Coap Anomaly Detection

Problem Description

Contents

Smart home networks with Internet of Things (IoT) devices can be a target of cyber attacks against IoT communication since many IoT protocols are implemented without authentication or encryption. Attackers can intercept IoT communication and also manipulate with IoT devices by sending spoofed commands to IoT sensors or controllers. This case study aims to demonstrate the security issues of IoT environment and to show how anomaly behavior can be detected by application of AI methods.

Input Data

[edit]

The sample data set represents monitoring and control data transmission between a client and a server in the test-bed network. The CoAP server is implemented on an IoT device (sensor, actuator, etc.) where it measures physical quantity like temperature and humidity or where it controls motion, light, smoke, etc. For our purposes, we will monitor CoAP communication and extract L7 (application layer) data like a code of the command, token, message ID, and URI. This dataset serves as an input for anomaly-based analysis of security incidents. Apart from regular communication datasets, students are also given samples of selected networks attacks, e.g., an unauthorized resource access, denial of service, etc. The dataset used in this case study consists of a collection of capture files as listed in Table bellow. Files idle.cap, regular.cap and observe.cap contain normal communication that will be used for learning the communication profile. File attack.cap contains samples of anomaly communication for testing the anomaly detection method.

File	Packets	Flows	Resources	Normal
idle.cap	25 096	716	5	yes
regular.cap	54 634	1 307	12	yes
observe.cap	17 480	415	8	yes
attack.cap	38 474	870	8	no

All files can be downloaded from: https://github.com/rysavy-ondrej/Ironstone/tree/master/Methods/Detection/CoapProfiling/SampleData

Method

[edit]

A simple anomaly detection method combining approaches by Manikopoulos and Crotti is used to create a statistical model of a communication profile and detect deviations.

The method has two stages. In the learning mode, the method uses input data to learn the normal profile for the system. In the discrimination mode, the learned method is applied to unknown data in order to classify the communication. An overview of the method is as follows:

Learning

[edit]

The learner observes a typical CoAP communication and builds profiles of communicated devices. The profile corresponds to a statistical resource usage model $\mathcal{M}$ that relates to an operation r^{op} on the resource r^{uri}. The statistical information of model $\mathcal{M}$ is characterized by random variables X_1, X_2 that represent the number of packets and the number of octets associated with the resource operation. The model characterizes usage of a monitored resource within the specific period of observation which is given by a fixed-size time window.

Discrimination

[edit]

Traffic classifier computes the similarity of currently observed behavior to the known behavior represented by corresponding resource usage model $\mathcal{M}$. In the operational phase, the network communication is analyzed as follows:

Step 1 [edit]

The traffic is captured within a given time window.

Step 2 [edit]

For each flow e resource usage label r is determined.

Step 3 [edit]

The expected resource usage model $\mathcal{M} = (f_{X_1,X_2}(x_1, x_2), t)$ is retrieved from the usage profile $\mathcal{P}$.

Step 4 [edit]

Flow features are extracted from flow e to form a vector of observations $\vec{y}$.

Step 5 [edit]

Finally, the joint probability function of model $\mathcal{M}$ is used to compute the probability for the observed behavior, $p = f_{X_1,X_2}(\vec{y})$. If the probability is greater than threshold t, the flow is marked as normal. Otherwise, it is labeled as abnormal.

Evaluation [edit]

The evaluation is performed by computing hit ratio (recall) and false positives. For each executed test, the following quantities are measured:

N: the number of all normal flows in the testing dataset

N_+: the number of flows correctly classified as normal (true positives)

N_-: the number of flows incorrectly classified as normal (false positives)

Hit ratio is computed as follows:

$H_r = \frac{N_+}{N}$.

False positive is given by the following equation:

$F_p = \frac{N_-}{N_- + N_+}$.

Table bellow presents results of flow classification using different profiles. The profile computed from idle dataset has poor hit ratio. It is because that dataset does not contain enough patterns to represent behavior of the system adequately. We also tested profiles based on samples drawn from multiple data files achieving hit ratio around 90%.

Profile	Hr	Fp
idle	39.91%	2.31 %
regular	75.98%	7.23 %
observe	84.82%	8.17 %
idle+regular	88.72%	6.36 %
idle+regular+observe	90.85%	6.46%

Source codes [edit]

As a part of this case study, we provide the example of a possible solution including all source codes necessary for creating a model and discriminate traffic.

The solution is represented by a dotnet core console application that implements a method for classification of CoAP network flows. The classification is based on the identification of communication patterns using a statistical model. Currently, the application requires a CSV input file containing decoded PCAP packets. This input file can be created using tshark as follows:

```
tshark -T fields -e frame.time_epoch -e ip.src -e ip.dst -e udp.srcport -e udp.dstport -e udp.length -e coap.code -e coap.type -e coap.mid -e coap.token -e coap.opt.uri_path_recon -E header=y -E separator=, -Y "coap && !icmp" -r <INPUT> > <OUTPUT-CSV-FILE>
```

The application provides learning and classification modes. Learning mode serves for creating a profile from provided CoAP communication. Classification mode applies previously created profile to identify known communication.

Learning Mode [edit]

To create a profile the application is executed with Learn-Profile command. Two mandatory arguments are expected. The first argument is a CSV file representing CoAP communication, which is used to compute statistical models for CoAP resources. The tool will write the computed profile to the output file specified as the second argument.

```
dotnet Ironstone.Analyzers.CoapProfiling.dll Learn-Profile -InputFile=SampleData\coap-learn.csv -WriteTo=SampleData\coap.profile
```

Classification Mode [edit]

Once we have a profile created from the representative samples of CoAP communication, we can use this profile to classify the CoAP flows. The profile is stored in the binary file, which the tool loads and uses to discriminate the input CSV file. To use the tool in the classification mode, execute it with Test-Capture command.

```
dotnet Ironstone.Analyzers.CoapProfiling.dll Test-Capture -ProfileFile=SampleData\coap.profile -InputFile=SampleData\coap-test.csv
```

The tool prints the output table for each time window. The table consists of flows each occupying a single row. For each flow, a score is computed. Depending on the calculated score and the corresponding model threshold the flow is classified.

The source codes including sample data sets are available at: https://github.com/rysavy-ondrej/Ironstone/tree/master/Methods/Detection/CoapProfiling

Further Reading [edit]

- C. Manikopoulos and S. Papavassiliou, “Network intrusion and faultdetection: A statistical anomaly approach,”IEEE Communications Mag-azine, 2002.
- M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, “Traffic classificationthrough simple statistical fingerprinting,”ACM SIGCOMM ComputerCommunication Review, 2007.