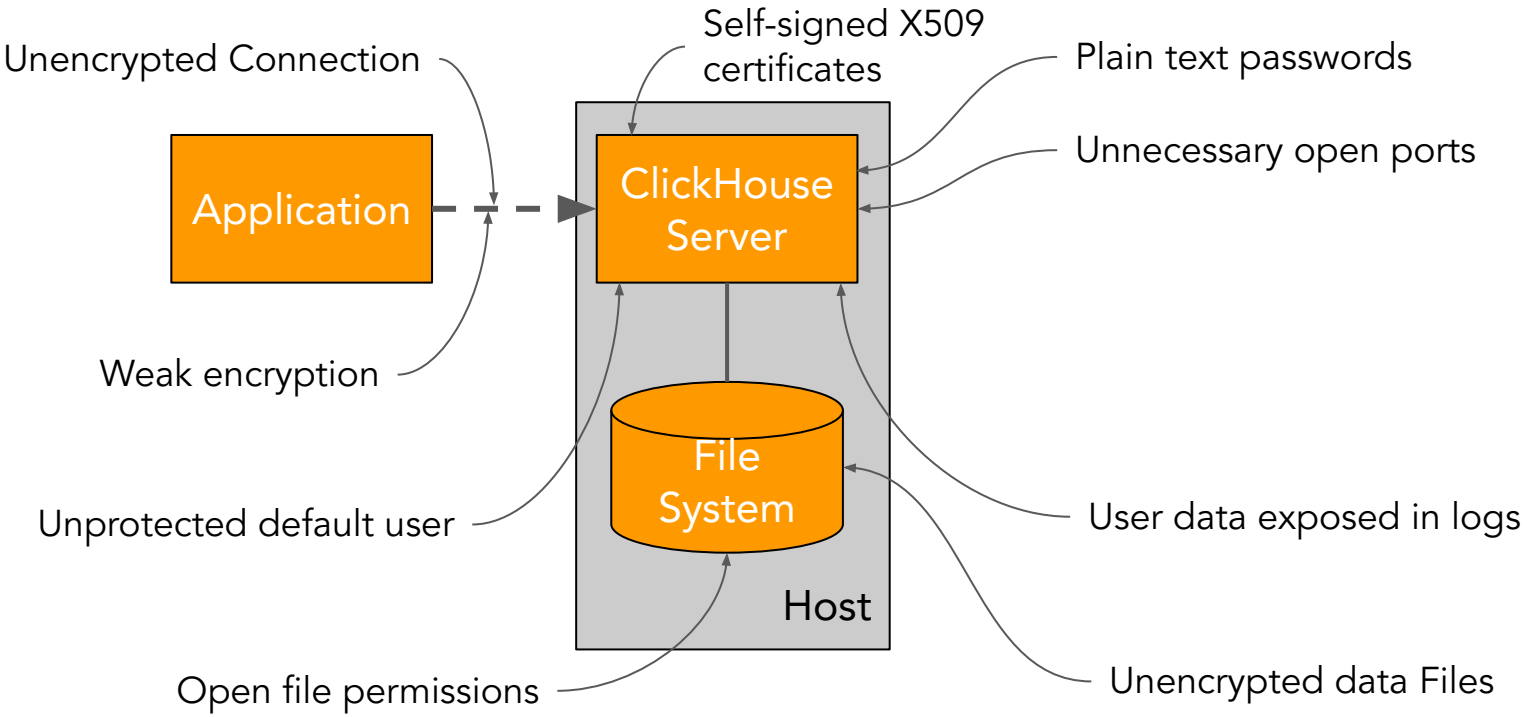
Is Open source (Apache 2.0)



It's the core engine for low-latency analytics

ClickHouse Server Attack Surfaces

Targets for hardening on ClickHouse server

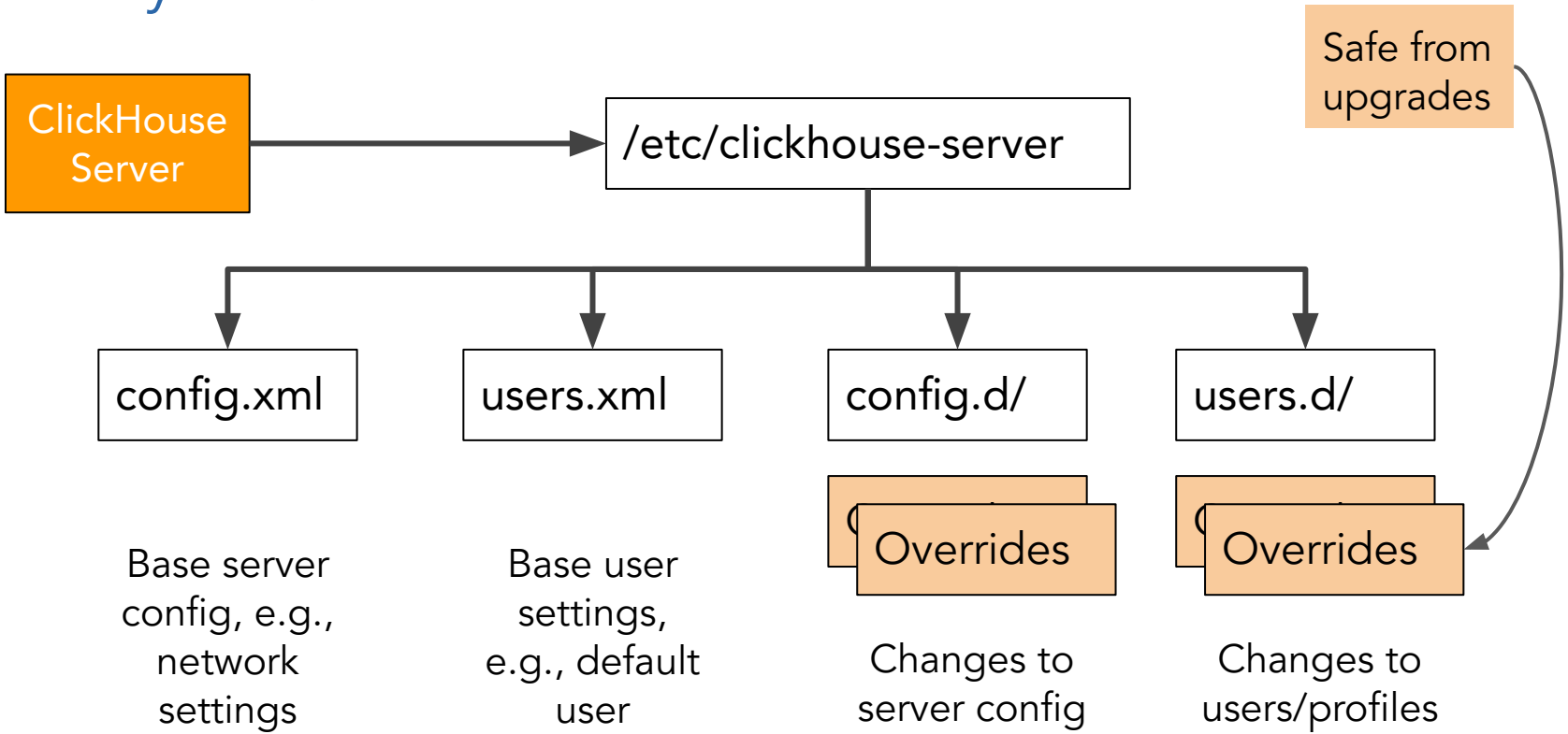


Initial ClickHouse security checks

Starting a ClickHouse test server on Docker

```
# Get ClickHouse container.
docker pull clickhouse/clickhouse-server:23.3.3.52
# Tag it.
docker tag clickhouse/clickhouse-server:23.3.3.52 \
  clickhouse:my-fortress
# Start.
docker run -d  --name my-fortress \
 -v $(pwd)/data/conf.d:/etc/clickhouse-server/conf.d \
 -v $(pwd)/data/users.d:/etc/clickhouse-server/users.d \
 -v $(pwd)/data/certs:/certs \
 clickhouse:my-fortress
```

Know your ClickHouse server file locations!



Look for plain text passwords

```
# Run on ClickHouse host
egrep -r '<password>.*</password>'
config.xml:          <password></password>
users.xml:           Example: <password>qwerty</password>.
users.xml:              <password></password>
```

Password inter-server queries

Password for default user

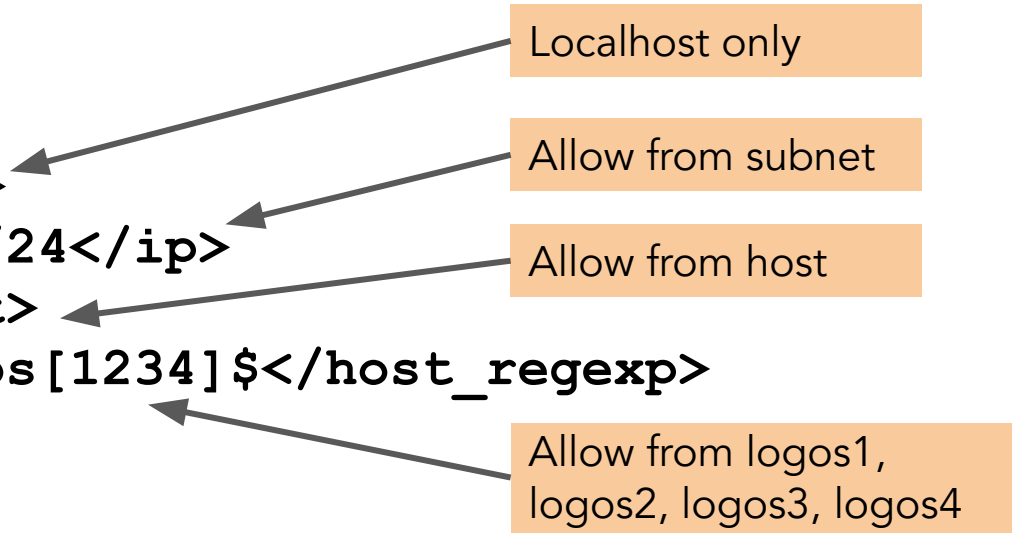
Tips for defining users

Generating passwords -- don't forget '-n'

```
echo -n "secret" | sha256sum | tr -d '-'
```

Network masks:

```
<networks>
    <ip>127.0.0.1</ip>
    <ip>192.168.128.1/24</ip>
    <host>logos2</host>
    <host_regexp>^logos[1234]$</host_regexp>
</networks>
```



Securing the default ClickHouse user

```
cat << EOF > /etc/clickhouse-server/users.d/default.xml
<clickhouse>
    <users>
        <default>
            <password remove='1' />
            <password_sha256_hex>f8cf...d8b5</password_sha256_hex>
            <networks>
                <ip>::1</ip>
                <ip>127.0.0.1</ip>
            </networks>
            <access_management>1</access_management>
        </default>
    </users>
</clickhouse>
EOF
```

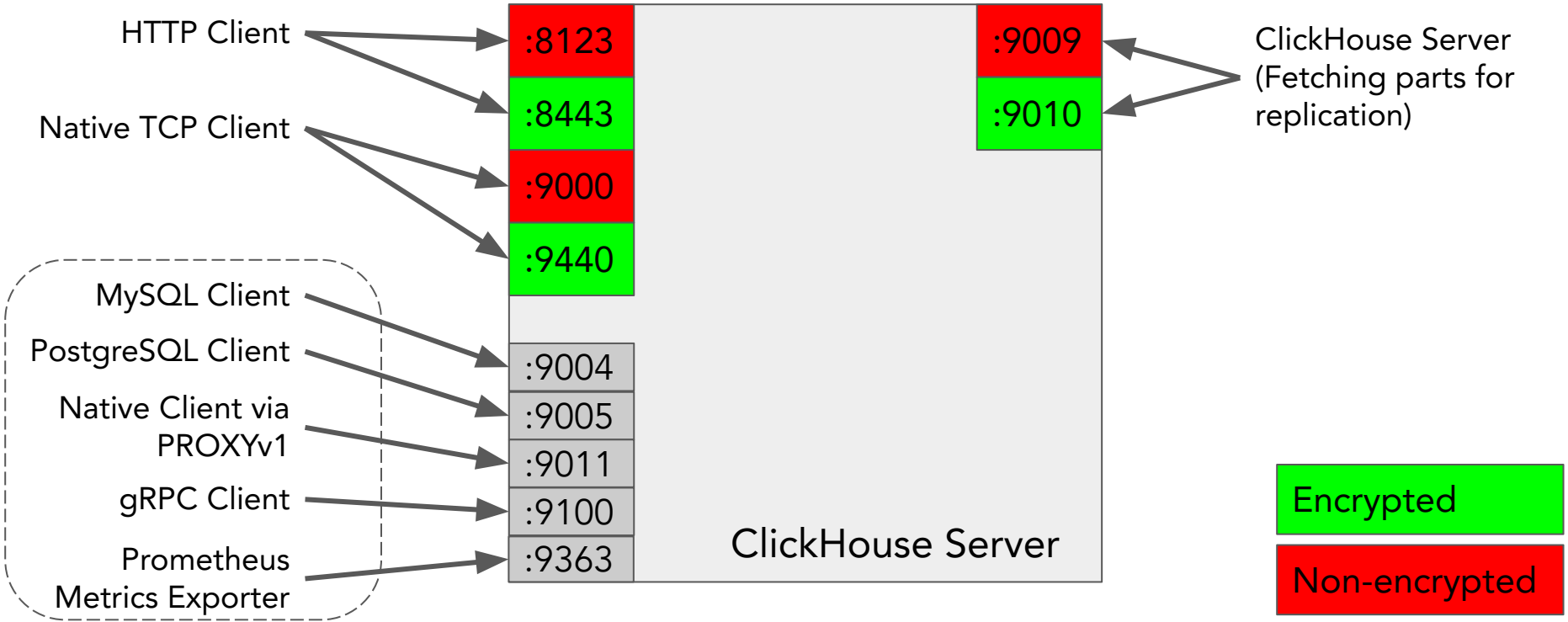
Remove blank password

Limit login to localhost

Allow default user to create more users (optional)

Locking down unused ClickHouse ports

Know your ClickHouse ports!

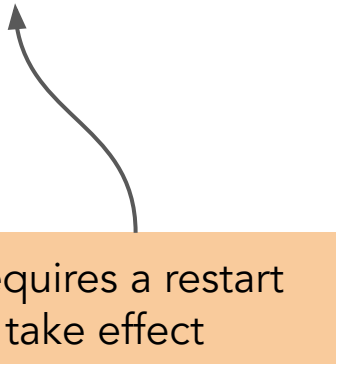


How to look for open ports…

```
# Run on ClickHouse host
netstat -tulpn | grep LISTEN
tcp        0      0 0.0.0.0:9000            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:9004            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:9005            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:9009            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:8123            0.0.0.0:*               LISTEN      -
```

Turning off undesired ports

```
cat << EOF >
/etc/clickhouse-server/conf.d/disable_mysql_and_pg_ports.xml
<clickhouse>
    <!-- Disable MySQL and PostreSQL emulation ports -->
    <mysql_port remove="true"/>
    <postgresql_port remove="true"/>
</clickhouse>
EOF
```



Requires a restart to take effect

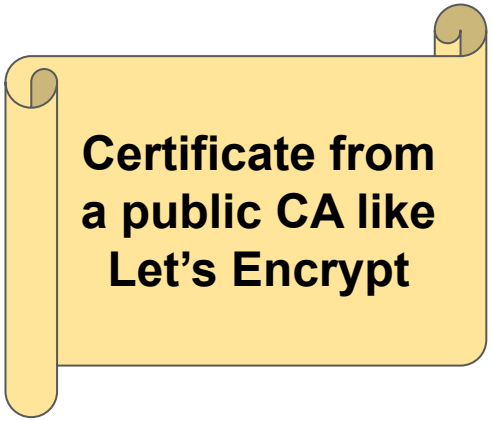
Ensure that open ports really disappeared

```
# Look for open ports
netstat -tulpn | grep LISTEN
tcp        0    0 0.0.0.0:9000      0.0.0.0:*    LISTEN      -
tcp        0    0 0.0.0.0:9009      0.0.0.0:*    LISTEN      -
tcp        0    0 0.0.0.0:8123      0.0.0.0:*    LISTEN      -

# Check for ports in processed XML.
cat /var/lib/clickhouse/preprocessed_configs/config.xml |\
  grep mysql_port
cat /var/lib/clickhouse/preprocessed_configs/config.xml |\
  grep postgresql_port
```

Creating your own certificate authority

Options for ClickHouse server certificates



Ideal for external services or services with clients you don't control

Ideal for internal services where you control the clients and their operating environment

Useful for testing. **Never use this approach for real data.**

Setting up your own certificate authority

```
# Generate CA key & cert.
cd /etc/clickhouse-server/certs
openssl genrsa -out fortress_ca.key 2048
openssl req -new -x509 -subj "/CN=Fortress CA" \
 -days 3650 -key fortress_ca.key \
 -extensions v3_ca -out fortress_ca.crt
```

Create a server certificate and Diffie-Hellman params

```
# Generate server key and cert, sign with CA key.
openssl req -newkey rsa:2048 -nodes \
 -subj "/CN=fortress" \
 -keyout server.key -out server.csr
openssl x509 -req -in server.csr -out server.crt \
 -CAcreateserial -CA fortress_ca.crt \
 -CAkey fortress_ca.key -days 365

# Generate Diffie-Hellman params
openssl dhparam -out dhparam.pem 4096
```

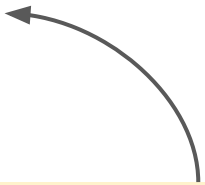
Two extra steps when running in a Docker container

```
# Ensure file permissions are set to ClickHouse user:group
sudo chown 101:101 /etc/clickhouse-server/certs/*

# Add fortress as a host name in /etc/hosts
vi /etc/hosts
...
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
...
172.17.0.10     a99dee369c69 fortress
:wq
```

Install the internal CA in the host certificate store

```
cp /etc/clickhouse-server/certs/fortress_ca.crt \
 /usr/local/share/ca-certificates
update-ca-certificates
```



Installs the certificate so SSL-enabled clients can see it automatically

Encrypting connections to ClickHouse

First, enable only encrypted ports

```
cat << EOF > data/conf.d/ssl.xml
<clickhouse>
    <!-- disable http and enable https port -->
    <http_port remove="true"/>
    <https_port>8443</https_port>

    <!-- disable tcp and enable secure tcp port -->
    <tcp_port remove="true"/>
    <tcp_port_secure>9440</tcp_port_secure>

    <!-- disable http and enable https inter-server port -->
    <interserver_http_port remove="true"/>
    <interserver_https_port>9010</interserver_https_port>
. . .
```

Second, configure the server settings for inbound TLS

```
<!-- configure OpenSSL -->
<openSSL>
    <server>
        <certificateFile>/etc/.../certs/server.crt</certificateFile>
        <privateKeyFile>/etc/.../certs/server.key</privateKeyFile>
        <caConfig>/etc/.../certs/fortress_ca.crt</caConfig>
        <loadDefaultCAFile>false</loadDefaultCAFile>
        <verificationMode>relaxed</verificationMode>
        <cacheSessions>true</cacheSessions>
        <disableProtocols>sslv2,sslv3</disableProtocols>
        <preferServerCiphers>true</preferServerCiphers>
        <invalidCertificateHandler>
            <name>RejectCertificateHandler</name>
        </invalidCertificateHandler>
    </server>
```

Third, configure outbound connections to other servers

```
        <client>
            <loadDefaultCAFile>false</loadDefaultCAFile>
            <caConfig>/etc/.../certs/fortress_ca.crt</caConfig>
            <certificateFile>/etc/.../certs/server.crt</certificateFile>
            <privateKeyFile>/etc/.../certs/server.key</privateKeyFile>
            <cacheSessions>true</cacheSessions>
            <disableProtocols>sslv2,sslv3</disableProtocols>
            <preferServerCiphers>true</preferServerCiphers>
            <verificationMode>strict</verificationMode>
            <invalidCertificateHandler>
                <name>RejectCertificateHandler</name>
            </invalidCertificateHandler>
        </client>
    </openSSL>
</clickhouse>
```

Now you can connect to ClickHouse with TLS!

```
clickhouse-client --host=fortress --password=WLNj00x/ --secure
ClickHouse client version 23.3.3.52 (official build).
Connecting to fortress:9440 as user default.
Connected to ClickHouse server version 23.3.3 revision 54462.

273ad4f0c468 :) quit
```

Enabling at rest encryption for ClickHouse data

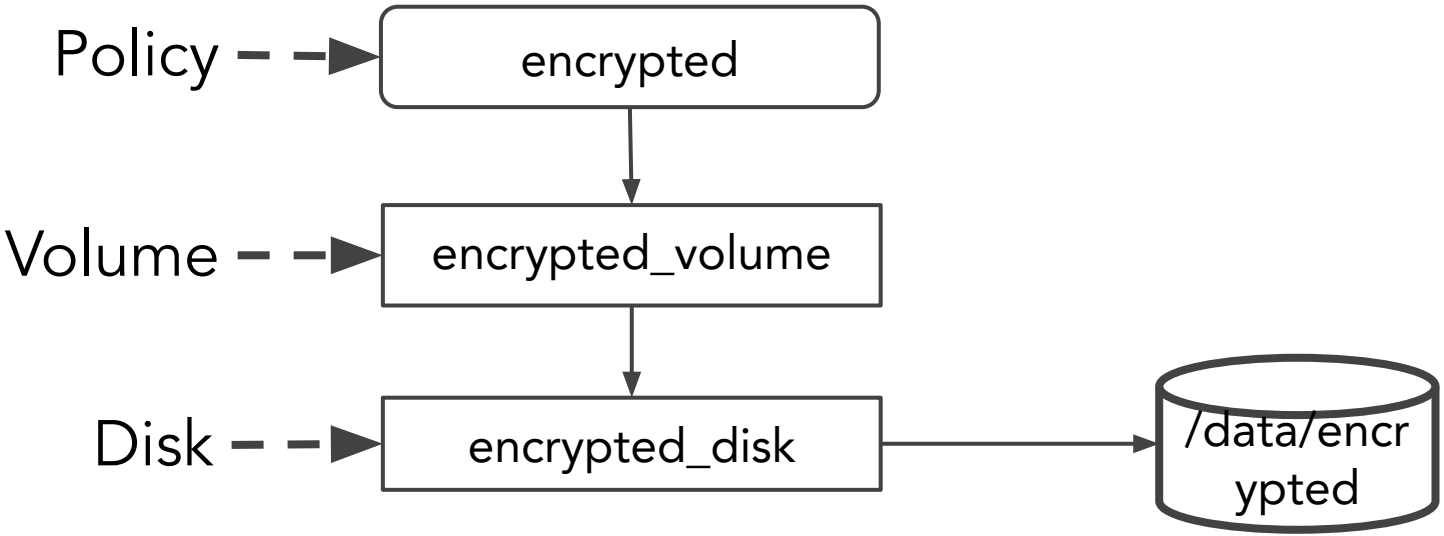
Encryption at rest, option 1: file system

File system encryption is the simplest path to global at-rest encryption.

Some of our favorite options:

1. LUKS (Linux Unified Key Setup) -- Encrypt local volume using DMCrypt
2. Cloud block storage encryption -- Use public cloud automatic encryption
    a. Example: Amazon EBS encryption
3. Kubernetes storage provider encryption -- Enable encryption in StorageClass if supported.
    a. Example: AWS EBS provider encrypted: “true” option

Option 2: Encrypted volume using storage policy



Definition of storage policy for encrypted disk

```
<clickhouse><storage_configuration>
        <disks>
            <disk1>
                <type>local</type>
                <path>/data/clickhouse_encrypted/</path>
            </disk1>
            <encrypted_disk>
                <type>encrypted</type><disk>disk1</disk>
                <path>encrypted/</path>
                ...
            </encrypted_disk>
        </disks>
        <policies>
            <encrypted><volumes><encrypted_volume>
                        <disk>encrypted_disk</disk>
            </encrypted_volume></volumes></encrypted>
    ...
```

Table with encrypted data

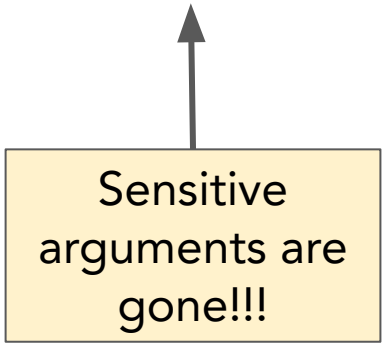
```
CREATE TABLE bench_encrypted
(
    `c_int` Int64,
    `c_str` varchar(255),
    `c_float` Float64
)
ENGINE = MergeTree
ORDER BY c_int
SETTINGS storage_policy = 'encrypted'
```

Protecting logs

Avoiding data leakage with query masking

ClickHouse log entry for a query:

```
2021.01.26 19:11:23.526691 [ 1652 ] {4e196dfa-dd65-4cba-983b-d6bb2c3df7c8}
<Debug> executeQuery: (from [::ffff:127.0.0.1]:54536, using production
parser) WITH unhex('658bb26de6f8a069a3520293a572078f') AS key SELECT
decrypt(???), key) AS plaintext
```



How to make queries “disappear” from logs

/etc/clickhouse-server/config.xml

```
<query_masking_rules>
    <rule>
        <name>hide encrypt/decrypt arguments</name>
        <regexp>
            ((?:aes_)?(?:encrypt|decrypt)(?:_mysql)?)\s*\(\s*(?:'(?:
        \\'|.)+'|.*?)\s*\)
         </regexp>
        <!-- or more secure, but also more invasive:
             (aes_\w+)\s*\(.*\)
        -->
        <replace>\1(???)</replace>
    </rule>
</query_masking_rules>
```

Final notes and more to come

Parting tips on host-level security

1. ClickHouse runs as clickhouse user (root access not required)
2. Protect directories containing data and credentials
    a. /etc/clickhouse-server -- Credentials
    b. /var/lib/clickhouse -- Data and (new!) credentials
    c. /var/log/clickhouse-server -- Logs, SQL queries may be exposed
3. Protect the network around ClickHouse
    a. Load balancers, firewalls
    b. Make server network non-routable to outsiders as far as possible

Areas for future presentations and work

- ClickHouse cluster security
- Hardening ClickHouse in cloud native environments (aka Kubernetes)
- FIPS-compatible ClickHouse builds for FedRAMP
- Protecting backups!

Making the Journey to FedRAMP: Cisco Umbrella, Altinity, and ClickHouse-based Analytics
Tuesday 20 June @ 10am PDT

https://altinity.com/making-the-journey-to-fedramp-cisco-umbrella-altinity-and-clickhouse-based-analytics

Background information

- ClickHouse Documentation – https://clickhouse.com/docs
- Altinity security documentation – https://docs.altinity.com
- Altinity Blog – https://altinity.com/blog
- Altinity Kubernetes Operator for ClickHouse on GitHub
  - Operator Hardening Guide
- Sample code
  - https://github.com/Altinity/clickhouse-sql-examples/tree/main/fortress-clickhouse


Thank you!
Any Questions?
Robert Hodges
https://altinity.com
Altinity.Cloud
FIPS-compatible
Altinity Stable Builds
Altinity Kubernetes
Operator for
ClickHouse