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

BY ANTHONY BYRNE[1], SHRIPAD NADGOWDA[2], AND AYSE K. COSKUN[1]

[1]Boston University; [2]IBM T.J. Watson Research Center

Sixth International Workshop on Serverless Computing (WoSC6)
December 8, 2020
www.serverlesscomputing.org/wosc6/#p7

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 <u>serverless container platforms</u> 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

AWS Fargate

Cloud Run

What Could Possibly Go Wrong?

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

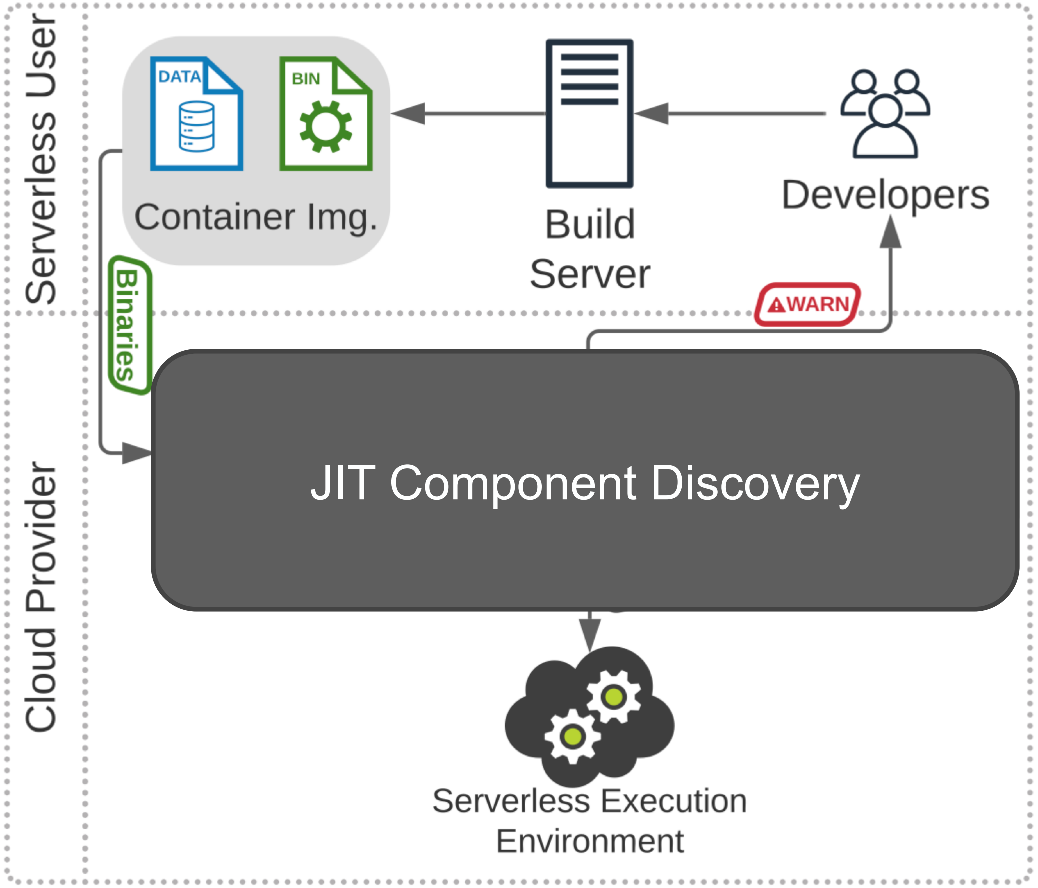
- “Undesirable” software components key to many scandals
  - OpenSSL: Heartbleed bug exposed 66% of web servers (2014)
  - Apache Struts: 143 million Equifax records breached (2017)
  - Several court cases regarding licensing (e.g., AGPL)

- 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

```
push r1
mov r2, 5
jnz 0x56
```

```
sub r0, 8, r0
stm r1, r0
str 5, r2
bisnz r4, r3
jcc r4, 56
```

```
REGISTERS    RAM
r0 = 10      0x0 = 0
r1 = 10      0x1 = 1
r2 = 10      0x2 = 2
r3 = 10      0x3 = 3
r4 = 10      0x4 = 4
```

```
REGISTERS    RAM
r0 = 2       0x0 = 0
r1 = 10      0x1 = 1
r2 = 5       0x2 = 1
r3 = 10      0x3 = 3
r4 = 1       0x4 = 4
```

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

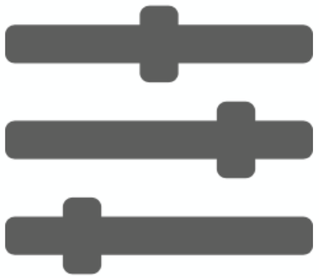
Approximately execute and collect final aVM context

- Post-execution reg. values become fingerprint

* D. Andriesse, A. Slowinska, and H. Bos, "Compiler-Agnostic Function Detection in Binaries," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017, pp. 177–189.

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



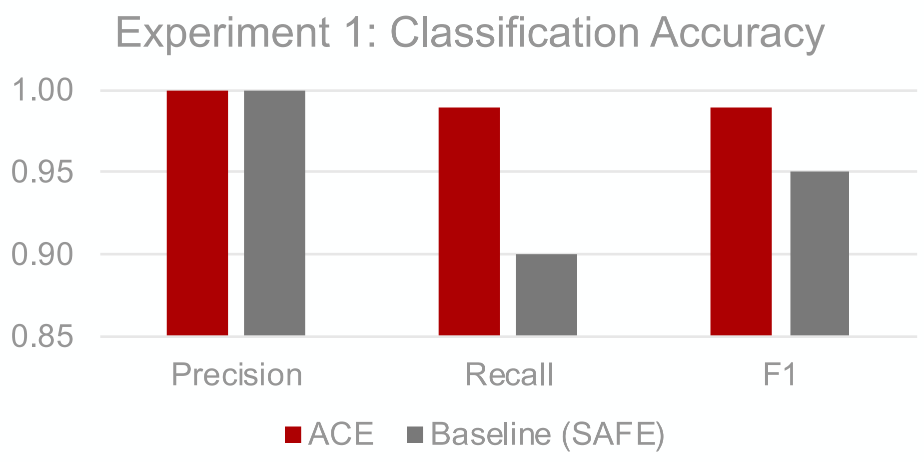
Evaluation

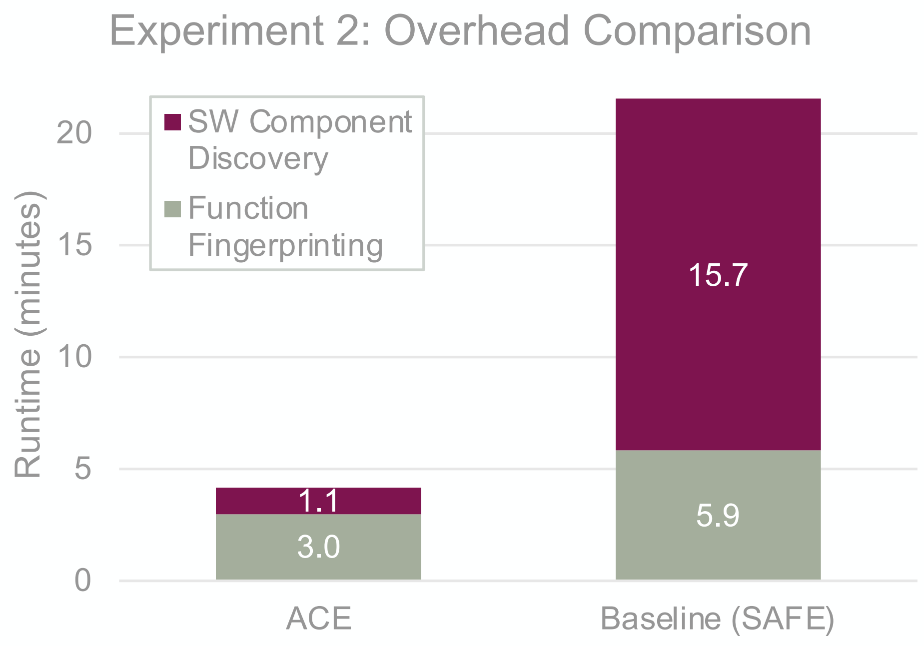
Accuracy: find the undesirable function

- 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





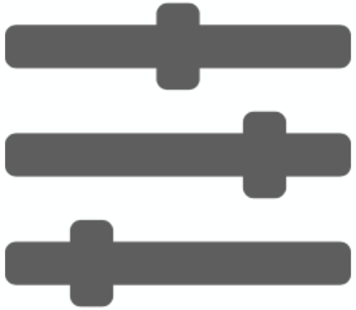
Why ACE for Serverless Component Discovery?

Speed

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

Resiliency

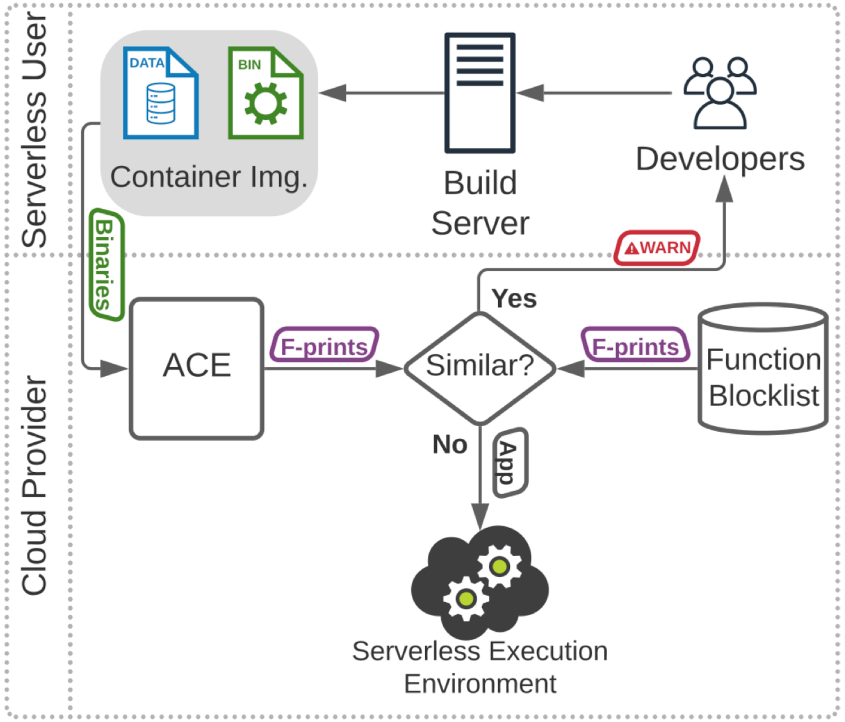
- Code filtering mitigates compiler variations
- 99% accurate binary classification of undesirable code



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 **undesirable software 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