Securing Messaging, APIs, and Integrations

Mary Cochran
App Dev Solutions Architect
May 7, 2019
BIGGEST BLOCKERS TO TRANSFORMING
INCLUDE TALENT GAPS, SECURITY & TECHNICAL DEBT

What are the top blockers to your organization's digital transformation efforts? (Choose up to 3)

- IT Talent or Skills Gap: 26%
- Security Gaps or Risk: 24%
- Technical Debt: 24%
- Cost Reduction Mandates: 22%
- Changing Capabilities Required for IT Mgmt: 20%
- Status Quo / Inertia: 20%
- Manual Business Processes: 16%
- Manual IT Operations: 16%
- Lack of Analytics or Insights for Business: 16%

Source: Global IT Trends & Priorities Research, November 2018, Qualtrics and Red Hat (Over 1,052 valid respondents)
68% of breaches took months or longer to discover\textsuperscript{2}

99% of the vulnerabilities exploited by the end of 2020 will continue to be ones known by security and IT professionals at the time of the incident\textsuperscript{3}

2018 speech by David Hogue, a National Security Agency official, who said the NSA had not responded to an intrusion that exploited a zero-day vulnerability in over two years.

81% of hacking-related breaches leveraged either stolen and/or weak passwords\textsuperscript{1}

\textsuperscript{1}2017 Verizon Data Breach Investigations Report
\textsuperscript{2}2018 Verizon Data Breach Investigations Report
\textsuperscript{3}Gartner, “Focus on the Biggest Security Threats, Not the Most Publicized,” November, 2017
“The softest target in most organizations is the app layer and attackers know this. Microservices thus both make this problem harder and easier for the defenders”

Many separate APIs and ports per app == numerous doors for attackers

Source: B. Cameron Gain for thenewstack.io, Microservices Security: Probably Not What You Think It Is, Mar 2018
https://thenewstack.io/microservices-security-probably-not-what-you-think-it-is/
INTEGRATION IS UNDERGOING RAPID CHANGE

Traditional integration **incompatible** with hybrid cloud development

Modern architectures and app development requires more **agile solutions**

Microservices ♦ Scalability ♦ DevOps
Agile Teams ♦ Cloud App Dev

USE INTEGRATION WHERE NEEDED, RATHER THAN CENTRALIZING
RED HAT APPLICATION INTEGRATION

- Compose and integrate microservices across an enterprise service network
- Automate and optimize business processes

**Runtimes**
- Red Hat OpenShift Application Runtimes
- Red Hat Data Grid
- Red Hat AMQ Broker
- OpenJDK

**Integration**
- Red Hat Fuse
- Red Hat AMQ
- Red Hat 3Scale API Management

**Automation**
- Red Hat Process Automation Manager
- Red Hat Decision Manager

**Develop, Deploy and Manage Across Cloud and On Premise**

Integration with RH Developer, CI/CD tools & Security Services
Optimized for OpenShift & Kubernetes Services
AMQ SECURITY
RED HAT AMQ 7

Flexible, standards-based messaging for the enterprise, cloud and the Internet of Things

**Broker**
- Store & forward
- Volatile & durable
- Full JMS 2.0 support
- Best-in-class perf

**Interconnect**
- Message router
- High-performance direct messaging
- Distributed messaging backbone

**Streams**
- Durable pub/sub
- Replayable streams
- Based on Apache Kafka

**AMQ Online (Messaging-as-a-Service)**
- Scalable, easy-to-manage messaging utility based on OpenShift container platform
- Available for self-managed and Red Hat-managed deployments (AMQ Online)

**Standard protocols**

**Polyglot clients**

**Common management**
SECURE TRAFFIC BETWEEN ROUTERS

- Step 1: Get your keys and certificates, such as PKCS12 truststore file

```
openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 65000 -out cert.pem
openssl x509 -text -noout -in cert.pem
openssl pkcs12 -inkey key.pem -in cert.pem -export -out truststore.p12
openssl pkcs12 -in truststore.p12 -noout -info
```
SECURE TRAFFIC BETWEEN ROUTERS

- Step 1: Get your keys and certificates, such as PKCS12 truststore file
- Step 2: Add your sslProfile to both router configs

```plaintext
sslProfile {
  name: router-ssl
  certFile: /absolute/path/to/cert.pem
  keyFile:/absolute/path/to/key.pem
  password: password
}
```
SECURE TRAFFIC BETWEEN ROUTERS

- Step 1: Get your keys and certificates, such as PKCS12 truststore file
- Step 2: Add your sslProfile to both router configs
- Step 3: Add/Adjust inter-router listener on Router A

```yaml
listener {
  role: inter-router
  host: 0.0.0.0
  port: 10003
  saslMechanisms: ANONYMOUS
  sslProfile: router-ssl
  authenticatePeer: false
  requireSsl: true
}
```
SECURE TRAFFIC BETWEEN ROUTERS

- Step 1: Get your keys and certificates, such as PKCS12 truststore file
- Step 2: Add your sslProfile to both router configs
- Step 3: Add/Adjust inter-router listener on Router A
- Step 4: Add/Adjust connector on Router B used to connect to A

```
connector {
  role: inter-router
  host: 0.0.0.0
  port: 10003
  saslMechanisms: ANONYMOUS
  sslProfile: router-ssl
  verifyHostName: no
}
```
SECURE TRAFFIC FROM CLIENTS TO ROUTERS

- Step 1: Add a Client Listener using SSL Profile

```
listener {
  host: 0.0.0.0
  port: amqp
  saslMechanisms: ANONYMOUS
  authenticatePeer: no
  sslProfile: router-ssl
  requireSsl: true
}
```
SECURE TRAFFIC FROM CLIENTS TO ROUTERS

- Step 1: Add a Client Listener using SSL Profile
- Step 2: Adjust Client URL to use SSL

SSL URL:
amqps://localhost:5672?transport.verifyHost=false&transport.storeType=PKCS12&transport.trustStoreLocation=/absolute/path/to/certificate.p12&transport.trustStorePassword=password
SECURE TRAFFIC BETWEEN BROKERS

- Step 1: Gather your keys and certificates, such as PKCS12 truststore file

```bash
openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 65000 -out cert.pem
openssl x509 -text -noout -in cert.pem
openssl pkcs12 -inkey key.pem -in cert.pem -export -out truststore.p12
openssl pkcs12 -in truststore.p12 -noout -info
```
SECURE TRAFFIC BETWEEN BROKERS

- Step 1: Gather your keys and certificates, such as PKCS12 truststore file
- Step 2: Edit Connectors and Acceptors to enable SSL

```xml
<acceptors>
  <acceptor name="artemis">tcp://localhost:61616?sslEnabled=true;keyStorePath=truststore.p12;keyStorePassword=password;trustStorePath=truststore.p12;trustStorePassword=password</acceptor>
</acceptors>

<connectors>
  <connector name="my-connector">tcp://localhost:61616?sslEnabled=true;keyStorePath=truststore.p12;keyStorePassword=password;trustStorePath=truststore.p12;trustStorePassword=password</connector>
</connectors>
```
SECURE TRAFFIC BETWEEN BROKERS AND ROUTERS

- Step 1: Add SSL Profile if not already there to Router

```json
sslProfile {
  name: router-ssl
  certFile: /absolute/path/to/cert.pem
  keyFile:/absolute/path/to/key.pem
  password: password
}
```
SECURE TRAFFIC BETWEEN BROKERS AND ROUTERS

- Step 1: Add SSL Profile if not already there to Router
- Step 2: Adjust Broker Connector

```json
connector {
    name: broker1
    host: localhost
    port: 61616
    role: route-container
    saslMechanisms: ANONYMOUS
    sslProfile: router-ssl
    verifyHostName: no
}
```
ADDING USERS TO THE BROKER

```
./artemis user add

activemq {
  org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule required
    debug=false
    reload=true
  org.apache.activemq.jaas.properties.user="artemis-users.properties"
  org.apache.activemq.jaas.properties.role="artemis-roles.properties";
}

connector {
  name: broker1
  host: localhost
  port: 61616
  role: route-container
  saslUsername: secureUser
  saslPassword: securePassword
}

<security-enabled>true</security-enabled>
<security-settings>
  <security-setting match="#">
    <permission type="createNonDurableQueue" roles="secure-role"/>
    <permission type="deleteNonDurableQueue" roles="secure-role"/>
    <permission type="createDurableQueue" roles="secure-role"/>
    <permission type="deleteDurableQueue" roles="secure-role"/>
    <permission type="createAddress" roles="secure-role"/>
    <permission type="deleteAddress" roles="secure-role"/>
    <permission type="consume" roles="secure-role"/>
    <permission type="browse" roles="secure-role"/>
    <permission type="send" roles="secure-role"/>
    <!-- we need this otherwise ./artemis data imp wouldn't work -->
    <permission type="manage" roles="secure-role"/>
  </security-setting>
</security-settings>
```
SASL

saslpasswd2 -c -f SASL_DATABASE_NAME.sasldb -u DOMAIN_NAME USER_NAME

pwcheck_method: auxprop
auxprop_plugin: sasldb
sasldb_path: /etc/sasl2/qdrouteråd.sasldb
mech_list: ANONYMOUS DIGEST-MD5
EXTERNAL PLAIN

router {
  mode: interior
  id: Router.A
  saslConfigPath: /etc/sasl2
  saslConfigName: qdrouterd
}

listener {
  role: inter-router
  host: 0.0.0.0
  port: 10003
  saslMechanisms: PLAIN
  sslProfile: router-ssl
  authenticatePeer: yes
  requireSsl: true
}

connector {
  role: inter-router
  host: 0.0.0.0
  port: 10003
  saslMechanisms: PLAIN
  saslUsername: sasluser2@mary.example
  saslPassword: password
  sslProfile: router-ssl
  verifyHostName: no
}

jmsConnectionFactory.setUsername("sasluser2@mary.example");
jmsConnectionFactory.setPassword("password");

#redhat #rhsummit
activemq {
    org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule required
debug=true
    initialContextFactory=com.sun.jndi.ldap.LdapCtxFactory
    connectionURL="LDAP://localhost:389"
    connectionUsername="CN=Administrator,CN=Users,DC=example,DC=com"
    connectionPassword=redhat.123
    connectionProtocol=s
    authentication=simple
    userBase="dc=example,dc=com"
    userSearchMatching="(CN={0})"
    userSearchSubtree=true
    roleBase="dc=example,dc=com"
    roleName=cn
    roleSearchMatching="(member={0})"
    roleSearchSubtree=true
};
FUSE SECURITY
ENTERPRISE INTEGRATION PATTERNS
Build integrations using enterprise best practices.

200+ COMPONENTS
Batch, messaging, web services, cloud, APIs, and more ...

BUILT-IN DATA TRANSFORMATION
JSON, XML, HL7, YAML, SOAP, Java, CSV, and more ...

INTUITIVE ROUTING
Develop integrations quickly in Java or XML.

NATIVE REST SUPPORT
Create, connect, and compose APIs with ease.
CAMEL ROUTE

Consumer
- Consume requests
- Start of a route
- Dispatching outgoing replies

Processor
- Intermediate node in the pipeline
- Standard processors or customized ones

Producer
- Produce requests
- End of route
- Dispatching outgoing requests
DEPLOY AN HTTPS ENDPOINT

```java
private void configureSslForJetty() {
    KeyStoreParameters ksp = new KeyStoreParameters();
    ksp.setResource("c:\Projects\blah\fakefilter.jks");
    ksp.setPassword("123456");

    KeyManagersParameters kmp = new KeyManagersParameters();
    kmp.setKeyStore(ksp);
    kmp.setKeyPassword("export-password");

    SSLContextParameters scp = new SSLContextParameters();
    scp.setKeyManagers(kmp);

    JettyHttpComponent jettyComponent = getContext().getComponent("jetty", JettyHttpComponent.class);
    jettyComponent.setSslContextParameters(scp);
}
```
DEPLOY AN HTTPS ENDPOINT

```java
public class MyRouteBuilder extends RouteBuilder {
    public void configure() {
        configureSslForJetty();
        configureSslForHttp4();
        from("jetty:https://0.0.0.0:4443/topython/?matchOnUriPrefix=true")
            .to("https4://backend.fake.com:4444/?q=ssl&bridgeEndpoint=true
                &throwExceptionOnFailure=false");
    }
}
```
CALLING AN HTTPS ENDPOINT

```java
private void configureSslForHttp4() {
    KeyStoreParameters trust_ksp = new KeyStoreParameters();
    trust_ksp.setResource("c:\\Projects\\blah\\fakeca.jks");
    trust_ksp.setPassword("123456");
    TrustManagersParameters trustp = new TrustManagersParameters();
    trustp.setKeyStore(trust_ksp);
    SSLContextParameters scp = new SSLContextParameters();
    scp.setTrustManagers(trustp);
    HttpComponent httpComponent = getContext().getComponent("https4", HttpComponent.class);
    httpComponent.setSslContextParameters(scp);
}
```
CALLING AN HTTPS ENDPOINT

```java
public class MyRouteBuilder extends RouteBuilder {
    public void configure() {
        configureSslForJetty();
        configureSslForHttp4();
        from("jetty:https://0.0.0.0:4443/topython/?matchOnUriPrefix=true")
            .to("https4://backend.fake.com:4444/?q=ssl&bridgeEndpoint=true &throwExceptionOnFailure=false");
    }
}
```
3SCALE SECURITY
RED HAT 3SCALE API MANAGEMENT

Control
- Security
- Key management
- Rate limiting
- Policy enforcement
- App and user management
- Provisioning

Visibility
- Analytics
- App tracking
- User tracking
- Traffic alerts
- Developer support

Flexibility
- Distributed
- Multi-department
- Multi-environment
- Highly scalable
- Powerful APIs
- Webhooks
Allow/restrict access to your API endpoints along with rate limits

Package your APIs. Create access tiers. Set rate limits.

API services
- Endpoint A
- Endpoint B

Rate limits
- X Calls / Minute
- Y Calls / Day

Monetization
- Free
- $X per Month
- $Y per Call

Package #1
Internal Teams

Package #2
Strategic Partners

Package #3
Developers
YOUR API SECURITY

Authenticate and restrict access to your APIs. Protect backend services.

Multiple authentication mechanisms

- API Key
- App ID / App Key
- OpenID Connect

Authenticate traffic
Restrict by policy
Drop unwelcome calls
Protect backend services
Generate overage alerts
Impose rate limits
OPENSIFT
### AUTOMATED & INTEGRATED SECURITY

<table>
<thead>
<tr>
<th>CONTROL</th>
<th>DEFEND</th>
<th>EXTEND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Security</td>
<td>Infrastructure</td>
<td></td>
</tr>
<tr>
<td>Container Content</td>
<td>Container Host Multi-tenancy</td>
<td>Security Ecosystem</td>
</tr>
<tr>
<td>Container Registry</td>
<td>Network Isolation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Audit &amp; Logging</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FINAL TASK:

FILL IN EVALUATION FORM IN RED HAT SUMMIT APP, DO IT NOW!

Fill in: Securing Messaging, APIS, and Integrations

Mary Cochran
App Dev Solutions Architect
May 7, 2019
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/redhat