EDDIE: EM-Based Detection of Deviations in Program Execution
Nazari et al, ISCA 2017

Presenter: Di Jin, Kaiyu Yang
Motivation

- Security matters
  - Hackers want your private information
  - In IoT (Internet of Things), attacks have further influence.
  - Advanced attacks can bypass static malware detection (e.g., anti-viruses, memory scan) through code mutation / injection.
  - A fast, accurate detector monitoring the software execution is in urgent need.

Source: https://www.shutterstock.com/search/malware
Traditional malware detection

- **Signature-based**
  - Detect attack if the signatures have been observed.
    - New attack signatures
    - Could be bypassed by metamorphic malware

- **Anomaly-based**
  - Monitor a set of **features**, report any deviations from the reference model as attacks.
    - Software monitoring
      - High performance overhead, low accuracy.
    - Hardware monitoring
      - High power consumption
EM emanations
  - Widely used in attacks
    - Side-channel attack (e.g., Van Eck phreaking)
    - Program profiling through EM signals

Can we use EM for security?

EDDIE (*EM-based Detection of Deviations in Program Execution*)
  - An EM emanation-based approach to monitoring program execution and detecting anomalies.
    - No direct intrusion to monitored systems, minimized overhead.
    - In the context of code injection, both burst and slow injections should be detected w. high accuracy and low latency.
**EDDIE: Overview**

- Idea: use the observed EM spectra of each part of the program over time as reference to find deviations.
STFT (Short-Term Fourier Transformation)
- Input: time-domain signals
- Output: a sequence of windows: time-frequency distribution

STS (Short-Term Spectrum)
- Convert signals in windows into spectrum.

Reference: a sequence of STSs in training
- Model loop regions and inter-loop region.
  - Peaks: active loops

  If: $|STS_{\text{reference}} - STS_{\text{monitored behavior}}| > \theta$
  Then: mark as anomaly.

● Training phase
  ○ Goal:
    ■ Find the possible STS sequences in which loop and inter-loop regions may execute.
    ■ Collect & map sample windows to those regions.
  ○ Loop-level state machine
    ■ “Peaks” in spectrum.
    ■ Profile program execution.
  ○ Measurement:
    ■ Signal sequence
    ■ Region identifier
    ■ Loop entry time
    ■ Exit time
**EDDIE**: Implementation

- **Statistical test**
  - STSs (sequence of Short-Term Spectra) belonging to the same code region are unlikely the same.
  - K-S test: nonparametric test to compare the observed and reference STS distributions.
  - One test for a peak: 1st strongest, 2nd strongest, …

Parametric test is not suitable in this case.

Source: Nazari et al.
**EDDIE: Implementation**

- Trade-off between detection accuracy and latency
  - The number of monitoring-observed STSs for K-S test ($n$)
    - Small $n$: low latency (recently STSs), low accuracy
    - Large $n$: high latency, high accuracy
  - In training, EDDIE determines $n$ separately for each region.
    - Perform a “grid search” on $n$ for the minimum false rejection rate (training phase is injection-free)

Source: Nazari et al.
Experiments on a Real IoT Device

- **Setup**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Detection Latency (ms)</th>
<th>False positives (%)</th>
<th>Accuracy (%)</th>
<th>Coverage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitcount</td>
<td>42</td>
<td>0.99</td>
<td>100</td>
<td>99.9</td>
</tr>
<tr>
<td>Basicmath</td>
<td>25</td>
<td>1.8</td>
<td>99.9</td>
<td>99.9</td>
</tr>
<tr>
<td>Susan</td>
<td>32</td>
<td>1.39</td>
<td>92.1</td>
<td>95.9</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>25</td>
<td>1.08</td>
<td>99.9</td>
<td>99.7</td>
</tr>
<tr>
<td>Patricia</td>
<td>28</td>
<td>0.98</td>
<td>92.3</td>
<td>95.2</td>
</tr>
<tr>
<td>GSM</td>
<td>24</td>
<td>0.9</td>
<td>96.2</td>
<td>57.1</td>
</tr>
<tr>
<td>FFT</td>
<td>17</td>
<td>0.76</td>
<td>93</td>
<td>99</td>
</tr>
<tr>
<td>Sha</td>
<td>11</td>
<td>1.9</td>
<td>97.2</td>
<td>98.9</td>
</tr>
<tr>
<td>Rijndael</td>
<td>12</td>
<td>0.56</td>
<td>99.9</td>
<td>97.1</td>
</tr>
<tr>
<td>Stringsearch</td>
<td>11</td>
<td>0.19</td>
<td>99.9</td>
<td>99.9</td>
</tr>
</tbody>
</table>

- **Injection**
  - Outside loops: invoking a shell and return (476k instructions, 3ms execution time)
  - Inside loops: 4 integer operations and 4 memory accesses (8 instructions)

Source: Nazari et al.
Effects of Various Factors

● Processor architectures
  - Power consumption signal generated by a simulator
  - 51 configurations, in-order or out-of-order, issue widths, pipeline depths, ROB sizes
  - Out-of-order cores have significantly higher latency
  - Pipeline depth has a weak effect, which diminishes when increasing the injection size

Table 2: EDDIE’s latency and accuracy when using a simulator.

Figure 5: False negative rate of variable injection rates.

Figure 4: Detection latency of 15 different regions in in-order and out-of-order architecture.

Figure 7: Detection latency of variable injection rates.

Source: Nazari et al.
Effects of Various Factors

- **Size of Injection**
  - Even two-instruction injections can be detected with high accuracy

- **Inject outside Loops**
  - Instructions type
    - 8 ADD vs. 4 ADD & 4 STORE
  - Off-chip operations are easier to detect

*Figure 8: EDDIE’s accuracy when changing the number of injected instructions outside loops.*

*Figure 10: Effect of changing the type of injected instructions on latency and accuracy.*

Source: Nazari et al.
The paper proposes EDDIE, an EM-based method for detecting anomalies in program execution. It has the advantage of introducing no overheads or any hardware/software change in the monitored system. EDDIE characterizes normal execution behavior in terms of peaks in the EM spectrum and identifies abnormal peaks during testing. EDDIE is evaluated both on a real IoT system and in a simulator. It is shown to be effectively for different processor architectures and code injection patterns.
Discussion

- Is EDDIE applicable in real-world (industry, academia)?
- What if the environment is power-costly, EM-noisy?
- Why does EDDIE try to avoid direct intrusion on the monitored system?
- Can EM-based anomaly detection be improved through ensembling? Features in existing works: acoustics emanations, power, timing variations, etc.
- Can we use models such as SVM to directly classify the EM signal to be normal/abnormal?
Q&A

- Thanks