Notification Support Infrastructure for Self-Adapting Composite Services

Erlend Andreas Gjære, Per Håkon Meland and Thomas Vilarinho
SINTEF ICT, Software Engineering, Safety and Security
Trondheim, Norway
Email: {erlendandreas.gjare, per.h.meland, thomas.vilarinho}@sintef.no

***Abstract*—Building reliable software services based on service components supplied by partners and third parties in potentially complex chains of providers, is inherently challenging. In the case of cloud-based services, providers may offer not only software (web services) but also infrastructure (e.g., processing and storage). Making composite services trustworthy and reliable requires all parties to be open about changes that may impact their customers, and they must inevitably deal with a fluctuating threat picture. In this paper, we describe a publish/subscribe-based notification infrastructure which allows a Service Runtime Environment (SRE) to receive alerts about changes and threats in service components. Notifications arise from both human and automated monitoring, and are published to a notification broker which handles subscriptions and message distribution. The SRE is enabled to react to these notifications automatically through adaptation of the service composition, based on rules that can syntactically match the contents of received notifications. In addition to describing the technical implementation, we show an example of how a composite service from the Air Traffic Management (ATM) domain can instantly adapt itself when it receives a notification about a threatened service component. We also demonstrate how a mobile client app is used to keep humans aware of the notifications.**

Keywords–Security notifications; self-adaptability; accountability, composite services.

I. INTRODUCTION

With the emerging paradigm of composite services, i.e., software services composed of functionality provided by several services components—changes or attacks on a service component implies an attack or change to the composite service as a whole. Service compositions in general are highly distributed and have a complex nature. They expose a greater attack surface than traditional stand-alone systems, and introduce multiple layers of policies, which may not be compatible. Web services and cloud-based software components may be offered by third party parties and potentially in long provider chains, and can be used in compositions along with other services. In the age of cloud computing, the importance of breach notifications is emphasised due to the need for transparency throughout entire chains of service providers—seeking trustworthiness through accountability [1].

To make these kinds of services reliable enough, we need a holistic approach to how they are built in terms of security and accountability. The entire line-up of involved service components and providers must be considered in terms of how situations need to be mediated and mitigated where (unwanted) changes or threats occur [2]. Simply having control over which relevant third parties (and fourth parties, etc.) our provider deals with is of course a place to start. Preventive measures in this context today are to a large extent being based on risk assessments, contractual relationships and audits. Reactive measures, on the other hand, include notifications via email, phone, remote log file-streams and dashboards for mediating significant changes and security incidents. The contents of these are however not as predictable as needed for triggering corrective activities *automatically* in composite services. Preparing for "failure" is equally needed to prepare a composite service for reacting this way.

To be able to respond to changes in an effective and truly timely manner, we need notifications that are machine-readable and syntactically clear in such a way that they can be used to trigger automatic adaptation of a composite service. This, however, requires a higher level of refinement than traditional log streams provide, and either automated or human processing of low-level input may be needed in advance to infer (and mediate) useful correlations between several more atomic log events. At the same time, notifications being sent if only a change or threat is actually encountered, could allow organisations to exchange more information without revealing "too much". We therefore seek a notification infrastructure where service providers report in a convenient and standardised way on behalf of themselves, and are equally able to receive notifications from the others corresponding to which service components are being used at any time.

There are strong regulatory reasons why such a notification infrastructure needs to be realised. The European Union implemented a breach notification law in the Directive on Privacy and Electronic Communications (ePrivacy Directive) in 2009 [3]. In this directive, it is stated under article 4, paragraph 2 that *"In case of a particular risk of a breach of the security of the network, the provider of a publicly available electronic communications service must inform the subscribers concerning such risk and, where the risk lies outside the scope of the measures to be taken by the service provider, of any possible remedies, including an indication of the likely costs involved."* Similarly, security breach notification laws have been enacted in most U.S. states since 2002 (the only states with no law are currently Alabama, New Mexico and South Dakota) [4].

Our main goal with this paper is to describe a notification infrastructure that can support automatic application of corrective measures to service compositions at runtime. The approach includes a publish-subscribe based infrastructure for message delivery, along with a way to prepare rules at design-time for enabling automated response at runtime. We demonstrate our

 ISBN: 978-1-61208-378-0

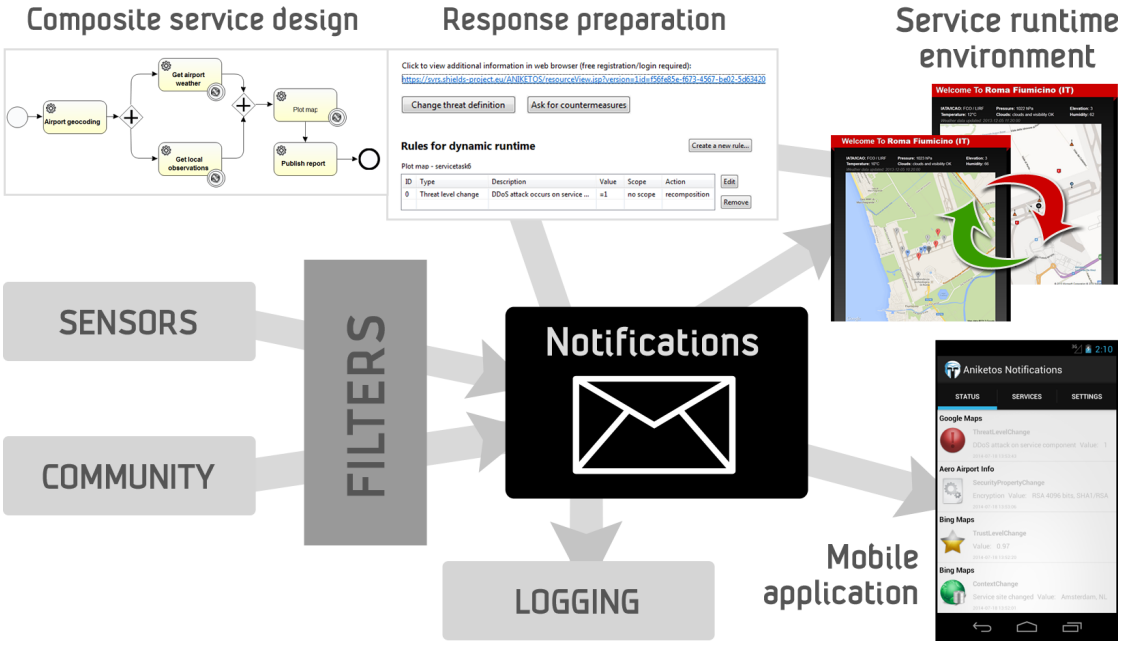

Figure 1: An outline of how composite services can be prepared and deployed ready for automatic adaptation, supported by an infrastructure for security notifications.

infrastructure by allowing a composite service in our Service Runtime Environment (SRE) to dynamically substitute one of its components without downtime, upon receiving notification about a threat. We also describe our accompanying mobile client app, which receives the same notification and can show them to a layperson, not necessarily a security expert, but for instance someone who has some kind of responsibility for the composite service.

Our work complements similar approaches from the past, such as Zheng and Lyu [5], who propose a user-collaborative failure data sharing mechanism for service-oriented systems. What separates our work from the rest is the focus on security violations and threats, as opposed to reliability in general.

This paper is structured as follows: Section II presents our approach at a high level including a user scenario that describes how to prepare for escalations, and what the output might be. Technical details of our implementation work are provided in Section III. Section IV discusses various technical design choices, shortcomings and challenges, related and future work, before Section V concludes the paper.

II. Approach

Figure 1 shows how our notification infrastructure can be used to support automatic adaptation of composite services. In this section, we illustrate the approach with a practical use case example from the Air Traffic Management (ATM) domain: The pilot of an aircraft uses our composite service to retrieve up-to-date information about the airport to where the flight is destined. The airport report is built from data pulled simultaneously from several web services and service providers. It includes both geographical information, the current weather, a few local observations related to, e.g., oil or water on the runway, and finally, a map onto which this information is plotted.

A. Notifications

At the core we have the notification messages, which tie everything together. These are designed in particular to facilitate real-time communication of security and change events within distributed systems (of systems), especially between service providers and across geographical boundaries. We do not claim it to be a perfect design—it is a prototype—but we do have a version, which supports automated adaptation of composite services based on a number of security requirements [6].

Each notification message needs to identify the resource it concerns. Since composite services deal with web services, and all such services each have their own public endpoint Uniform Resource Locator (URL), we use this URL as the unique service ID. The service ID is needed to create subscriptions and to match received notifications with local rules in the SRE. The notifications are further typed into a specific category, i.e., one of those specified in Table I (first column). Some of the notifications types have support for a few sub-types in a hierarchy below. These sub-types may be essential to provide an appropriate response to the received notification, for instance where different sub-types of threats require different countermeasures, and commonly agreed values here are needed. There is also a value field required for each notification. The value is needed to specify the actual status of the notified type (and sub-type), e.g., if a threat is on the rise or if it has passed, which is expressed with probability as a float value between 0 and 1. The value can also be a simple string, or even a Javascript Object Notation (JSON) [7] object (a human-readable format for transmitting data). Self-adaptation may be based on the value, so there is a need to

 ISBN: 978-1-61208-378-0

TABLE I: NOTIFICATION TYPES AND CONTENT SPECIFICATION

ThreatLevelChange	Supports several sub-types of threat families, e.g., *DDoS attack*, *Service injection*, *Vulnerable crypto library*, *Trust poisoning*, etc. Value specifies the probability of a threat being encountered (between 0 and 1). Also possible to specify a threat ID from a common threat repository for linking to a very specific threat (and more info on-line).
SecurityPropertyChange	Supports sub-types with various security properties to detail how a service is implemented in terms of security, e.g., *Encryption*, *Backup cycle*, etc. Implementation details are to be provided in the value field, such as length of backup cycle.
ContextChange	Supports sub-types describing the context of the service, e.g., *Server site location*, *Backup location*, etc. In these particular cases, a location name (e.g., country) would go into the value field.
ContractViolation	Supports sub-types for violated properties in cases where a violation is detected during service contract matching with consumer policy, e.g., *Separation of duty*, *Binding of duty*, etc.
ServiceChange	Intended for composite services to provide an update if a recomposition has been made and hence a service component has been substituted. Sub-types such as *Component added* and *Component removed* can be used, or even *Service not provided* if a service is taken (temporarily) down due to a vulnerable component that could not be replaced.
TrustLevelChange	Specifies a level of trustworthiness (between 0 and 1). The values need to be published by an authoritative source for trust ratings (not self-declared).

standardise a format to use for each individual notification type (and sub-type) here.

B. Composite Service Design and Response Preparation

To create composite services we use the Business Process Model and Notation (BPMN) [8] and a modified graphical tool [6] to model the service specification as a process consisting of BPMN service tasks. In our example shown in Figure 2, visible as boundary error intermediate events [9], we have modelled a threat *DDoS-attack occurs on service component* due to the importance of high availability. The Distributed Denial of Service (DDoS) threat can in this case be both monitored and dealt with quite easily, so we want to be notified whenever an attack occurs. Instead of just allowing the attacked component to bring down our entire composition, we need to prepare the composite service to switch automatically to another map component not under attack.

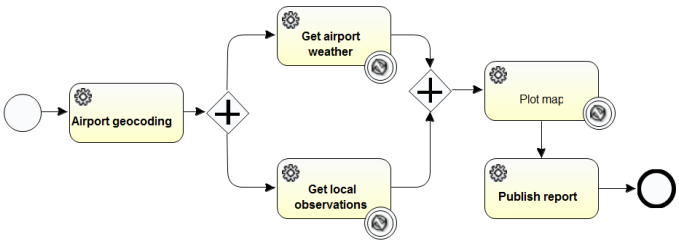

Figure 2: A composite service process is modelled in BPMN, with triggering events for notification on changes and threats.

Based on the triggering events modelled in the composite service definition, rules can be defined in the model to address the scenarios we are able to foresee. It is the service tasks, not the actual web service components, which are targeted in the scope of these rules. This makes the rules independent of the actual service component implementations we choose to deploy, and we do not need to define individual rule-sets for each potential composition plan.

For each rule, one must define the same properties as a notification would be expected to carry to get a match. This includes setting the notification type, sub-type and/or a value with some comparison of either equal to (also used for string comparison), larger than (or equal) and smaller than (or equal). In the scope section of the rule editor, it is further possible to define where in the process the rule shall apply. If a particular threat or change occurs for a component, we might not need to perform reactive measures unless it happens before, after or during the execution of a particular task in our process. We also have an option to launch an additional service process, e.g., one that might initiate hardening of other parts of the system, and/or send notification messages to clients and/or service technicians. In the end, we might end up with a list of several rules for several service tasks, or simply one for the one we have defined in our case study, as shown in Figure 3. Recent work by Salnitri et al. [10] goes into more details on rule definition and execution.

To perform a recomposition in our use case, we need an alternative service specification ready where another functionally equal service component is used to realise the map plotting task. We define one composition plan where the report generation is done based on the map service from Google, and we define another where the same responsibility is served by Bing. One of these plans will be then deployed as default. When there is a threat notification and the other composition plan is not affected by it, then this plan is considered more appropriate and can be deployed immediately in place of the attacked one. It is, however, not always the case that we have two candidate components for the same service task. One might also set the rule to simply stop providing the entire composite service, e.g., if a non-replaceable component has changed its policy and becomes no longer compliant.

C. Filtered Input From Sensors and Community

In order to generate notifications in the first place, there must obviously be some kind of monitoring in place for service components. In the case of cyber-threats, some service providers have dedicated security operations to monitor and process low-level input from e.g., syslog. Security Information and Event Management (SIEM) systems are used to gather and correlate security events from multiple logging sources both in real time and aggregated from previous events, and are sometimes purchased as a service from third party providers. This, however, does not normally involve logs from other organisations, except when changes are already publicly disclosed. Some monitoring efforts are also performed on a national level or sometimes by joint efforts across, e.g., an entire business sector, such as the Norwegian financial sector cybercrime unit [11]. Such intelligence could be a valuable source of input for an even wider community in the global service market. Apart from trustworthiness ratings, which need to come from an independent source, it should at least be possible to publish alerts for services provided by oneself.

 ISBN: 978-1-61208-378-0

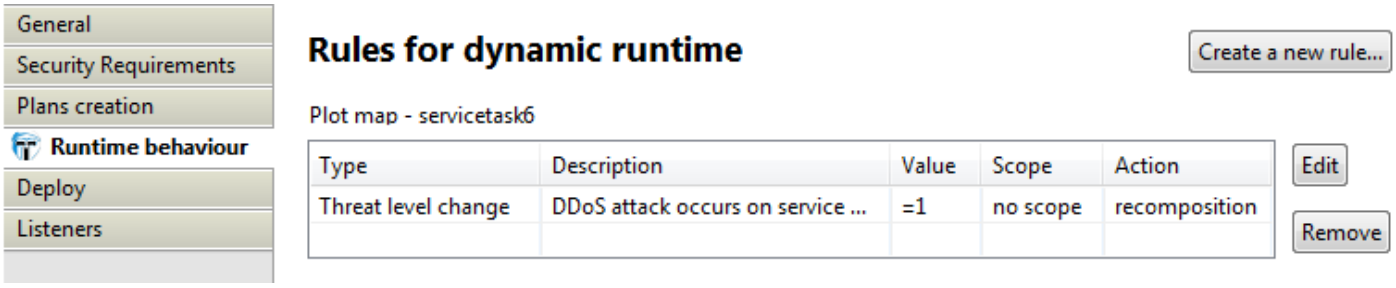

Figure 3: A rule has been defined for responding to a DDoS-attack on the map service.

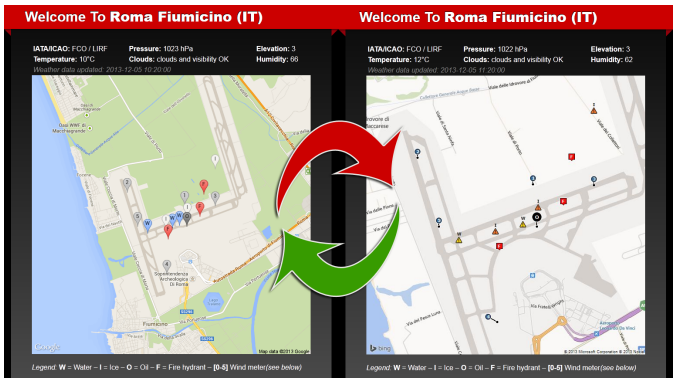

Figure 4: The SRE has replaced the Google map service with Bing Maps through a recomposition.

In the security community, there are also people who for instance create scanner bots, which are used to check the Internet for particular vulnerabilities, such as in the case of the *Heartbleed* bug [12]. A threat "*Vulnerable crypto library*" could then be warned against with the notification infrastructure, both when it is publicly disclosed and detected on a particular server, as well as immediately when the server has been patched (with a 0 value). In the particular case of Heartbleed, it would also be possible to add a threat ID referring to, e.g., the Common Vulnerability and Exposures (CVE) database [13] (in this case CVE-2014-0160), which would provide a pointer to more specific information. Assurance from the security community would however be necessary to validate the notifications, or else it would be easy to inject malicious notifications and cause severe business disruption for service providers who are shut out.

Regardless of input source, notifications are collected through the common web service interface provided for publishers. For SIEM or Security-as-a-Service (SaaS) products, it is easy to add support for invoking this notification service by an operator. We have implemented a web interface to be available for manual reporting, and have integrated other independent monitoring platforms, such as a threat monitoring module [14], which is able to infer the required level of semantics and trigger appropriate notifications. As long as the format can match what the SRE is able to interpret from the notifications, the rules are in practice agnostic to monitoring implementations.

D. Notification Receivers

In order to trigger events in the self-configuring service process, the SRE must be able to receive such notifications and align these with the previously defined and deployed rules. Since a threat monitor has now purportedly detected that the original map service used in our case study is hit by a DDoS-attack, the SRE is notified about this through subscriptions based on the deployed service composition. As it can find a match with the rule we previously defined on DDoS-attacks, the SRE initiates the specified action according to that rule, which is here to try a recomposition.

Since we had support for two different map services providing the same functionality, we have in practice prepared an additional composition plan for the airport report composite service. When the notification concerning a DDoS-attack on the map service is triggered and received by the SRE, a service verification mechanism [15] can tell us that the first plan no longer satisfies our security requirements. The rule in Figure 3 will initiate a recomposition accordingly. The original composition plan with the DDoS-ed map service will be ignored, and the second plan becomes the top-ranked. The recomposition proceeds with deploying the second composition plan, containing the alternative map service instead of the original one. Nevertheless, the same level of functionality is provided, as illustrated with Figure 4 where the airport reports, before (to the left) and after recomposition, are lined up next to each other.

While monitoring and responding to changes and threats in real time is our main goal, we also need to store the notifications for future reference. A common repository for historical security and service change notifications is valuable for doing research on previous events. There are of course questions to answer in this respect, e.g., how the repository should be managed and to what level of explicitness historical data should be made accessible over time. A repository could provide insights on questions like, e.g., cascading events between service providers, temporal aspects of incident/response, etc. Issues that must be dealt with in this context, as well as for the industry as a whole, is about the fear of negative publicity, customer repercussions and lost revenue caused by being "too" open. Instead of being seen as accountable, some may see service providers that do their job on reporting incidents as having inferior security.

 ISBN: 978-1-61208-378-0

TABLE II: MESSAGING PATTERNS APPLIED IN OUR IMPLEMENTATION

Event message	Notification messages are used to transmit events from one service to another. These notifications are reliable and asynchronous.
Publish-Subscribe Channel	The provider of the notification sends the event on a publish-subscribe channel, which delivers a copy of a particular event to each receiver. The receivers must have expressed an interest in such event on beforehand.
Data-Type Channel	The different notification types enables the receivers to easily interpret the incoming events.
Guaranteed Delivery	The notification infrastructure uses a store and forward mechanism to ensure message durability.
Message Bus	Independent systems and services of various types are able to communicate in a loosely coupled fashion through the use of the message bus pattern.
Event-Driven Consumer	The receving parties automatically consume messages as they become available.

III. Implementation Details

To operate a self-adapting distributed service infrastructure at runtime, a supporting system for Machine-to-Machine (M2M) messaging across networks was needed. Since there may be an endless number of services and SREs utilising this infrastructure, we cannot provide all messages to everyone. A publish-subscribe channel pattern was found suitable for this, since each service provider already needs to define which service components are to be used in a service composition. Appropriate subscriptions can be derived automatically from these service specifications. Registering the subscriptions should be handled by the SRE, which will then receive notifications only about threats for relevant services.

In addition to publish-subscribe, we have designed the notification infrastructure according to a number of messaging patterns in service-oriented architectures. Table II gives an overview of these, based on Chatterjee [16].

A. Notification Message Broker

The entire infrastructure relies on the delivery of notifications to subscribers. In a publish-subscribe architecture, this is the responsibility of the message broker. In addition to dispatching the messages and providing an endpoint for subscriptions, the broker will receive notifications to publish from authorised publishers.

Our implementation builds on Apache ActiveMQ [17] in this role, utilising the de-facto standard Java Message Service (JMS) [18] specification to provide compatibility with many platforms and messaging protocols. A broad variety of wire level protocols are supported, meaning that brokers can be connected to clients built with any programming language or platform with a compatibility for just one of these protocols. The performance of ActiveMQ brokers can be scaled up horizontally by configuring several instances in a network of brokers, if needed. Our broker is in addition deployed on a cloud-based infrastructure, for further increase in scalability.

The notifications are organized in a hierarchy of JMS topics and sub-topics, in practice building a single subscription string. The first part of the subscription string contains the service ID. Then, below that topic there are different sub-topics mapping to the different security notification types, each part of the subscription string separated with a dot '.'. With this hierarchy, one could decide to subscribe to all notifications from a service or particular types or sub-types within that service. If no type or sub-type is specified, the wild-card '>' will match all topics from that character and onwards in the subscription string. In this way, topics for each of the notification types can be created dynamically, without needing to explicitly subscribe to all of them individually, and without risking topics that no-one subscribes to. Hence, new sub-topics can be added and those subscribing to the main topic will start receiving notifications from the new sub-topic as well.

In addition, the entire hierarchy has common root nodes where one can set the access control level for all notifications below them. With that in mind, we created an entirely public channel ("pub") that anyone can subscribe to anonymously, but not publish through. For publishing notifications, the publishing client needs to authenticate to the broker. Apache ActiveMQ authentication/authorization configuration capabilities offer the possibility of implementing authentication able to access/subscribe to notifications from another channel, e.g., a service level that for instance would require formal contracts to be established between participating parties in advance. An ActiveMQ JMS topic corresponding to a service notification will finally have the format *pub.[serviceName].[notificationType].[subType]*. So, for example, the topic *pub.http://demo-aniketoswp6.rhcloud.com/googlemap/service.ThreatLevelChange.DDoS attack on service component* would map into:

- **Channel**: Public
- **Service ID**: http://demo-aniketoswp6.rhcloud.com/googlemap/service
- **Notification Type**: ThreatLevelChange
- **Sub-Type**: DDoS attack occurs on service component

In our implementation, we have granted subscription and notification retrieval access to all clients (anonymous access), and publish access only to authorized users. However, this set-up could be configured differently on the broker, allowing specific settings per topic or subtopic. For the authentication regarding the authorized access, we have used a simple XML configuration file, mainly because it was just a prototype implementation. In a real deployment, one could have relied on ActiveMQ support for Java Authentication and Authorization Service (JAAS) [19] or LDAP. When it comes to encryption of the messages, we have been using Transport Layer Security (TLS) encryption towards both publishers and subscribers. However, due to a limitation of the Android mobile client library used for the wire protocol, we had to support messaging in clear text towards the mobile client app.

B. Service Runtime Environment

The SRE is responsible for executing the composite service at runtime. We have implemented ours using the Activiti Engine [20], which can take BPMN composition plans as input, as they have been built with our graphical modelling tool. In the runtime engine, all BPMN is converted to executable code, based on the standard and without further manual work.

For the SRE, we have a plug-in [6] that is able to connect to the notification broker and create subscriptions

 ISBN: 978-1-61208-378-0

based on the rules attached to a service deployment. Whenever a composition plan is deployed, it includes all service IDs needed to create the subscriptions. For each service ID in the composition, there is a corresponding JMS topic being an individual channel of notifications, which a client can subscribe to. Accordingly, when subscribing to notifications for several services, there must be registered individual subscriptions for each of these services. The granularity of the rules will determine the relevance in cases of, e.g., slight variations in threat levels (not all changes will actually trigger an action).

The plug-in is a Java client implementation using the ActiveMQ library to connect to the broker. When subscriptions are made, the plug-in starts to listen for incoming notifications according to the made subscriptions. When a notification arrives, it compares it with all active rules for a potential match. The rules are simple XML-representations of what was described at design-time, having elements for each of the notification properties to match, as well as an element controlling scope and one controlling action(s) to perform.

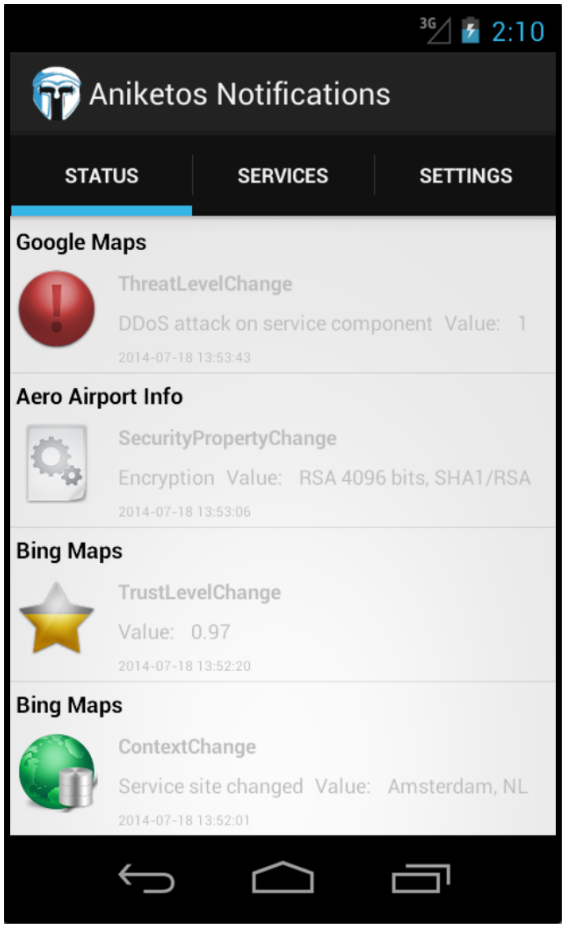

Figure 5: Screenshot of the Android client app for receiving notifications on-the-go.

C. Android Client App

The client app is perhaps not a critical part of the infrastructure, but might indeed be a useful one. While we have a reference implementation available for a regular Java-based subscription client—and any client can subscribe to the topics using any one of the protocols supported by the ActiveMQ broker—this does not fit very well with the usage scenario of receiving updates at *any* time. A mobile client app, on the other hand, can always be brought along by the person(s) responsible for monitoring a service, listening silently in the background and then notifying when something happens.

A working implementation of our client app for the Android mobile operating system has been made. Its functionality is currently rather simple, just enabling users to subscribe (and unsubscribe) to services, and show a feed of the updates. The app does not support choosing the exact type or subtype of notification of each service to subscribe to, but this is a limitation set to simplify the graphical user interface. The notification feed is a vertical time-line where new notifications are placed on top, much like in social media. Notifications are filtered on either service alone (by tapping from the unfiltered list, which is seen in Figure 5) or both service and notification type together (by tapping from the service filtered list). The app allows the broker address to be customised.

In terms of connecting to and communicating with the ActiveMQ broker, our app utilises the binary MQ Telemetry Transport (MQTT) protocol [21]. MQTT is designed with a simple API, a fixed header of just 2 bytes length and a light keep-alive mechanism. MQTT is with its very small overhead especially suitable for small-footprint devices, as witnessed by its use in Facebook's mobile messenger application [22]. MQTT has support for publish-subscribe, and its community develops client libraries for several platforms. When our notification app was implemented, the most popular MQTT client library was the Java-based *MQTT-Client* library from Fusesource [23]. This library does however not support TLS, but since integration worked out-of-the box, we decided to use it for our initial implementation. The MQTT connection is handled by a service, which maintains the connection even if the app is not running in the foreground.

The app creates subscription strings for topics in the broker based on service ID, and receives notifications as soon as they are published. The performance can however depend on the available data connection speed, as the case always is for mobile devices. If the mobile client has been offline, any notifications that have been published during this time-frame will be received immediately upon re-connection. This is achieved with durable subscriptions (using a customisable time-out). The notifications are delivered according to the quality of service specified when either establishing the connection or publishing the message, depending on the protocol used. In order to identify anonymous subscribers between sessions, the app identifies itself with the unique device ID provided by the operating system, each time it connects to the broker.

IV. Discussion

The challenges of the notification infrastructure presented in this paper are very much similar to the ones ENISA [24] has identified for the European Union (EU) with data breach notification requirements for the electronic communications sector, in the ePrivacy Directive [3]. The bullets below explain our view on how these are or should be handled:

 ISBN: 978-1-61208-378-0

- *Risk prioritisation: The seriousness of a breach should determine the level of response, and breaches should be categorised according to specific risk levels.* With our solution, the first prioritisation is done by the receiver when he determines which notifications he wants to subscribe to—as is the nature of a publish-subscribe architecture. Secondly, there are dedicated notifications for changes in threat level, and the receiver can specify threshold values for when corrective actions should be applied. If the receiver has missed out on subscribing to certain threats these notifications will never be received. This is a weakness in cases where entirely new threats appear.
- *Communication channels: Operators want assurances that notification requirements will not negatively impact their brands.* Here, the objectives should be to prevent tampering of messages and ensure that false breach reports are not submitted. The current implementation of our notification infrastructure have several weaknesses, e.g., there is only a weak authentication scheme for the notification source, and notifications are not signed. However, the authentication scheme could be improved by configuring LDAP or JAAS on the ActiveMQ broker, and the encryption of the messages exchanged with the mobile client could be achieved by using a MQTT client library with TLS support, e.g., IBM's MQTT SDK [25]. A comprehensive list of requirements for a secure logging infrastructure has further been described by the Common Event Expression (CEE) project [26], which should be taken into account in future work.
- *Resources: Budgetary allocations for regulatory authorities should reflect new regulatory responsibilities.* The notification infrastructure itself is cloud deployable and can be scaled up and down according to needs. The cost of brokering notifications is very low. If the reporting from service providers is handled automatically by sensors, this cost is negligible as well, while involving manual human labour increases cost. These costs could in turn be reduced with participation from Computer Security Incident Response Teams (CIRTs), collaborating across sectors or even across borders, such as within the EU.
- *Enforcement: Data controllers will be less incentivised to comply with regulations if regulatory authorities do not have sufficient sanctioning powers.* In addition to regulatory authorities, sending out breach notifications should be motivated by contractual terms between the service provider and consumer, audits from an independent third party, as well as a genuine desire to achieve trust by being open. A late disclosure of incidents will in many cases damage the reputation of a service provider more than the incident itself.
- *Undue delay in reporting: Regulatory authorities want to see a short deadline for reporting breaches to authorities and data subjects [..] Service providers, however, want their resources to be focused on identifying if the problem is serious and solving the problem, instead of spending time reporting details, often prematurely, to regulatory authorities*. Our notifications are primarily short messages that can be distributed rapidly. Due to the nature of publish-subscribe, messages will be delivered as soon as they are published and the subscriber is online. Such messages takes little time and effort to create, especially if done automatically. Therefore, delays are not considered a major obstacle. Within the EU, telecommunications operators and ISP providers must inform national authorities within 24 hours after breach detection with at least an initial set of information, with more details to follow within three days [27].
- *Content of notifications: Operators want to make sure that the content of the notifications does not impact negatively on customer relations. Regulatory authorities, however, want to see that the notifications provide the necessary information and guidance in line with the rights of the data subjects.* As stated above, our messages as short, early warnings that do not contain much information by themselves. More detailed content can be sent out at later stage through other means. Since we can send notifications wrapped entirely as JSON objects, we are also able to extend the value of the notification messages as a JSON object with additional properties to enable customisation as needed. A potential threat to all notification systems is related to fake notification and manipulation of reputation based systems. We refer to one of our previously published paper for a deeper discussion on this [28].

A few related efforts have been made on standardising security event message content and formats, but we are unaware of efforts with the purpose of supporting automated service composition. While our work started in the experimental end with self-adaptable service compositions in mind, standardisation is needed at some point. There will definitively be potential to learn from similar initiatives, however at the time of writing there seems to be little activity in the area. The already mentioned CEE project [26] was for instance initiated to standardise event descriptions to support auditing and users' ability to comprehend event log and audit data. CEE defines both delivery methods and filtering, as well as an event structure—although in a flat manner. CEE is extensible in a way that can redefine any part of the taxonomy, although that might not be a good thing for a publish-subscribe infrastructure. The project has however been shut down due to a lack of financial support. The Distributed Audit Services (XDAS) specification [29] was similarly created to support the principle of accountability and detection of security policy violations in distributed systems. XDAS defines a taxonomy of events categories (layers) comprising varying levels of semantics and context, but is very focussed on recording events for correlating audit trails. The specification is hence largely targeted at auditing and compliance, and has not become a formally approved standard. Work on XDAS v2 appears to have been initiated in recent years, but without publishing any significant progress.

Having a notification infrastructure by itself is obviously of little value if there are neither agents creating notifications nor anyone receiving them. Together with academia and industry (18 partners in total), we have evaluated the API,

integration and performance with others tools for monitoring and adaptation service-oriented systems. Results from this evaluation shows high levels of satisfaction for installation, documentation, integration and stability [30]. Our focus for future work evolves around integration with new tools and securing the infrastructure itself.

V. Conclusion

We have demonstrated a distributed notification infrastructure that facilitates runtime management and adaptation of service compositions. Though a service component may be regarded as reliable and secure enough when the composite service is designed, its security and privacy properties and attributes for quality of service can change during its life-time. In addition, risks are not static, and threats and vulnerabilities in service components can impact the overall security level of a composite service. Due to regulatory pressure and a need for trustworthiness through accountability in service composition chains, we believe that the concept of a common notification infrastructure is needed. Further work is however needed in terms of securing the infrastructure, research is needed on how it could be managed in practice, and standardisation work is needed to agree on notification content descriptions.

Acknowledgement

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grants no 257930 (Aniketos), 317631 (OPTET) and 371550 (A4Cloud). The authors are not affiliated with any of the service providers referenced in the use case, and the events described are purely hypothetical.

References

[1] S. Pearson et al., "Accountability for cloud and other future Internet services," 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, Dec 2012, pp. 629–632.

[2] E. A. Gjære and P. H. Meland, "Threats management throughout the software service life-cycle," EPTCS, vol. 148, 2014, p. 114.

[3] European Commission, "Directive on privacy and electronic communications," Tech. Rep., 2002.

[4] National Conference of State Legislatures. Security breach notification laws. [Online]. Available: http://www.ncsl.org/research/telecommunications-and-information-technology/security-breach-notification-laws.aspx [retrieved: September, 2014]

[5] Z. Zheng and M. R. Lyu, "Collaborative reliability prediction of service-oriented systems," in Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 35–44. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806809

[6] Aniketos project. AniketosEU on GitHub. [Online]. Available: https://github.com/AniketosEU [retrieved: September, 2014]

[7] ECMA International, ECMA-404 The JSON Data Interchange Standard, Std., October 2013.

[8] Object Management Group, Business Process Model and Notation (BPMN) Version 2.0, Std., January 2011.

[9] P. H. Meland and E. A. Gjære, "Threat Representation Methods for Composite Service Process Models," International Journal of Secure Software Engineering, vol. 4, no. 2, 2013, pp. 1–18.

[10] M. Salnitri, E. Paja, and P. Giorgini, "Preserving compliance with security requirements in socio-technical systems," in Proceeding of Cyber Security and Privacy (CSP) forum 2014, 2014.

[11] FinansCERT. Norwegian financial sector cybercrime unit. [Online]. Available: http://www.finanscert.no/engelsk.html [retrieved: September, 2014]

[12] Riku, Antti, Matti and Neel Mehta. Heartbleed bug. [Online]. Available: http://heartbleed.com/ [retrieved: September, 2014]

[13] MITRE Corporation. Common vulnerabilities and exposures (cve) database. [Online]. Available: http://cve.mitre.com/ [retrieved: September, 2014]

[14] Aniketos project, "Deliverable D4.3 Algorithms for responding to changes and threats," Tech. Rep., August 2013.

[15] B. Zhou et al., "Secure service composition adaptation based on simulated annealing," in 6th Layered Assurance Workshop, 2012, p. 49.

[16] S. Chatterjee. Messaging Patterns in Service-Oriented Architecture, Part 1. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa480027.aspx [retrieved: September, 2014]

[17] The Apache Software Foundation. Apache ActiveMQ website. [Online]. Available: http://activemq.apache.org/ [retrieved: December, 2014]

[18] Oracle, Java Message Service Specification, Std., November 1999.

[19] Oracle. Java Authentication and Authorization Service (JAAS) Reference Guide. [Online]. Available: http://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/JAASRefGuide.html [retrieved: September, 2014]

[20] Activiti. Activiti bpm platform website. [Online]. Available: http://www.activiti.org/ [retrieved: September, 2014]

[21] IBM and Eurotech, MQTT V3.1 Protocol Specification, Std., July 2014. [Online]. Available: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

[22] L. Zhang. Building Facebook Messenger. [Online]. Available: https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920 [retrieved: September, 2014]

[23] Fusesource. MQTT-Client An Open Source Java MQTT v3.1 Client. [Online]. Available: http://mqtt-client.fusesource.org/index.html [retrieved: September, 2014]

[24] S. Górniak et al., "Data breach notifications in the eu," ENISA, Tech. Rep., 2011.

[25] IBM Corporation. Getting started with the MQTT client for Java on Android. [Online]. Available: http://www-01.ibm.com/support/knowledgecenter/SS9D84_1.0.0/com.ibm.mm.tc.doc/tc10130_.htm [retrieved: September, 2014]

[26] MITRE Corporation, "CEE Log Transport (CLT) Specification," Tech. Rep., 2012. [Online]. Available: https://cee.mitre.org/language/1.0-beta1/clt.html

[27] European Commission. Digital Agenda: New specific rules for consumers when telecoms personal data is lost or stolen in EU. [Online]. Available: http://europa.eu/rapid/press-release_IP-13-591_en.htm [retrieved: September, 2014]

[28] P. H. Meland, "Service injection: A threat to self-managed complex systems," in Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on, Dec 2011, pp. 1–6.

[29] The Open Group, Distributed Audit Services (XDAS), Std., January 1997.

[30] Aniketos project, "Deliverable D7.3 Results of the final validation and evaluation of the ANIKETOS platform," Tech. Rep., May 2014.

 ISBN: 978-1-61208-378-0