ON THE APPLICABILITY OF BINARY CLASSIFICATION TO DETECT MEMORY ACCESS ATTACKS IN IOT

C&ESAR 2018 - Rennes | CEA Leti | KERROUMI Sanaa | 08/11/18
1 IoT node
2 Related works
3 Problem statement
4 Proposed methodology
5 Results
6 Take out and lessons learned
What’s an IoT Node

- **Internet Of Things**
  “The interconnection via the internet of computing devices embedded in everyday objects enabling them to communicate”

- **The “thing” in IoT can be anything and everything as long as it has a unique identity and can communicate via the internet**
  - Sensors, actuators or combined sensor/actuator
  - Limited capabilities in terms of their computational power, memory, energy, availability, processing time, cost, …
    - **limits their abilities to handle encryption or other data security functions**
  - Designed to disposable
    - **updates/security patches may be difficult or impossible.**
  - Designed to last for decades
    - **any unpatched vulnerabilities will stay for very long**
  - A foothold in the network (e.g., IoT goes nuclear, thermometer in the fish tank attack)

EDGE-NODE VULNERABILITIES: WHAT COULD POSSIBLY GO WRONG?
EDGE-NODE VULNERABILITIES: WHAT COULD POSSIBLY GO WRONG?

• **Attack modes:**
  • Software attacks
  • Side channel attacks
  • Physical attacks
  • Network attacks

• **Why are we interested in the memory access attacks?**
  • It is particularly hard to fake or hide malicious tasks memory accesses
  • It offers a great view on what’s going on inside the device
  • Alluring target for the attacker
    • Control the node
    • Read encryption keys or protected code
    • …
EXISTENT COUNTERMEASURES: PREVENT VS PROTECT

<table>
<thead>
<tr>
<th>Fuses and flash readout protection</th>
<th>Encryption</th>
<th>Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td><strong>Pros</strong></td>
<td><strong>Pros</strong></td>
</tr>
<tr>
<td>• Inexpensive</td>
<td>• Preserve privacy and confidentiality</td>
<td>• Proactive</td>
</tr>
<tr>
<td>• Efficient</td>
<td>• Convenient</td>
<td>• Scalable</td>
</tr>
<tr>
<td>• Easy to implement</td>
<td>• Expenses</td>
<td>• Pervasive</td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td><strong>Cons</strong></td>
<td><strong>Cons</strong></td>
</tr>
<tr>
<td>• Mostly set on level that permits access to memory (post deployment upgrades)</td>
<td>• Compatibility</td>
<td>• False Positives</td>
</tr>
<tr>
<td></td>
<td>• Encryption keys</td>
<td>• Leak</td>
</tr>
<tr>
<td></td>
<td>• Widespread security compromise</td>
<td>• Mimicry attacks</td>
</tr>
</tbody>
</table>
• **Memory heat map**
  • Idea: profiling memory behavior by representing the frequency of access to a particular memory region (regardless of which component accessed it) during a time interval. The MHM is then combined with an image recognition algorithm to detect any anomalies.

• **Strengths:**
  • system wide anomalies detection (not just malicious ones)
  • can be used in real-time embedded systems

• **Limitations:**
  • expensive to compute: need to store several images of nominal MHM
  • wrong architecture (Config3 and higher)

Yoon, Man Ki, et al, "Memory heat map: anomaly detection in real-time embedded systems using memory behavior". In Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE (pp. 1-6). IEEE.
**System call distribution**

- **Idea**: learn the normal system call frequency distributions, collected during legitimate executions of a sanitized system, combined by a clustering algorithm (k-means). If an observation, at a run time, is not similar to any identified clustered. The observation is doomed malicious.

**Strengths**:
- simple

**Limitations**
- Require an OS
- need a throughout training
- no adaptation of centroids (any change even if nominal would be flagged as malicious)
- application to be monitored need to be very deterministic
- definition of cut off line influence the FPR and detection rate

---

PROBLEM STATEMENT

• Existent detection solutions are:
  • Not directly related to memory access attacks
  • Too expensive to compute
  • Used features in detection are either hard or impossible to acquire for constrained node (e.g., hardware performance counters, control flow, instruction mix, etc.)

• Analyze the effectiveness of binary classifiers combined by simples features to detect memory access attacks in the context of a low cost IoT node
METHODOLOGY:

- **2 phases’ methodology:**
  - Design: performed during the design of the node to build the detector
  - Operation: the detector in operation

  ![Flowchart of 2 phases' methodology](chart.png)

- In this presentation we will focus on the design part of the detector
USE CASE PRESENTATION: CONNECTED THERMOSTAT

Temperature measurement

10 seconds

Interrupts

User action buttons

Screen display

Temperature

Mode

Temp. target

Internal variables

Heat power

Variables stored into RAM

Heating regulation loop

1 minute

Send data to heating device

Wake up signal

10 seconds / wake up signal
**IN MORE DETAILS**

- **Raw data: memory access log**
  - Timestamp
  - Accessed address
  - Data manipulated
  - Type of data
  - Flag to indicate if the access is nominal or suspicious

- **Features – computed each time window**
  - Number of memory reads, number of memory accesses, cycles between consecutive reads, address increment, number of “unknown” (first-encountered) addresses, amount of read/accessed data ...

---

**Feature extraction & selection**

**Machine learning method**

**Evaluation and trade-offs**
ATTACK SCENARIOS

- **Classic dump (CD):** basic memory dump require minimal effort from the attacker:
  - Attacker reads the entire memory in a contiguous way, the memory reads are spaced regularly in time and memory space

Attacker assumed to be aware of the presence of some security monitor ➔ avoid obvious change in the memory patterns of the device

- **Dumping in bursts (DB):**
  - The memory is read in bursts, the accessed addresses are still contiguous but the time step between two consecutive reads is incremented by constant (BD(cts)), linearly (BD(lin)) or randomly (BD(rand))

- **Dump in non contiguous way (NG)**
  - The address increment between two consecutive reads is incremented by constant (NG(cts)), linearly (NG(lin)) or randomly (NG(rand))
TRAINING & TESTING DATASETS

<table>
<thead>
<tr>
<th>dataset</th>
<th>Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>Nominal+ CD</td>
<td>DB and NG</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>(1) Nom+(CD+NG+BD) (2) Nom+(CD+NG) (3) Nom+(CD+BD)</td>
<td>(1) Nom+(CD+NG+BD)* (2) Nom+BD (3) Nom+ NG</td>
</tr>
</tbody>
</table>
**EXTRACTED FEATURES**

- **Nread**: number of reads per time interval
- **Inc**: number of address increment per time interval
- **Time2Reads**: average time elapsed between two consecutive reads in time interval
- **NmembAcc**: number of memory access per time interval
- **UnknownAd**: number of unknown addresses accessed during a time interval

**Processor/memory trace**

**Feature extraction & selection**

**Machine learning method**

**Evaluation and trade-offs**
Let $X = \{x_1, \ldots, x_n\}$ be our dataset and let $y_i \in \{1, -1\}$ be the class label of $x_i$.

The decision function ($f$) assigns each new instance a label based on prior knowledge gathered during the training.

- **List of classifiers included in the analysis**
  - K nearest neighbor, Support vector machine, decision tree, random forest, naïve Bayes, linear discriminant analysis and quadratic discriminant analysis.
• **Assumption:**
  - Features are independent

• **Intuition:**
  - Given a new unseen instance, we (1) find its probability of it belonging to each class, and (2) pick the most probable.

\[
P(c_j|x) = \frac{P(x|c_j)P(c_j)}{p(x)}
\]
**LINEAR DISCRIMINANT ANALYSIS**

- **Assumption**:  
  - Every class distribution is Gaussian and the covariance matrices are identical.

- **Intuition**  
  \[
  \hat{\delta}_k(x) = x^T \Sigma^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^T \Sigma^{-1} \hat{\mu}_k + \log(\hat{\pi}_k)
  \]

  - \(\hat{\delta}_k(x)\) is the estimated discriminant score that the observation will fall in the \(k\)th class based on the value of the predictor variable \(x\).
  - \(\hat{\mu}_k\) is a class-specific mean vector, and \(\Sigma\) is a covariance matrix that is common to all \(K\) classes.
  - \(\hat{\pi}_k\) is the prior probability that an observation belongs to the \(k\)th class.

- An observation will be assigned to class \(k\) where the discriminant score \(\hat{\delta}_k(x)\) is the largest,
QUADRATIC DISCRIMINANT ANALYSIS

**Assumptions**

- Class distribution is Gaussian but with different covariance matrices

**Intuition**

\[
\hat{\delta}_k(x) = x^T \Sigma^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^T \Sigma^{-1} \hat{\mu}_k + \log(\pi_k)
\]

- \( \hat{\delta}_k(x) \) is the estimated discriminant score that the observation will fall in the \( k \)th class based on the value of the predictor variable \( x \)
- \( \hat{\mu}_k \) is a class-specific mean vector, and \( \Sigma \) is a covariance matrix that is common to all \( K \) classes
- \( \pi_k \) is the prior probability that an observation belongs to the \( k \)th class
- an observation will be assigned to class \( k \) where the discriminant score \( \hat{\delta}_k(x) \) is the largest,
K NEAREST NEIGHBOR KNN

- **Assumption:**
  - Data have a notion of distance (data are in a metric space)

- **Intuition:**
  - Lazy learner ➔ store all the training data and for every new incoming new observation, the algorithm will try to find the k nearest neighbor and do a majority voting
DECISION TREE

- **Assumption:** None
- **Intuition**
  - Decompose a complex decision into a union of several simpler decisions

Diagram:
- Decision Tree: Should I accept a new job offer?
- Nodes: salary at least $50,000, commute more than 1 hour, offers free coffee, accept offer, decline offer
- Root node: decision nodes
- Leaf nodes: number of memory reads per time interval vs. number of address increments
RANDOM FOREST

- **Assumption:**
  - None

- **Intuition**
  - A collection or ensemble of simple tree predictors, each capable of producing a response when presented with a set of predictor values.
• **Assumption**
  - Linear SVM: the decision boundary is linear

• **Intuition:**
  - The decision boundary should be as far away from the data of both classes as possible
    - We should maximize the margin \( m = \frac{1}{||w||} \)
  - This maximum-margin separator is determined by a subset of the data points (support vectors).

  ➔ It will be useful computationally if only a small fraction of the data points are support vectors, because we use the support vectors to decide which side of the separator a test case is on.
SUPPORT VECTOR MACHINE: SOFT MARGIN

Slack variables $\xi_i$ can be added to allow misclassification of difficult or noisy examples.

1. Slack variables $\xi_i$ can be added to allow misclassification of difficult or noisy examples.
SUPPORT VECTOR MACHINE : KERNEL SVM

Projection to higher dimensional space where we can find a linear separator
The final classification rule is quite simple:

$$f(x_{tes}) = \text{sig} \ \eta b + \sum_{s \in SV} \alpha_s y_s K(x_{tes}, x_s)$$

- All the cleverness goes into selecting the support vectors that maximize the margin and computing the weight to use on each support vector.
- We also need to choose a good kernel function and set the parameters of the used kernel.
- Popular kernels:
  - polynomial of a degree $d$ $K(x_i, x_j) = (x_i^T x_j + 1)^d$
  - radial basis function $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$
HOW CAN WE BUILD THE DETECTOR? TRAINING ON CLASSIC DUMP ATTACK

Processor/memory trace
Feature extraction & selection
Machine learning method
Evaluation and trade-offs

Number of memory reads per time interval

Number of address increment

- Attack
- Nominal
- Nearest Neighbors
- Linear SVM
- RBF SVM
- Decision Tree
- Random Forest
- Naive Bayes
- LDA
- QDA
**EVALUATION METRICS**

- **False positive rate:** number of false alarms generated by the classifier
  \[ FP = \frac{FP}{FP + TN} \]

- **False negative rate:** number of miss detection by the classifier
  \[ FNR = \frac{FN}{FN + TP} \]

- **Precision:** is a measure of a classifiers exactness
  \[ PP = \frac{TP}{TP + FP} \]

- **Leakage:** number of bytes leaked before the classifier detects

- **Cost**
  - Memory footprint of the classifier (in bytes)
  - Computation (number of basic arithmetic operation needed to classify one instance)
**COMPUTATIONAL COST FOR CLASSIFYING ONE INSTANCE**

**Principle**

- In order to compare the computation cost in predicting the label of one instance for each classifier, we decomposed the learnt decision function of each classifier to basic arithmetic operations (additions, subtraction, comparison, multiplications, square root; exponential and divisions).
- The memory cost is computed by calculating the number of variables needed by each classifier.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Add(+)/Sub(-)/comparison</th>
<th>Mul (×)</th>
<th>Sqrt()</th>
<th>Exp()</th>
<th>Div</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSVM</td>
<td>$(d + 1)n_s - 1$</td>
<td>$(d + 2)n_s$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RSVM</td>
<td>$(d + 1)n_s$</td>
<td>$(d + 3)n_s$</td>
<td>0</td>
<td>$n_s$</td>
<td>$n_s$</td>
</tr>
<tr>
<td>KNN</td>
<td>$2n(d + 1) - 2 \times k$</td>
<td>$n \times d$</td>
<td>$n$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LDA</td>
<td>$d$</td>
<td>$d$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QDA</td>
<td>$d^2 + d$</td>
<td>$d^2 + 2d$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>$n_c$</td>
<td>$2d$</td>
<td>$d$</td>
<td>$d$</td>
<td>$2d$</td>
</tr>
<tr>
<td>Random Forest</td>
<td>$n_{tre} \times (h + 1)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>$h$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- $d$: number of features
- $n_s$: number of support vectors
- $n$: number of observations in training dataset
- $k$: number of neighbors
- $n_c$: number of classes
- $h$: depth of the tree
- $n_{tree}$: number of trees in the random forest
DETECTION PRECISION & LEAKAGE OF CLASSIFIERS TRAINED ON CLASSIC DUMP
DETECTION PRECISION & LEAKAGE OF CLASSIFIERS TRAINED ON VARIANTS DUMPS
CLASSIFIERS PERFORMANCE (TRAINED ON CLASSIC DUMP)

- False alarm rate vs. Leakage (bytes)
- Memory footprint vs. Computation cost in basic operations

Classifiers: RSVM, L SVM, KNN, LDA, RF, QDA, NB, RSVM, L SVM, QDA, RF, D T
CLASSIFIERS PERFORMANCE (TRAINED ON DUMP VARIANTS)

- False alarm rate (%)
  - RSVM: 0.5
  - L SVM: 0.4
  - NB: 0.3
  - DT: 0.2
  - KNN: 0.1
  - LDA: 0.0

- Leakage (bytes)
  - $3 \times 10^1$
  - $4 \times 10^1$
  - $6 \times 10^1$
  - $10^2$

- Memory footprint in bytes
  - RSVM: $10^5$
  - KNN: $10^4$
  - NB: $10^3$
  - L SVM: $10^2$
  - QDA: $10^1$
  - LDA: $10^0$
  - DT: $10^{-1}$

- Computation cost in basic operations
  - $10^2$
  - $10^3$
  - $10^4$
COMPARISON OF CLASSIFIERS PERFORMANCE

Trained on classic dump

Trained on dump variants
**Take out:**
- Binary classifiers are a great choice for low cost detectors
- Diversifying the training dataset can increase the accuracy of the detection but that comes at the cost of the implementation complexity
- Even when trained on limited examples of attacks binary classifiers were able to detect efficiently (few bytes leakage and detection accuracy around 90%)

**Next step**
- Implementation on hardware
- Exploration of other types of attacks
- Evaluation of mimicry attack cost to evade the detection
Q&A

Contact us for more details:
Sanaa.kerroumi@cea.fr
Anca.Molnos@cea.fr
Damien.Couroussé@cea.fr