AWSLang: Probabilistic Threat Modelling of the Amazon Web Services environment

Amandeep Singh Virdi
(virdi@kth.se)
foreseeti / KTH / NTNU

Introduction

- Attack simulations provide a viable means to test the cyber security of a system.
- One common approach to implement such simulations is the use of attack graph, which trace the various dependencies of every step and their connection to one another in a formal way.
- To further facilitate attack simulations and to reduce the effort of creating new attack graphs for each system of a given type, domain-specific languages are employed.
- This thesis report presents **AWSLang**, which can be used to design IT system models in context to the AWS (Amazon Web Services) environment and analyse their weaknesses.

Background

- **Threat modelling** involves understanding the complexity of the system and identifying all possible threats to the system, regardless of whether they can be exploited or not.
- Threat modelling looks at the system from an adversary's perspective to help designers anticipate attack goals and determine answers to questions about *what* the system is designed to protect, and from *whom*.
- Employing **attack graphs** during the threat modelling procedure offers a lot more advantages than other available methods to model a system.
- Generally, the production and analysis of attack graphs employ three steps: modelling the system; using **attack steps** to build a connected graph; analysis of the newly-constructed graph.

Background

- To facilitate threat modelling especially with respect to the construction of attack graphs and their analysis, **probabilistic relational model** has been proposed.
- It specifies how a graphical representation of probabilistic dependencies between variables should be constructed from a model that instantiates a class diagram (**metamodel**), such as the one of UML (Unified Modelling Language).
- Effectively, probabilistic relational threat modelling makes use of **Bayesian networks** to allow for attack simulations.
- With a system architecture model (metamodel) as the foundations, **attack simulations** allow for the execution of numerous parallel virtual penetration tests.

Meta Attack Language (MAL)

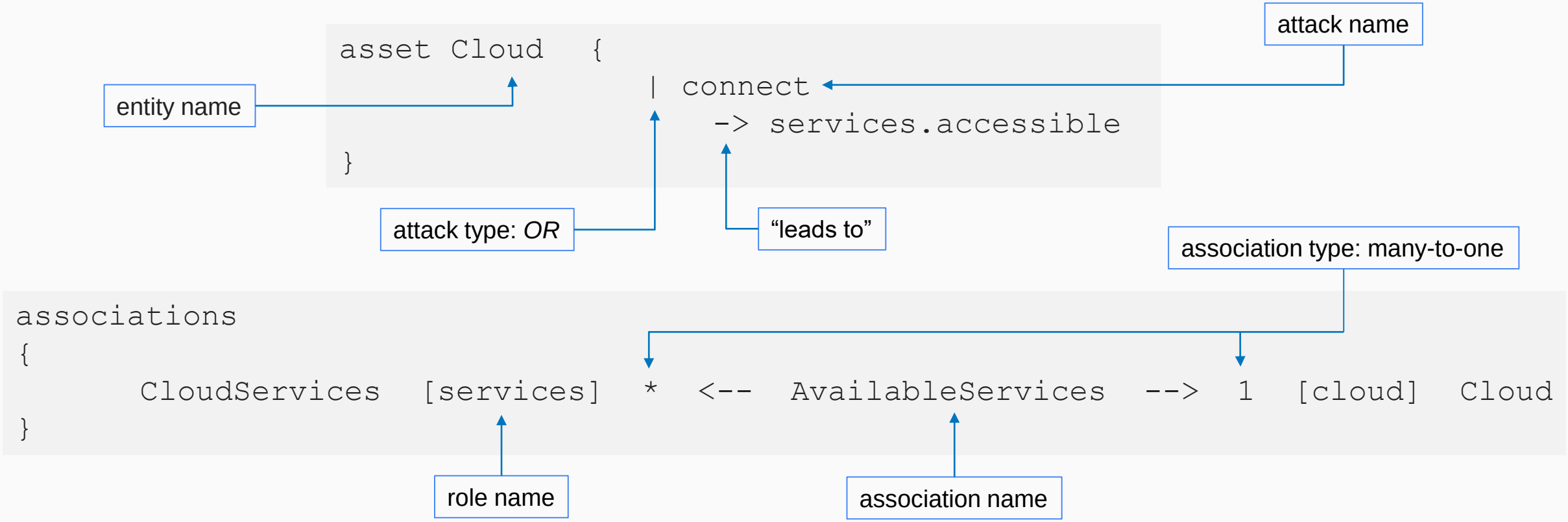
- Proposed by Johnson et al, MAL is a meta language – and not a domain-specific language – that facilitates attack simulations by defining what information is needed to model a particular system and specifying the generic attack logic.
- Classes containing attack steps constitute the core entities of a MAL specification. For example, `Cloud` can be an entity in a MAL specification, initialized as `cloud`.
- Furthermore, the entities of MAL are related to each other. For example, the class `Cloud` can be associated with class `CloudServices` (initialized as `services`) to refer to the services offered by the cloud.
- Attack steps are connected to each other, and thus a successful compromise of one step leads to the second step.

Meta Attack Language (MAL)

```
asset Cloud   {
                    | connect
                        -> services.accessible
}
```

```
associations
{
      CloudServices  [services]  *  <--  AvailableServices  -->  1  [cloud]  Cloud
}
```

Meta Attack Language (MAL)



Meta Attack Language (MAL)

- MAL features inheritance in a manner comparable to other object-oriented languages.

```
abstractAsset CloudServices {
                                | accessible
                                    -> requestData
                                & requestData
}

asset VirtualMachine extends CloudServices {
                                                | accessKey
                                                    -> requestData
                                                & requestData
}
```

Meta Attack Language (MAL)

- Some steps may be accomplished without effort. However, for to compromise some entity there is a fixed amount of time required.

  For example, if the time to crack the access key of the virtual machine takes a mean of 12 hours and a standard deviation of 5 hours, it be specified by a Gamma distribution as follows:

```
asset VirtualMachine {
                        & crackAccessKey [GammaDistribution(24, 0.5)]
}
```

Amazon Web Services (AWS)

- Like most cloud service providers, AWS provides on-demand cloud computing platform and utilities to individuals and enterprises (both in the private and public sector), on a paid subscription basis.
- The AWS Cloud Infrastructure is spread over many geographical regions. In AWS terminology, these are called Regions and Availability Zones (AZs).
- AWS provides services in 19 (and expanding) categories, ranging from dynamic compute capacity to storage to virtual networks, etc.

Compute
Amazon EC2
Amazon EC2 Auto Scaling
Amazon Elastic Container Service
Amazon Elastic Container Service for Kubernetes
Amazon Elastic Container Registry
Amazon Lightsail
AWS Batch
AWS Elastic Beanstalk
AWS Fargate
AWS Lambda
AWS Serverless Application Repository
Elastic Load Balancing
VMware Cloud on AWS

Storage
Amazon Simple Storage Service (S3)
Amazon Elastic Block Store (EBS)
Amazon Elastic File System (EFS)
Amazon Glacier
AWS Storage Gateway
AWS Snowball
AWS Snowball Edge
AWS Snowmobile

Database
Amazon Aurora
Amazon RDS
Amazon DynamoDB
Amazon ElastiCache
Amazon Redshift
Amazon Neptune
AWS Database Migration Service

Software
AWS Marketplace

Migration
AWS Migration Hub
AWS Application Discovery Service
AWS Database Migration Service
AWS Server Migration Service
AWS Snowball
AWS Snowball Edge
AWS Snowmobile

Networking & Content Delivery
Amazon VPC
Amazon VPC PrivateLink
Amazon CloudFront
Amazon Route 53
Amazon API Gateway
AWS Direct Connect
Elastic Load Balancing

Developer Tools
AWS CodeStar
AWS CodeCommit
AWS CodeBuild
AWS CodeDeploy
AWS CodePipeline
AWS Cloud9
AWS X-Ray
AWS Tools & SDKs

Media Services
Amazon Elastic Transcoder
Amazon Kinesis Video Streams
AWS Elemental MediaConvert
AWS Elemental MediaLive
AWS Elemental MediaPackage
AWS Elemental MediaStore
AWS Elemental MediaTailor

Management Tools
Amazon CloudWatch
AWS Auto Scaling
AWS CloudFormation
AWS CloudTrail
AWS Config
AWS OpsWorks
AWS Service Catalog
AWS Systems Manager
AWS Trusted Advisor
AWS Personal Health Dashboard
AWS Command Line Interface
AWS Management Console
AWS Managed Services

Machine Learning
Amazon SageMaker
Amazon Comprehend
Amazon Lex
Amazon Polly
Amazon Rekognition
Amazon Machine Learning
Amazon Translate
Amazon Transcribe
AWS DeepLens
AWS Deep Learning AMIs
Apache MXNet on AWS
TensorFlow on AWS

AWS Cost Management
AWS Cost Explorer
AWS Budgets
Reserved Instance Reporting
AWS Cost and Usage Report

Analytics
Amazon Athena
Amazon EMR
Amazon CloudSearch
Amazon Elasticsearch Service
Amazon Kinesis
Amazon Redshift
Amazon QuickSight
AWS Data Pipeline
AWS Glue

Security, Identity & Compliance
AWS Identity and Access Management (IAM)
Amazon Cloud Directory
Amazon Cognito
Amazon GuardDuty
Amazon Inspector
Amazon Macie
AWS Certificate Manager
AWS CloudHSM
AWS Directory Service
AWS Firewall Manager
AWS Key Management Service
AWS Organizations
AWS Secrets Manager
AWS Single Sign-On
AWS Shield
AWS WAF
AWS Artifact

Mobile Services
AWS Mobile Hub
Amazon API Gateway
Amazon Pinpoint
AWS AppSync
AWS Device Farm
AWS Mobile SDK

AR & VR
Amazon Sumerian

Application Integration
Amazon MQ
Amazon Simple Queue Service (SQS)
Amazon Simple Notification Service (SNS)
AWS AppSync
AWS Step Functions

Customer Engagement
Amazon Connect
Amazon Pinpoint
Amazon Simple Email Service (SES)

Business Productivity
Alexa for Business
Amazon Chime
Amazon WorkDocs
Amazon WorkMail

Desktop & App Streaming
Amazon WorkSpaces
Amazon AppStream 2.0

Internet of Things
AWS IoT Core
Amazon FreeRTOS
AWS Greengrass
AWS IoT 1-Click
AWS IoT Analytics
AWS IoT Button
AWS IoT Device Defender
AWS IoT Device Management

Game Development
Amazon GameLift
Amazon Lumberyard

Scope and Delimitations of the project

Elastic Compute Cloud (EC2)

Simple Storage Service (S3)

Virtual Private Cloud (VPC)

Identity and Access Management (IAM)

Methodology

Using the Design Science Research Process Model (DSR Cycle) to provide the overall structure to the project, the following objectives were identified:

Perform a **Domain Survey**

A Systematic Literature Review (SLR) of the AWS domain was conducted.

Construct a **Feature Matrix**

It maps the elements, or “assets,” in coreLang with the assets in AWSLang.

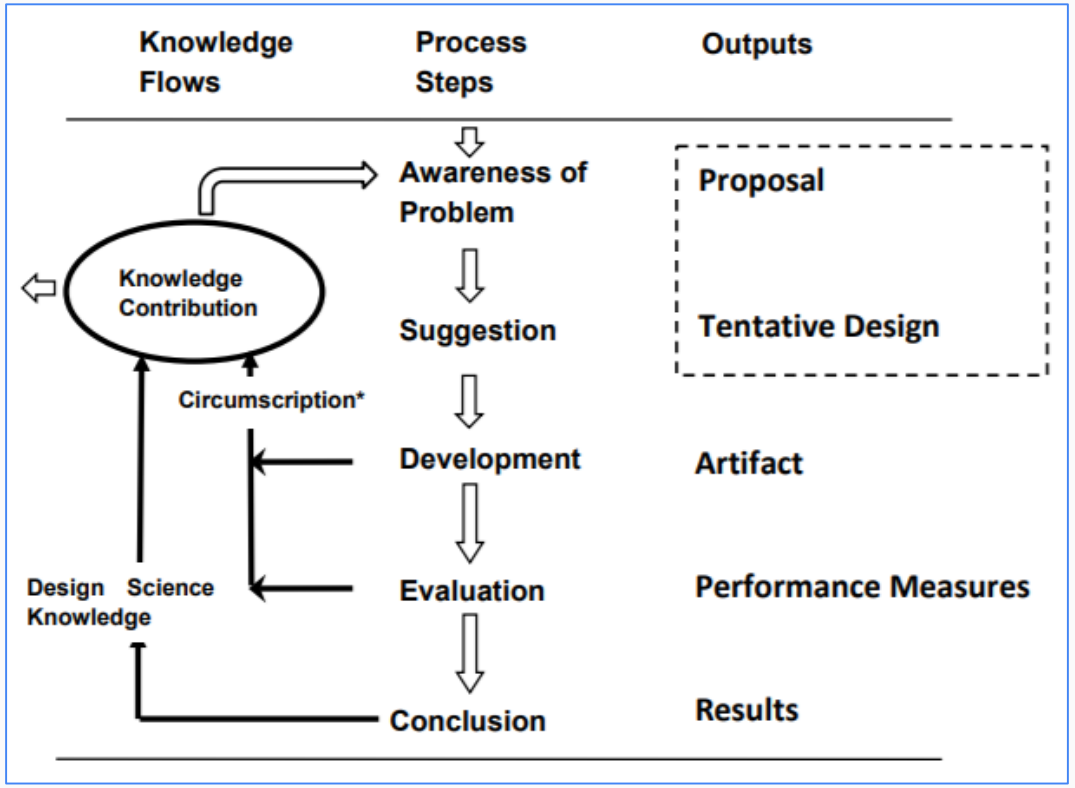


Image taken from “A Design Science Research Methodology for Information Systems Research”

Methodology

Build a **MAL specification**

AWSLang was written using simple text editors, such as the Notepad++ tool, using prevalent Java syntax; and complied via a custom compiler based off the ANTLR framework.

Write **Test cases**

Written in Java using the Eclipse editor, two categories of test cases were written: Unit tests, and Use Case tests.

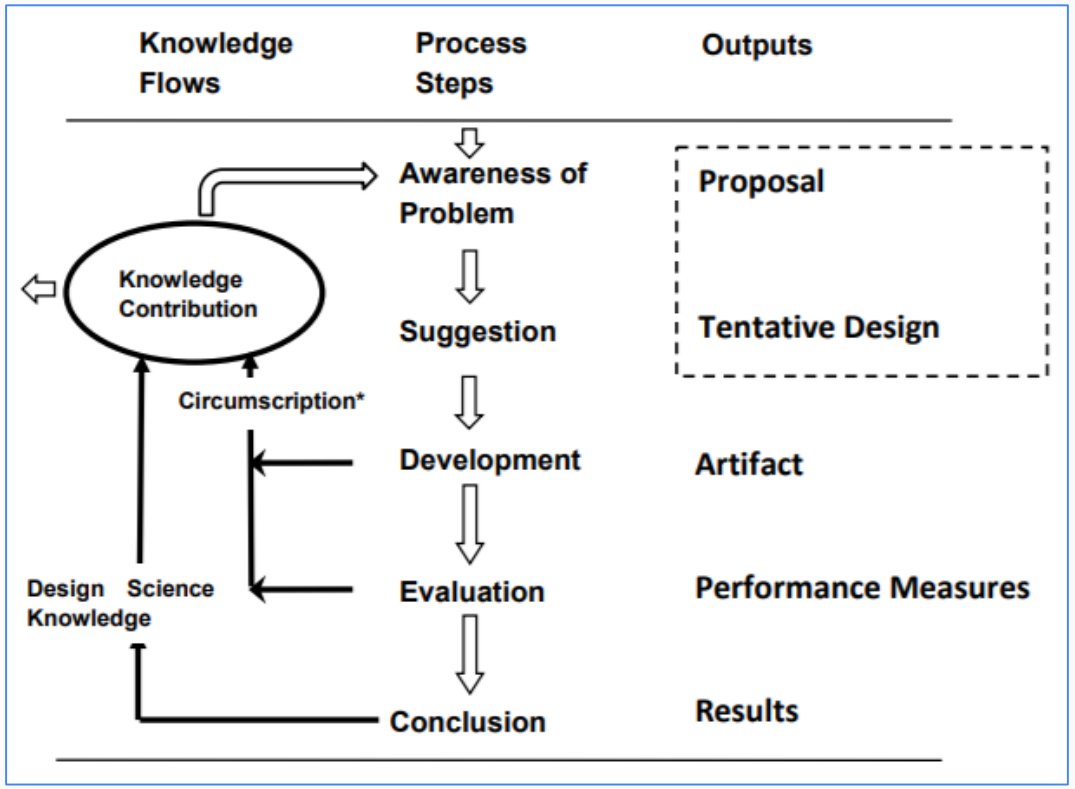


Image taken from “A Design Science Research Methodology for Information Systems Research”

Domain Survey: Amazon EC2

- Amazon Elastic Compute Cloud provides scalable computing capacity in the Amazon Web Services cloud.
- The virtual computing environments offered by the EC2 service are called as ***instances***.
- These instances are grounded on various configuration of CPU, memory, storage, and networking capacity for the instances, known as the *instance types*.
- While the instance type essentially determines the hardware of the host computer, the *Amazon Machine Image (AMI)* is a template that contains the software configuration (such as an **operating system**, an application server, and other **applications**) for the instance.

Domain Survey: Amazon S3

- Amazon Simple Storage Service is an object storage service designed to offer high availability and fast retrieval at high speed for any amount of data.
- Data in Amazon S3 is stored as *objects* within ***buckets***.
- The *object key* uniquely identifies the object in the bucket. In most cases, the file names act as the object keys.
- *Object metadata* is a set of name-value pairs that need to be set at the time the object is uploaded into the bucket.
- Amazon S3 supports *subresources* to store and manage the bucket configuration information using the Amazon S3 API or via the web console.

Domain Survey: Amazon VPC

- Amazon Virtual Private Cloud enables subscribers to launch AWS resources into a virtual network that they have defined.
- Within each VPC, subscribers can create, connect, and launch individual ***subnets***.
- Each subnet must be associated with a *route table*, which specifies the allowed routes for outbound traffic and inbound traffic for the subnet.
- ***Security groups*** control inbound and outbound traffic for the resource within the VPC.

Domain Survey: Amazon IAM

- The AWS Identity and Access Management is a service offered by AWS to securely control access to the resource within its cloud environment.
- An ***IAM user*** is a unique identity recognized by AWS services and applications.
- An ***IAM group*** is a collection of IAM users. It lets subscribers specify permissions for multiple users, which can make it easier to manage the permissions for those users.
- An ***IAM role*** is an IAM entity that defines a set of permissions for making AWS service requests. IAM roles are not associated with a specific user or group. Instead, trusted entities assume roles.

coreLang

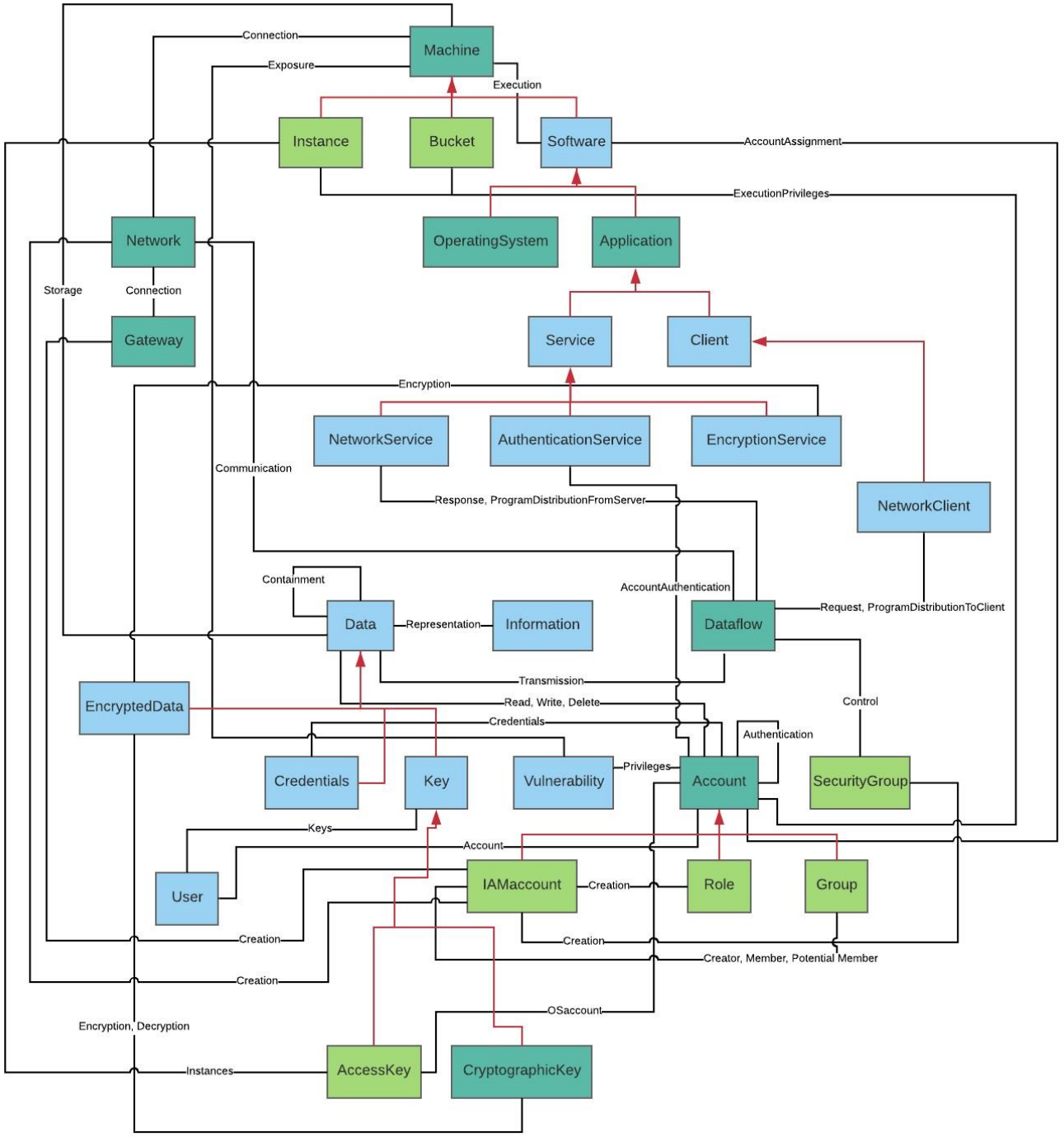
- The core language on which AWSLang is based on mainly consists of standard IT entities useful for representing traditional networks and similar models.
- The major elements of coreLang are `Machine`, `Account`, `Vulnerability`, `Network`, `Data`, `Dataflow`, and `User`.
- Each `User` has an `Account` on a `Machine` that they use to request `Data` stored on it. `Machine`s communicate to each other via `Dataflow`s over a `Network`.
- Some of the other elements in coreLang are: `AuthMachine`, `VulnMachine`, `SoftMachine`, `Software`, `Product`, `Service`, `Client`, `AuthenticationService`, `NetworkService`, `NetworkClient`, `Router`, `Information`, `AuthData`, `CoreEncryptedData`, `CryptographicKey`, etc.

Feature Matrix

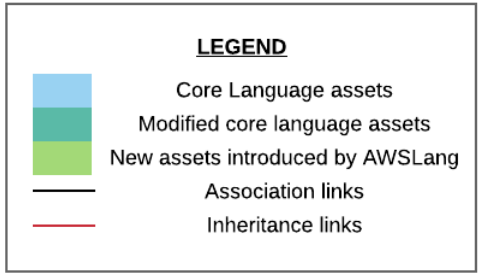
- The main advantage of constructing such a feature matrix is that it prevents repetition of work.
- There are assets in AWSLang that have been adapted off coreLang, albeit with a name change. This is different from the “Adopted from coreLang with modifications” column in the Feature Matrix. That column is used to identify only those assets from coreLang whose internal logic needed to be modified for use in AWSLang.

Assets in AWSLang	Adapted from coreLang without change	Adopted from coreLang with modifications	Not used in coreLang / New asset created for AWSLang
AccessKey			X
Account		X	
Application		X	
AuthenticationService	X		
Bucket			X
Client	X		
Credentials	X		
CryptographicKey	X		
Data	X		
Dataflow		X	
EncryptedData	X		
EncryptionService		X	
Gateway		X	
Group			X
IAMaccount			X
Information	X		
Instance			X
Key	X		
Machine	X		
Network		X	
NetworkClient	X		
NetworkService	X		
OperatingSystem		X	
Role			X
SecurityGroup			X
Service	X		
Software	X		
User	X		
Vulnerability	X		

MAL specification



AWSLang extends the core by introducing elements such as `IAMaccount`, `Role`, `Group`, etc., that are needed in order to model the AWS environment.



AWSLang: A brief introduction

```
asset Instance extends Machine    {
:
       | keyaccess
              ->     attemptConnectBasicAWSProtection,
                     attemptConnectAdvancedAWSProtection

       & attemptConnectBasicAWSProtection
              ->     authenticate

       | attemptConnectAdvancedAWSProtection [ExponentialDistribution(6.0)]
              ->     authenticate

       # advancedAWSProtection
              ->     attemptConnectBasicAWSProtection
:
}
```

AWSLang: A brief introduction

```
asset AccessKey extends Key  {
:
       | modifyKeyFile
              ->     _compromise

       & _compromise [ExponentialDistribution(6.0)]
              ->     compromise

       | compromise
              ->     assignedInstances.keyaccess,
                     assignedOSaccount.authenticate
:
}
```

AWSLang: A brief introduction

```
asset Bucket extends Machine {
:
        | attemptConnectPublicBucket
                ->      bruteForceAttack

        & bruteForceAttack [ExponentialDistribution(3.0)]
                ->      data.requestAccess

        # privateBucket
                ->      bruteForceAttack
:
}
```

AWSLang: A brief introduction

```
asset Application extends Software {
:
        | access
                ->      attemptAccessNoFirewall,
                        attemptAccessWithFirewall

        & attemptAccessNoFirewall
                ->      _machineAccess

        | attemptAccessWithFirewall [ExponentialDistribution(3.0)]
                ->      _machineAccess

        # firewallProtection
                ->      attemptAccessNoFirewall
:
}
```

Evaluation

- For validating AWSLang, two different categories of testing were applied:

  Unit tests, to ensure that each individual asset in AWSLang behaves like it is expected to.

  Use case tests, that rely on the compiled attack list and validate that assets in AWSLang interact with each as they are expected to.

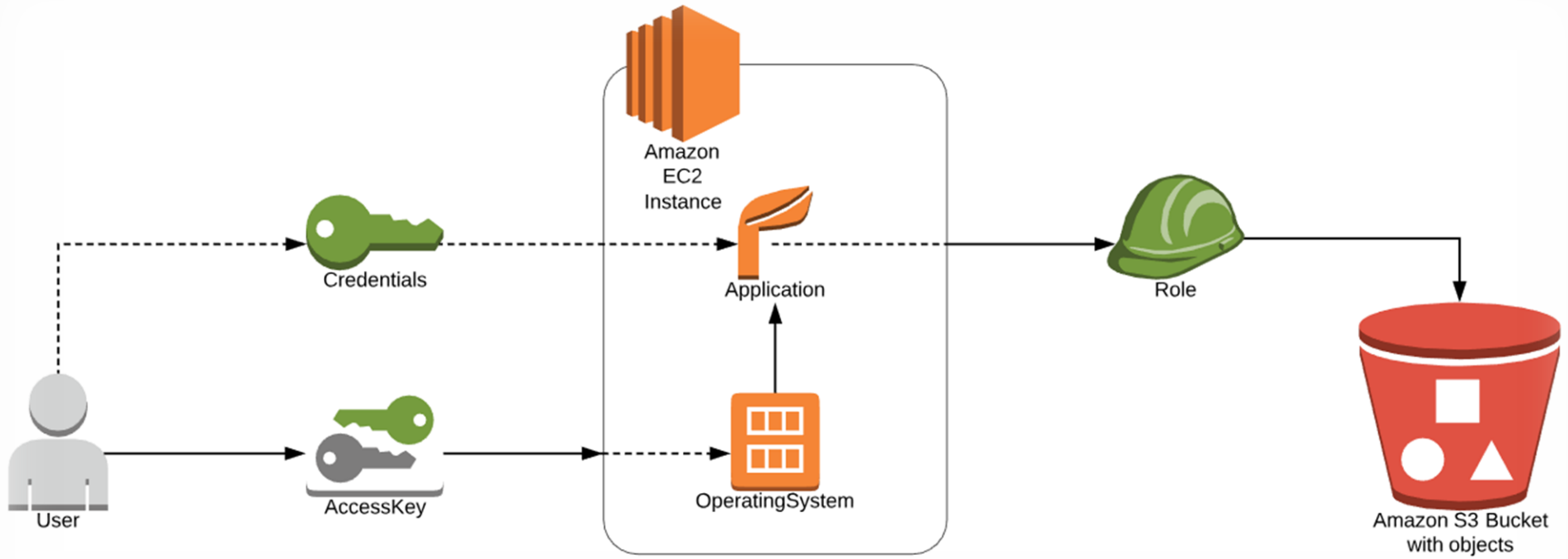
- Additionally, AWSLang was cross checked by two developers, each of whom are also working on a realization of MAL and is therefore familiar with the intricacies of the language. Furthermore, this also helped uncover any readability or user-friendliness issues due to the lack of documentation that the language might suffer from.

Unit tests sample: `IAMaccountTest`

```
Attacker attacker = new Attacker();
attacker.addAttackPoint(iamacct.compromise);
attacker.attack();

iamacct.compromisedAccess.assertCompromisedInstantaneously();
instance.compromisedAccess.assertCompromisedInstantaneously();
instance.denialOfService.assertCompromisedInstantaneously();
bucket.compromisedAccess.assertCompromisedInstantaneously();
bucket.access.assertCompromisedInstantaneously();
group.compromisedAccess.assertCompromisedInstantaneously();
role.compromisedAccess.assertCompromisedInstantaneously();
network.compromisedAccess.assertCompromisedInstantaneously();
gateway.compromisedAccess.assertCompromisedInstantaneously();
secGrp.compromisedAccess.assertCompromisedInstantaneously();
```

Use case tests sample: `TestUseCaseRoles_Instances`



Conclusion and Future Work

- The exponential growth in the usage of cloud service providers and the increasing reliance of IT-systems to their ubiquitous nature in addition to the capability that they provide make it paramount to assess their security, especially as these security and privacy concerns continue to grow.
- AWSLang will foster security analysts in the AWS cloud domain to model their cloud systems and to focus on analysing possible weaknesses.
- Although AWSLang is its current form is able to model the most-widely used AWS services and their interaction, further work remains.

Thank you!