ACE: Just-in-time Serverless Software Component Discovery Through Approximate Concrete Execution

BY ANTHONY BYRNE¹, SHRIPAD NADGOWDA², AND AYSE K. COSKUN¹

¹Boston University; ²IBM T.J. Watson Research Center
Serverless Containers: More Than Just FaaS

- “Serverless computing” encompasses more than Lambda functions
  - FaaS requirements (language, runtime, etc.) too strict for many developers

- Cloud providers offer serverless container platforms as a compromise
  - “Just hand us your Docker image, and we’ll handle everything else”
  - Bestow serverless benefits on any containerized app: scaling, billing, etc.
  - Allows executables not typically found in FaaS: compiled C/C++/Go binaries
What Could Possibly Go Wrong?

- Cloud apps often made of in-house and off-the-shelf parts
  - Libraries, microservices, helper tools, etc.

- “Undesirable” software components key to many scandals
  - OpenSSL: Heartbleed bug exposed 66% of web servers (2014)
  - Apache Struts: 143 million Equifax records breached (2017)
  - Several court cases regarding licensing (e.g., AGPL)

- Binaries harder to screen for undesirables than Python/Java/JS code
  - No “requirements.txt” or other component manifest, just 1’s and 0’s

*How can we discover software components in serverless binaries?*
Just-in-time Component Discovery for Serverless

- Serverless gives cloud providers unprecedented access to developers’ applications
  - Use it for good by scanning apps “JIT” before harm occurs

- Serverless binaries present special challenges
  - Metadata stripped and obscured through static linkage
  - Most analysis techniques slow and error-prone

- Binary function fingerprinting provides a framework
  - Disassemble binaries, fingerprint functions, check blocklist
  - If fingerprint similar to “known bad” one, then flag for review

How can we fingerprint a binary function?
Introducing Approximate Concrete Execution

**Step 1**
Disassemble raw binary and determine function bounds

- Prior work\(^*\) provides function bounds in stripped binaries

**Step 2**
Translate assembly code to IR function-by-function

- REIL: simple MIPS-like register layout, infinite memory

**Step 3**
Filter code and provision a REIL “approximate VM”

- Remove all control flow instructions and sort (to account for compiler diffs.)

**Step 4**
Approximately execute and collect final aVM context

- Post-execution reg. values become fingerprint

---

Evaluating ACE for Serverless: Goals

- A JIT serverless binary fingerprinting method must...
  - produce representative fingerprints resistant to compiler variations
  - introduce very little overhead to the serverless environment
  - be tunable to different users’ needs and applications
Evaluation

Accuracy: find the undesirable function
- Inject dummy function into ~230 C/C++ cloud apps
- Compile clean & injected apps and disassemble
- Classify each of the 37k functions using ACE
  - Positive: exact match to known dummy fingerprint

Overhead
- 5.2x faster end-to-end than baseline
  - Most functions fingerprinted in <10 milliseconds
- ACE requires no pre-training
  - Learning-based methods like SAFE require training and constant updating of ML models
- Minimal overall impact on cold-start or deployment latency

![Experiment 1: Classification Accuracy](image1)

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE</td>
<td>1.00</td>
<td>0.95</td>
<td>0.98</td>
</tr>
<tr>
<td>Baseline (SAFE)</td>
<td>0.85</td>
<td>0.90</td>
<td>0.88</td>
</tr>
</tbody>
</table>

![Experiment 2: Overhead Comparison](image2)

<table>
<thead>
<tr>
<th>SW Component Discovery</th>
<th>Function Fingerprinting</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE</td>
<td>1.1</td>
</tr>
<tr>
<td>Baseline (SAFE)</td>
<td>15.7</td>
</tr>
<tr>
<td></td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>5.9</td>
</tr>
</tbody>
</table>
Why ACE for Serverless Component Discovery?

**Speed**
- No model training
- No complex instruction emulation
- 5.2x faster end-to-end

**Resiliency**
- Code filtering mitigates compiler variations
- 99% accurate binary classification of undesirable code

**Versatility**
- Several variables (sensitivity, aVM register size, etc.) tunable to users’ needs
- Output vector suited to almost any search technique
- Serverless container platforms vulnerable to problems with undesirablesoftware components
- ACE enables just-in-time discovery of these components through binary function fingerprinting
- We’re excited to see future work apply the aVM to further improve serverless performance & security

Concluding Remarks

More info at bu.edu/peaclab
Please send feedback to abyrne19@bu.edu

This work has been partially funded by IBM Research