Scheduling Black-box Mutational Fuzzing
ACM CCS 2013

Maverick Woo
Carnegie Mellon University
pooh@cmu.edu
Our Crew

Maverick Woo  Sang Kil Cha  Samantha Gottlieb  David Brumley
1 Introduction

A General (or professor) walks into a cramped cubicle, telling the lone security analyst (or graduate student) that she has one week to find a zero-day exploit against a certain popular OS distribution, all the while making it sound as if this task is as easy as catching the next bus. Although our analyst has access to several program analysis tools for finding bugs [8, 10, 11, 21] and generating exploits [4, 9], she still faces a harsh reality: the target OS distribution contains thousands of programs, each with potentially tens or even hundreds of yet undiscovered bugs. What tools should she use for this mission? Which programs should she analyze, and in what order? How much time should she dedicate to a given program? Above all, how can she maximize her likelihood of success within the given time budget?
Typical Exploit Generation

Bug Finding

Fuzzing  ➔  Bug Triage

crashes ➔ bugs ➔ Exploit Generation
Scheduling is Equally Important

1 Introduction

A General (or professor) walks into a cramped cubicle, telling the lone security analyst (or graduate student) that she has one week to find a zero-day exploit against a certain popular OS distribution, all the while making it sound as if this task is as easy as catching the next bus. Although our analyst has access to several program analysis tools for finding bugs [8, 10, 11, 21] and generating exploits [4, 9], she still faces a harsh reality: the target OS distribution contains thousands of programs, each with potentially tens or even hundreds of yet undiscovered bugs. What tools should she use for this mission? Which programs should she analyze, and in what order? How much time should she dedicate to a given program? Above all, how can she maximize her likelihood of success within the given time budget?
Scheduling Black-box Mutational Fuzzing
A common program testing technique popularized by Miller et al. in late 1980s [18]

- Use a **fuzzer** to generate **test inputs** to program-under-test
- At its simplest, look for **crashes**—memory corruption, uncaught exceptions, failed assertions, etc.
A **black-box** fuzzer observes a program’s **I/O behavior** only

- Simplification: only distinguish *termination* vs. *crash*

Detect anomaly by *mutating* a valid input (= seed)
A black-box fuzzer observes a program’s I/O behavior only
• Simplification: only distinguish termination vs. crash

Given a seed input $s$ and a mutation ratio $r$:
1. Select $d = r \times |s|$ bits in $s$ uniformly at random
2. Flip each selected bit with probability $\frac{1}{2}$
Key Observations:
1. We can *reproduce* a program crash by storing (a) the seed input and (b) the PRNG seed
2. Mutation = *uniform sampling* from the Hamming cube of radius $d$ centered at $s$
Scheduling **Black-box Mutational Fuzzing**

### Key Observations:
1. We can **reproduce** a program crash by storing (a) the seed input and (b) the PRNG seed.
2. Mutation = **uniform sampling** radius $d$ centered at $s$.

---

“Fuzz Configuration”
(i) program $p$
(ii) seed input $s$
(iii) mutation ratio $r$
Scheduling **Black-box Mutational Fuzzing**

**Key Observations:**

1. We can *reproduce* a program crash by storing (a) the seed input and (b) the PRNG seed.
2. Mutation = *uniform sampling* over a radius \(d\) centered at \(s\).

**“Fuzz Configuration”**

(i) program \(p\)
(ii) seed input \(s\)
(iii) \(0.04\%\)
Key Observations:
1. We can reproduce a program crash by storing (a) the seed input and (b) the PRNG seed.
2. Mutation = uniform sampling radius $d$ centered at $s$
A *fuzz campaign* comprises a sequence of *epochs*:
1. takes a list of (program, seed) pairs as input
2. at the beginning of each epoch, picks one (program, seed) pair to fuzz based on data collected from previous epochs

We investigate two *epoch types*:

- **Fixed-run**: fixed number of fuzz runs in each epoch
  - implemented in CMU CERT BFF v2.6 [14]
- **Fixed-time**: fixed amount of time in each epoch
  - proposed in this paper
  - slightly harder to implement
Given a list of $K$ fuzz configurations \{($p_1, s_1$), $\cdots$, ($p_K, s_K$)\}, the \textit{Fuzz Configuration Scheduling (FCS)} problem seeks to maximize the number of \textit{unique} bugs discovered in a fuzz campaign that runs for a duration of length $T$.

\textbf{Important Assumptions:}
1. Only \textbf{one} configuration can be fuzzed within an epoch
2. Separate program analysis of ($p_i, s_i$) is \textbf{not} allowed
3. Bugs from different ($p_i, s_i$) are \textbf{disjoint}

See paper for discussions
How to Solve the FCS Problem?

Two *competing goals* during a fuzz campaign:

**Explore** each $(p_i, s_i)$ sufficiently often so as to identify pairs that can yield new bugs

**Exploit** knowledge of $(p_i, s_i)$ that are likely to yield new bugs by fuzzing them more

**Good News:**

- Clearly a *Multi-Armed Bandit (MAB)* problem!
Multi-Armed Bandits
MAB in Berlin
How to Solve the FCS Problem?

Two *competing goals* during a fuzz campaign:

*Explore* each \((p_i, s_i)\) sufficiently often so as to identify pairs that can yield new bugs

*Exploit* knowledge of \((p_i, s_i)\) that are likely to yield new bugs by fuzzing them more

Good News:

• Clearly a *Multi-Armed Bandit (MAB)* problem!

• *Lots* of published MAB algorithms
  – *provably optimal* algorithms for many settings, e.g., Auer et al. 2002 [2] handles certain *adversarial* cases
How to Solve the FCS Problem?

Bad News: recognizing “FCS ∈ MAB” is not enough

Given a list of $K$ fuzz configurations $\{(p_1, s_1), \ldots, (p_K, s_K)\}$, the Fuzz Configuration Scheduling (FCS) problem seeks to maximize the number of unique bugs discovered in a fuzz campaign that runs for a duration of length $T$.

1. Classic MAB: once you identify a good beer, it stays good
   $\Rightarrow$ drink it often to accumulate rewards 😊
2. Our Setting: each program has a finite number of bugs
   $\Rightarrow$ bug exhaustion gives a diminish of return 😞

We are not aware of MAB algorithms that cater to our case...
$\Rightarrow$ We need our own algorithms!
How to Solve the FCS Problem?

Bad News: recognizing “FCS ∈ MAB” is **not** enough

Given a list of $K$ fuzz configurations $\{(p_1, s_1), \ldots, (p_K, s_K)\}$, the Fuzz Configuration Scheduling (FCS) problem seeks to maximize the number of **unique** bugs discovered in a fuzz campaign that runs for a duration of length $T$.

1. Classic MAB: once you identify a good beer, it **stays** good
   $\Rightarrow$ drink it often to **accumulate** rewards 😊

2. Our Setting: each program has a **finite**
   $\Rightarrow$ bug **exhaustion** gives a diminishment

We are not aware of MAB algorithms that cater to our case... $\Rightarrow$ We need our own algorithms!
Key Observations:

1. We can *reproduce* a program crash by storing (a) the seed input and (b) the PRNG seed

2. Mutation = *uniform sampling* from the Hamming cube of radius $d$ centered at $s$
Modeling Black-box Mutational Fuzzing

Consider the repeated fuzzings of a fixed \( (p_i, s_i) \) and let \( outcome_i(j) \) denote the \( j \)-th outcome in the sequence:

- Termination \( \Rightarrow \) ID 0
- Crash \( \Rightarrow \) bug ID obtained from bug triage

**Key Observation:**

BMF is *memoryless*, i.e., \( outcome_i(j) \) are *i.i.d.* RVs for a fixed \( i \)
Coupon Collector’s Problem (CCP)

Suppose every box of breakfast cereal comes with a coupon that is randomly chosen among $M$ different coupon types

• How many boxes do you expect to buy before you have collected at least one coupon of each type?

Traditional Setting

• Coupon types are uniformly distributed $\Rightarrow \Theta(M \log M)$

Our Setting

• Bugs do not occur uniformly at random $\Rightarrow$ Weighted CCP
• Prevalence of different bugs is unknown ahead of time
Coupon Collector’s Problem (CCP)

Suppose every box of breakfast cereal comes with a coupon that is randomly chosen among $M$ different coupon types

• How many boxes do you expect to buy before you have collected at least one coupon of each type?

Traditional Setting

• Coupon types are uniformly distributed $\Rightarrow \Theta(M \log M)$

Our Setting

• Bugs do not occur uniformly at random
• Prevalence of different bugs is unknown

Also observed by Arcuri 2010 [1]
WCCP w/ Unknown is Intractable

No Free Lunch Theorem
(you did pay the registration, right?)
WCCP w/ Unknown is Intractable

No Free Lunch Theorem
Wolpert and Macready 2005 on [22]
• “Any two optimization algorithms are *equivalent* when their performance is averaged across *all* possible problems”
“Bring Your Own Prior”

No Free Lunch Theorem
Wolpert and Macready 2005 on [22]

• “Any two optimization algorithms are \textit{equivalent} when their performance is averaged across all possible problems”

Circumvention may be possible!

• NFL Theorem does not apply if we focus on distributions that are \textit{more likely} to occur in \textit{practice}

• More accurate \textit{model} $\Rightarrow$ More accurate \textit{predictions} $\Rightarrow$ More \textit{bugs}
Q: Suppose we have flipped a biased $H$-$T$ coin $n$ times and every time it comes up $H$. Does $\Pr[T]$ have to be small?

A: No, so long as $\Pr[T] < 1$, our observation is always possible.

Confidence Intervals:
$\Pr[T] < 3/n$ in 95% of all “parallel universes”

Usage:
1. Suppose $(p_i, s_i)$ has yielded $n$ different outcomes so far
2. Collectively call all $n$ outcome types $H$
3. With 95% confidence, $\Pr[T]$ (i.e., new outcome)] $< 3/n$

See discussion in Jovanovic 1997 [15]
Algorithm Design Space

We explore **3 dimensions** in algorithm design and present:

- **2 Epoch Types**
  - fixed-run
  - fixed-time

- **5 MAB Algorithms**
  - Round-Robin
  - Uniform-Random
  - EXP3.S.1 from Auer et al. 2002 [2]
  - Weighted-Random
  - $\varepsilon$-Greedy

- **5 Belief Metrics**

\[
2 \times (3 + 2 \times 5) = 26 \text{ Scheduling Algorithms}
\]
Belief Metrics

The belief over \((p_i, s_i)\) is a heuristic to estimate the likelihood of yielding a **new** outcome in the next fuzz run of this pair

- Weighted-Random & \(\varepsilon\)-Greedy both bias towards pairs with higher belief

\[
\begin{align*}
\text{RPM} &= \frac{3}{\text{#runs}} \\
\text{EWT} &= \frac{3}{\text{time spent}} \\
\text{DENSITY} &= \frac{\text{bugs}}{\text{#runs}} \\
\text{RGR} &= \text{#bugs} \\
\text{RATE} &= \frac{\text{#bugs}}{\text{time spent}}
\end{align*}
\]

\[
\begin{align*}
\text{No Prior} & \quad \text{With "Bug Prior"}
\end{align*}
\]
The Evaluation Challenge

1. Find *large & representative* data sets
If an algorithm performs well on such data sets, then we gain confidence that it is superior for current practice.

2. How *good* is an algorithm, really?
Is an algorithm that finds 200 bugs in 10 days *good* or *bad*?
⇒ Need to *know* max #bugs that can be found in 10 days, but this is circular! We are trying to solve *this* problem!

3. How to try many algorithms *affordably*?
Yes, we tried *way* more than 26 combinations... 😊
How To Pull This Off

http://s3.amazonaws.com/rapgenius/filepicker%2FgkTHRLQszS3MggKloYA_money.jpg
How To Pull This Off

Step 1. Select two *representative* datasets:

**Intra-Program:** 100 randomly-sampled seeds for FFmpeg
**Inter-Program:** 100 file converters in Debian w/ valid seeds

Step 2. Fuzz *each* of the 200 pairs on EC2 for 10 days—

*48,000* CPU hours (∼5.5 CPU years) later:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#runs</th>
<th>#crashes</th>
<th>#bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intra-program</td>
<td>636,998,978</td>
<td>906,577</td>
<td>200</td>
</tr>
<tr>
<td>Inter-program</td>
<td>4,868,416,447</td>
<td>415,699</td>
<td>223</td>
</tr>
</tbody>
</table>

Table 1: Statistics from fuzzing the two datasets.

Step 3. Build the FuzzSim *replay* system to *simulate* any scheduling algorithm with *no* additional fuzzings
FuzzSim Overview

- Example log entry:
  \((p=FFMPEG, s=a.avi, timestamp=100, run=42, PRNG=17)\)
- Can simulate any schedule using log files
  - Including \textbf{Offline Optimal} (\(\approx\) dynamic prog. for BOUNDED KNAPSACK)

Figure 1: FuzzSim architecture.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Epoch</th>
<th>MAB algorithm</th>
<th>RPM</th>
<th>EWT</th>
<th>Density</th>
<th>Rate</th>
<th>RGR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intra-Program</td>
<td>Fixed-Run</td>
<td>$\epsilon$-Greedy</td>
<td>72</td>
<td>77</td>
<td>87</td>
<td>88</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted-Random</td>
<td>72</td>
<td>84</td>
<td>84</td>
<td>93</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uniform-Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXP3.S.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Round-Robin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fixed-Time</td>
<td>$\epsilon$-Greedy</td>
<td>51</td>
<td>94</td>
<td>51</td>
<td>109</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted-Random</td>
<td>67</td>
<td>94</td>
<td>58</td>
<td>100</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uniform-Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXP3.S.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Round-Robin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fixed-Run</td>
<td>$\epsilon$-Greedy</td>
<td>90</td>
<td>119</td>
<td>89</td>
<td>89</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted-Random</td>
<td>90</td>
<td>131</td>
<td>92</td>
<td>135</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uniform-Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXP3.S.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Round-Robin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter-Program</td>
<td>Fixed-Run</td>
<td>$\epsilon$-Greedy</td>
<td>126</td>
<td>158</td>
<td>111</td>
<td>164</td>
<td>117</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted-Random</td>
<td>152</td>
<td>157</td>
<td>100</td>
<td>167</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uniform-Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXP3.S.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Round-Robin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Comparison between scheduling algorithms.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Epoch</th>
<th>MAB algorithm</th>
<th>#bugs found for each belief</th>
<th>RPM</th>
<th>EWT</th>
<th>Density</th>
<th>Rate</th>
<th>RGR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed-Run</td>
<td>ε-Greedy</td>
<td>72</td>
<td>77</td>
<td></td>
<td></td>
<td>87</td>
<td>88</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Weighted-Random</td>
<td>72</td>
<td>84</td>
<td></td>
<td></td>
<td>84</td>
<td><strong>93</strong></td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>Uniform-Random</td>
<td>72</td>
<td>84</td>
<td></td>
<td></td>
<td>84</td>
<td><strong>93</strong></td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>EXP3.S.1</td>
<td>58</td>
<td>58</td>
<td></td>
<td></td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Round-Robin</td>
<td>74</td>
<td>74</td>
<td></td>
<td></td>
<td>74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intra-Program</td>
<td>ε-Greedy</td>
<td>51</td>
<td>94</td>
<td>51</td>
<td></td>
<td>109</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Weighted-Random</td>
<td>51</td>
<td>94</td>
<td></td>
<td></td>
<td>109</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uniform-Random</td>
<td>94</td>
<td>94</td>
<td></td>
<td></td>
<td>94</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXP3.S.1</td>
<td>58</td>
<td>58</td>
<td></td>
<td></td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Round-Robin</td>
<td>94</td>
<td>94</td>
<td></td>
<td></td>
<td>94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed-Time</td>
<td>ε-Greedy</td>
<td>99</td>
<td>89</td>
<td>89</td>
<td></td>
<td>89</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Weighted-Random</td>
<td>99</td>
<td>89</td>
<td></td>
<td></td>
<td>89</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uniform-Random</td>
<td>99</td>
<td>99</td>
<td></td>
<td></td>
<td>99</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXP3.S.1</td>
<td>72</td>
<td>72</td>
<td></td>
<td></td>
<td>72</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Round-Robin</td>
<td>90</td>
<td>90</td>
<td></td>
<td></td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter-Program</td>
<td>ε-Greedy</td>
<td>126</td>
<td>158</td>
<td>111</td>
<td></td>
<td>164</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Weighted-Random</td>
<td>126</td>
<td>158</td>
<td></td>
<td></td>
<td>164</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uniform-Random</td>
<td>158</td>
<td>158</td>
<td></td>
<td></td>
<td>158</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXP3.S.1</td>
<td>161</td>
<td>161</td>
<td></td>
<td></td>
<td>161</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Round-Robin</td>
<td>158</td>
<td>158</td>
<td></td>
<td></td>
<td>158</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Comparison between scheduling algorithms.

Recommendation 1: Use Weighted Random w/ Rate
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Epoch</th>
<th>MAB algorithm</th>
<th>#bugs found for each belief</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>RPM</td>
</tr>
<tr>
<td>Intra-Program</td>
<td>Fixed-Run</td>
<td>ε-Greedy</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted-Random</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uniform-Random</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXP3.S.1</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Round-Robin</td>
<td>72</td>
</tr>
<tr>
<td>Intra-Program</td>
<td>Fixed-Time</td>
<td>ε-Greedy</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted-Random</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uniform-Random</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXP3.S.1</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Round-Robin</td>
<td>51</td>
</tr>
<tr>
<td>Inter-Program</td>
<td>Fixed-Run</td>
<td>ε-Greedy</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted-Random</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uniform-Random</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXP3.S.1</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Round-Robin</td>
<td>126</td>
</tr>
<tr>
<td>Fixed-Time</td>
<td></td>
<td>ε-Greedy</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weighted-Random</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uniform-Random</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXP3.S.1</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Round-Robin</td>
<td>152</td>
</tr>
</tbody>
</table>

Table 2: Comparison between scheduling algorithms.

Recommendation 2: Use Fixed-Time Campaigns.
Comparison with CERT BFF v2.6

CERT BFF is the state-of-the-art fuzzing framework
  – Supports fuzzing **one** program w/ **multiple** seeds
  – Varies mutation ratio **online**
  – **Fixed-run** epochs
  – **Weighted-Random** MAB algorithm
  – use **Density** (#bugs/#runs) as belief

**Fixed-time** Weighted-Random **Rate** finds on average 1.5x more bugs in our datasets (at a fixed mutation ratio)
Intra: FFMPEG Dataset

Inter: File Converters Dataset
Future Work

Vary mutation ratio
• $m$ mutation ratios $\Rightarrow m$-fold cost increase

Online bug triage
• triage time is currently being discounted

Other program testing techniques
• black-box generational (grammar-based) fuzzing?
• concolic execution?
Summary

MAB

Not Enough!

Start
Summary

Start

MAB

WCCP

Not Enough!
Summary

Start

MAB → WCCP → Not Enough! → NFL
Summary

Rule of Three

WCCP

MAB

Not Enough!

NFL

Start
Summary

Algorithm Design
Rule of Three

MAB
WCCP
Not Enough!
NFL
Start
Summary

Start

MAB

WCCP

Not Enough!

NFL

Algorithm Design

Rule of Three

Open Science

Not Enough!
FuzzSim[1]: Black-box Fuzzing Simulator

Black-box mutational fuzzing is an effective, albeit simple, way to find bugs in software. FuzzSim is a black-box fuzzing scheduling simulator that can test various seed selection algorithms. The main purpose of this tool is to ask the following question: given a set of programs and seeds, what is the best way to schedule fuzzing of the programs on the seeds to maximize the number of unique bugs found within a fixed amount of time?

Installation

Download FuzzSim

FuzzSim Download Page

Download files from the below:

- fuzzsim.0.1.tgz (MD5: 0420abf52d6d74014fac8607fedc99ea)
- fuzzdata-intra.tgz (MD5: 3dd151054c01e14d39e25eda0e48dd35)
- fuzzdata-inter.tgz (MD5: 0a3ec7c2b1550812a87d517cbf2e82c2)

After downloading:

$ tar xvzf fuzzsim.0.1.tgz
$ cd fuzzsim-0.1
$ ./configure
$ make

Quick Start

To see the usage, type: