Chemotactic Test Case Recombination for Large-Scale Fuzzing

Konstantin Böttinger

Fraunhofer Institute for Applied and Integrated Security,
85748 Garching, Germany
E-mail: konstantin.boettinger@aisec.fraunhofer.de

Received 29 August 2017; Accepted 1 September 2017;
Publication 26 September 2017

Abstract

We present a bio-inspired method for large-scale fuzzing to detect vulnerabilities in binary executables. In our approach we deploy small groups of feedback-driven explorers that guide colonies of high throughput fuzzers to promising regions in input space. We achieve this by applying the biological concept of chemotaxis: The explorer fuzzers mark test case regions that drive the target binary to previously undiscovered execution paths with an attractant. This allows us to construct a force of attraction that draws the trailing fuzzers to high-quality test cases. By introducing hierarchies of explorers we construct a colony of fuzzers that is divided into multiple subgroups. Each subgroup is guiding a trailing group and simultaneously drawn itself by the traces of their respective explorers. We implement a prototype and evaluate our presented algorithm to show the feasibility of our approach.

Keywords: Fuzzing, Random Testing, Vulnerability Detection.

1 Introduction

The constant proliferation of serious software vulnerabilities states a major threat against the technologies that surround us. As critical infrastructures of modern society increasingly depend on the functioning of software we face the

Journal of Cyber Security, Vol. 5_4, 269–286.
doi: 10.13052/jcsm2245-1439.542

severe challenge to harden our core systems. This often boils down to finding vulnerabilities before the adversary does. Undisclosed security-critical bugs known as *zero-days* will continue to emerge on the surface of black markets to attract players of a variety of backgrounds. A common strategy to decrease the risk of being successfully attacked is to increase the effort it takes to compromise our assets. This points us to research automated methods that allow us to systematically perform vulnerability analysis of software. The nowadays most effective way to proceed in this direction is random testing of target binaries, also called fuzzing. State-of-the-art fuzzing frameworks all share one overall goal: Generating and pitching suitable program inputs (also called test cases) into the target in order to eventually trigger an exploitable bug. For suchlike bug hunting there is a straight forward track: The more input we generate to test a binary target the more code coverage we achieve during program execution and the more likely we will find what we are looking for. This results in parallel large-scale testing by running distributed fuzzer instances on a computer cluster.

However, state-of-the-art in distributed large-scale fuzzing basically reduces to pure parallelization. Recent research focuses on advancing single fuzzers [2, 11, 15] and optimal scheduling of fuzzers, test case corpora, and targets during fuzzing campaigns [19]. But how can we optimize the interaction between fuzzers? How can we transform a cluster of isolated fuzzers into a colony that works together and collectively adapts to the binary under test?

Inspired by biology two observations in particular guide our research presented in this paper: Colonies with dedicated explorers and the concept of chemotaxis.

1.1 Colonies with Explorers

Several species such as honeybees, ants, rats, and bats reveal dedicated exploring behavior of colony individuals that primarily function as scouts. Investigation of the environment by just a small fraction of explorers seems to be an efficient way for some colonies to gain information regarding the surrounding territory. In case the explorer found an interesting spot (for example a source of food during foraging) it reports its findings back to the colony. The famous dance of the honeybees [13] is just one example for this behavior. Hence we define dedicated subgroups of explorer fuzzers that guide higher throughput worker fuzzers. In fact, we can divide modern fuzzing frameworks into two categories, namely (1) feedback fuzzers that instrument

their targets in order to gain runtime information during program execution and (2) black-box fuzzers that are blind to what happens during execution and only see program crashes in case of a triggered bug. While fuzzers of the first category (including white-box and evolutionary fuzzers) are relatively slow they nowadays achieve similar levels of code coverage compared to traditional fast executing black-box fuzzers. Both categories, the relatively slow feedback driven explorers as well as the fast and efficient black-box worker fuzzers have their right to exist in modern fuzzing campaigns and both provide comparable results. Inspired by colony behavior in biology, is there a way to combine the explorer sight into runtime (gained by dynamic instrumentation) with the speed of black-box worker fuzzers? How can we achieve guidance by the explorers and transfer information to the blind black-box fuzzers? This brings us to the second observation found in biology.

1.2 Chemotaxis

Regardless if we look at bacteria, mold fungus, termites, ciliates, or algae, all those species have one thing in common: They make use of chemical substances to transmit information between individuals of the colony in order to trigger collective behavior. The movement of organisms responding to chemical stimuli is called *chemotaxis*. Positive chemotaxis causes the individuals to move towards regions of higher concentration of an attractant. Ant colonies [16] coordinating their foraging behavior using attracting trail pheromones impressively illustrate the power of chemo-taxis. Can we mimic social behavior of biological colonies using the concept of chemotaxis?

In this paper, we construct an algorithm for distributed large-scale fuzzing that equips feedback-driven explorer fuzzers with the ability to attract high throughput fuzzers by marking regions in the input space with an attractant. First, we develop the main idea on a single subgroup of explorers guiding a single subgroup of workers: By controlling the attractant concentration among promising test case regions the *seeing* feedback-driven explorers guide the colony of *blind* (but fast) black-box fuzzers in order to maximize code coverage. Second, we generalize this approach to multiple hierarchies of fuzzers: We introduce multiple hierarchies of explorers by further subdividing our scouts according to their overall test case throughput.

In summary, we make the following contributions:

- We introduce a novel method for distributed large-scale fuzzing in computer clusters based on the biological concept of chemotaxis in order to maximize coverage of execution paths in the target under test.

- We construct a mechanism for distributing attractants in input space and define the resulting force field of attraction exerted on high throughput fuzzers.
- We implement and evaluate our presented algorithm to show the feasibility of our approach.

The remainder of this paper is organized as follows. In Section 2 we discuss related work. In Section 3 we present our algorithm for guided fuzzing. We implement and evaluate our approach in Section 4 and discuss properties, modifications, and expansions of the proposed algorithm in Section 5. The paper concludes with a short outlook in Section 6.

2 Related Work

This paper is an extension of *Guiding a Colony of Black-box Fuzzers with Chemotaxis* [3]. The fuzzing discipline has evolved from its very beginnings [9] in 1990 to an active area of research providing advanced testing frameworks [1, 2, 11, 14, 15]. Beyond generational (format-aware) and mutational (format-blind) fuzzers we can generally distinguish between feedback-driven and black-box fuzzers. The first category makes use of instrumentation frameworks (such as Pin, Valgrind, DynamoRIO, Dyninst, DTrace, QEMU, and the like) to gain detailed information regarding program execution. For example, the fully self-adaptive fuzzer presented in [2] and [1] adjusts its parameters according to code coverage feedback from dynamic instrumentation of the target binary. Further, evolutionary and white-box fuzzers such as AFL, Driller (enhancing AFL with symbolic execution), EFS, Sage, Choronzon, Honggfuzz, libFuzzer, Kasan, Kcov, and BFF belong to this category. While binary instrumentation provides advanced test case generation based on runtime feedback, it comes with relatively high overhead (see [8] for a benchmark) and resulting moderate test case throughput. In contrast, black-box fuzzers (such as zzuf, Peach, and Radamsa) pitch test cases into the targeted binary without gathering feedback from dynamic instrumentation, which makes them significantly faster compared to feedback-driven fuzzers. We refer to [18] and [17] for a detailed and comprehensive account on random testing. Attempts to optimize black-box fuzzing [12, 19] often neglect the distributed nature of parallel large-scale fuzzing campaigns. Up to now it is unclear how to effectively organize and guide a large set of black-box fuzzers in order to maximize code coverage.

3 Guided Fuzzing

In this section we present the overall algorithm for collective random testing of binary targets by a colony of fuzzers guided by dedicated explorers.

Our final goal is to optimize massively parallel large-scale fuzzing in computer clusters to find vulnerabilities in a binary target. Let $\dot{F}$ denote the set of feedback-driven fuzzers and F the set of fast non-instrumenting black-box fuzzers, respectively. Inspired by biology we refer to $\dot{F}$ as the explorers and to F as the worker individuals. The explorers receive information from dynamic instrumentation (e.g. regarding code coverage) and therefore see what happens during execution of the target. As motivated in the introduction we present a guidance mechanism that enables the seeing explorers to transfer information to the blind black-box worker fuzzers by mimicking the concept of chemotaxis. We achieve this by constructing explorer traces in the target input space to attract the workers F. In the following we first formalize how to construct such traces and then define the force of attractivity and resulting colony movement analog to chemotaxis.

3.1 Attractant Trace Generation

We assume the inputs of the target binary under test to be bit strings of length N and denote the input space as $\mathcal{I} = \{0, \ldots, 2^N\}$. Each fuzzer provides a corpus $C \subset \mathcal{I}$ of current test cases. During a fuzzing campaign the individual fuzzers constantly update their set of current test cases, which generates a trace in input space for each fuzzer. Inspired by chemotaxis we want the explorers to leave behind an attractant on their way through input space. More formally, assume we have n_E explorers $\dot{F}^i$ each starting with a set of seed inputs $\dot{C}_{t_0}$. After some time t_1 of fuzzing, each $\dot{F}^i$ has updated its initial seed inputs to the current working corpus $\dot{C}^i_{t_1}$. During the fuzzing campaign, the $\dot{F}^i$ generate corpora $\dot{C}^i_{t_1}, \dot{C}^i_{t_2}, \ldots, \subset \mathcal{I}$. To construct a trace of $\dot{F}^i$ in $\mathcal{I}$, we calculate the center of each intermediate corpus of test cases and then mark these centers with an attractant.

3.1.1 Trace generation

We first need to define the center of a corpus $\dot{C} \subset \mathcal{I}$ of test cases. Instead of the arithmetical mean we are interested in the bit string $\hat{c}$ that is most similar to all of the strings in $\dot{C}$. This choice is justified by the following example: Consider a corpus $\dot{C}$ of bit strings each of which respects the input format of a given target. The arithmetical mean of $\dot{C}$ might be a bit string with corrupted file format including wrong headers and metadata. Therefore, we define the

center $\hat{c}$ of $\dot{C}$ to be the string of length N that coincides with the majority of inputs in $\dot{C}$ in each bit position. The complexity of this calculation is bound by $\mathcal{O}(n^2)$. Periodically extracting the corpus of current test cases of the explorers $\dot{F}_i$ and calculating their centers $\hat{c}^i$ yields a trace

$$T^i := (\hat{c}^i)_{\tau \in \mathcal{T}} := (\hat{c}^i_{\tau_0}, \hat{c}^i_{\tau_1}, \hat{c}^i_{\tau_2}, ...) \tag{1}$$

where $\mathcal{T} = \{\tau_0, \tau_1, \tau_2, ...\}$ indicates the extraction times during the fuzzing campaign.

3.1.2 Attractant spraying

Now that we have a trace T^i for each explorer $\dot{F}^i$ we can spray this trace with an attractant in order to draw the black-box worker fuzzers F^i. In this step we augment each center of trace T^i with an attractant concentration that decreases over time. Naturally, the most recently generated corpus of an explorer should have a higher concentration of attractant than a previously generated corpus. This correlates to diffusivity and resulting fall in concentration of real chemical attractants in biological chemotaxis. To realize this we define a monotonically decreasing function $f\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ to yield the sprayed trace

$$\bar{T}^i := \big((\hat{c}^i_{\tau_0}, f(t-\tau_0)), (\hat{c}^i_{\tau_1}, f(t-\tau_1)), ...\big) \tag{2}$$

$$= \big(\hat{c}^i_{\tau}, f(t-\tau)\big)_{\tau \in \mathcal{T}} \tag{3}$$

for $i = 1, \ldots, n_E$, where t denotes the current time of the fuzzing campaign. In our implementation (see Section 4) we generate the sprayed traces periodically after a fixed amount of time so we can assume without loss of generality the discrete time indexing set $\mathcal{T} = \mathbb{N}$. The choice of spraying function f determines attractant concentration of explorer traces over time. If f decays fast, the explorers will leave only a short attracting trace in time, whereas a slower decay yields longer attracting traces. To avoid persistent attraction of already extensively explored regions we must construct f such that attractant concentration decays to zero after some time, i.e. $\lim_{t\to\infty} f(t) = 0$. Moreover to keep computing complexity in subsequent steps low we define f to map identical to zero after time t_z, i.e.

$$\forall t \geq t_z : f(t) = 0. \tag{4}$$

We discuss different choices of f and resulting attracting behavior of the black-box worker fuzzers in Section 4.

3.2 Positive Chemotaxis

Next, we construct an attraction mechanism for the sprayed traces $\bar{T}^i$ left behind by the explorers $\dot{F}^i (i = 1, \ldots, n_E)$. The traces $\bar{T}^i$ should attract the black-box worker fuzzers $F^j (j = 1, \ldots, n_W)$. Again, we refer to the position of an $\dot{F}^i$ as the center $\hat{c}^i \in I$ of its current corpus of test cases $\dot{C}^i \subset \mathcal{I}$. Mathematical modeling of chemotaxis usually makes use of partial differential equations [7, 10], which describes movement and emerging spatial pattern formation accurately in terms of biology. Since we are more interested in computational efficiency than in biological accuracy, instead of simulating our colony of fuzzers with partial differential equations we define a lightweight attracting function g that acts as a force of gravity on the corpora of black-box fuzzers F^j. While f (as described above) determines the distribution and decay of attractant concentration of traces in input space $\mathcal{I}$, g determines the force of attraction dependent on both the distance and the concentration of the attractant. Therefore, g: $\mathbb{R}^2_{\geq 0} \rightarrow [0, 1]$ is a function of two variables. We discuss and evaluate different choices of g and resulting attracting behavior in Section 4.

To determine the force of attraction that an explorer trace exerts on a black-box worker fuzzer $F^j (1, \ldots, n_W)$ we need the attractant concentration of its trace as well as the distance between centers of the trace $\bar{T}^i$ and the corpus C^j of F^j. The individual centers $\hat{c}^i \in \mathcal{I}$ of explorer traces $\bar{T}^i$ have already assigned a concentration as given in Equation (2). For the metric we choose Hamming distance δ in $\mathcal{I}$: Two bit strings $x = (x_1, \ldots, x_N)$ and $x' = (x'_1, \ldots, x'_N)$ then have distance

$$\delta(x, x') := |\{j \in 1, \ldots, N | x_j \neq x'_j\}|. \tag{5}$$

For a single test sample $x \in \mathcal{I}$ function g then gives the force of attraction $a \in [0, 1]$ that a center $\hat{c}^i$ exerts on *x* at time *t*:

$$a = g(f(t - \tau_i), \delta(\hat{c}^i, x)). \tag{6}$$

Now that we have defined the force a of attraction on $x \in \mathcal{I}$, we construct a movement of x analog to chemotaxis. We can move x towards $\hat{c}^i$ in the Hamming distance if we flip bits in x to match the corresponding bits in $\hat{c}^i$. Therefore, let $a \in [0, 1]$ be the fraction of bits in x that we flip to match bit string $\hat{c}^i$, where the bit positions to be flipped are randomly chosen among $1, \ldots, N$. For example, $a = 1$ causes all bits in the mutated version x' of x to match those in $\hat{c}^i$ resulting in $\delta(\hat{c}^i, x') = 0$. An attracting force of $a = 0$ on the other hand leaves x unchanged.

Finally, an explorer $\dot{F}^i$ draws a black-box worker F^j by letting its trace $\bar{T}^i$ (i.e. all centers $\hat{c}^i$ of its trace with nonzero attractant concentration) simultaneously attract all test cases in the current corpus $C^j \subset \mathcal{I}$ of F^j.

3.3 Guided Fuzzing Algorithm

The algorithm for guiding a dedicated colony of black-box fuzzers is depicted in Figure 1.

Input: $f,\ g,\ n_E,\ n_W,\ t'$

for $i = 1, ..., n_E$ **:**
    $\dot{C}^i_{t_0} \leftarrow \text{Seed}()$
    $\dot{F}^i \leftarrow \text{Initialize}(\dot{C}^i_{t_0})$
for $j = 1, ..., n_W$ **:**
    $C^j_{t_0} \leftarrow \text{Seed}()$
    $F^j \leftarrow \text{Initialize}(C^j_{t_0})$

do:
    for $i = 1, ..., n_E$ **:**
        $\dot{C}^i \leftarrow \text{Corpus}(\dot{F}^i)$
        $\hat{c}^i \leftarrow \text{Center}(\dot{C}^i)$
        $T^i \leftarrow \text{Trace}(\hat{c}^i)$
        $\bar{T}^i \leftarrow \text{Spray}(T^i, f)$

        for $j = 1, ..., n_W$ **:**
            $C^j \leftarrow \text{Corpus}(F^j)$

            for $\hat{c}$ **in** $\bar{T}^i$ **:**
                $C^j \leftarrow \text{Attract}(\hat{c}, C^j, g)$

    for $j = 1, ..., n_W$ **:**
        $F^j \leftarrow \text{Initialize}(C^j)$

    $\text{Fuzz}(t')$

    while (true)

Figure 1 Algorithm for guided fuzzing with input functions f, g and parameters n_E, n_W and t'.

The first two loops initialize the n_E explorers as well as the n_W black-box worker fuzzers. The seed input corpora $C^i_{t_0}, C^j_{t_0} \subset \mathcal{I}$ are sets of bit strings of length *N*. They can be generated randomly or alternatively may originate from a previous fuzzing campaign, but we don't assume any constraints on them (e.g. validity regarding the input format).

After initialization phase we enter the process of attractant trace generation, positive chemotaxis, and fuzzing. This main iteration is repeated until a tester stops the fuzzing campaign. The first loop in the main iteration extracts the test case corpora $\dot{C}^i$ of explorers $\dot{F}^i$, calculates their centers $\hat{c}^i$, appends them to the respective traces T^i and sprays the traces with the attractant according to the choice of f. Next, the centers $\hat{c}^i$ of the sprayed traces $\bar{T}^i$ attract all n_W test case corpora C^j of black-box worker fuzzers F^j. This force of attraction results in positive chemotaxis and is regulated by function g as given in Equation (6).

Next, we reinitialize the black-box worker fuzzers F^j with the updated respective test case corpora and let the whole colony of fuzzers perform random testing for a fixed amount of time t'.

Regarding computational complexity of the proposed algorithm we constructed each step to be efficient. The cost of center calculation is bound by $\mathcal{O}(n^2)$, which is tractable considering that we only process the working set of current test cases of an explorer. Spraying the traces T^i with an attractant as indicated by Equation (2) is a lightweight operation on a two-dimensional array that holds for each calculated center $\hat{c}^i$ the corresponding attractant concentration given by f. Calculation of the force of attraction as defined in Equation (6) requires computing the Hamming distance, which requires low overhead. Further, we carefully bound the time for computing the force of attraction that the explorer traces $\bar{T}^i (i = 1, \ldots, n_E)$ exert on the corpora $C^j (j = 1, \ldots, n_W)$ by limiting the number of centers with nonzero attractant concentration, as guaranteed by Equation (4). Finally, we process these steps that lead to repositioning of the corpora of F^j only once for each time interval t'. During the fuzzing step (denoted by $Fuzz(t')$ in Figure 1) all fuzzer instances of both the explorers and the black-box workers run unaffected.

3.4 Explorer Hierarchies

Next, we generalize the algorithm for guided fuzzing as depicted in Figure 1 to multiple hierarchies of explorers by further subdividing our scouts according to their overall test case throughput. This further distinction is motivated by the

observation that there is a spectrum between feedback-driven fuzzers and black-box fuzzers. For example, test frameworks enhanced with symbolic execution functionality (such as [4–6]) are computationally more complex than more efficient evolutionary fuzzers (see [14] for a recent benchmark), but both categories make use of feedback from binary instrumentation. Further, the efficiency of black-box fuzzers depends on the targeted input format: In some cases (of complex input formats) it is useful to deploy a grammar-based fuzzer that generates mostly valid test cases in order to feed them as seed into feedback-driven fuzzers. In order to cover such situations with our approach, we need to define multiple hierarchies of fuzzers. This results in a colony of fuzzers divided into multiple subgroups each guiding a trailing group and simultaneously drawn itself by the traces of their respective explorers.

To achieve this, we divide the whole fuzzing colony into n_K classes $\mathcal{F}^i$ and define a set of arrows A between those classes. An arrow $(\mathcal{F}^i, \mathcal{F}^j)$ indicates that fuzzers of class $\mathcal{F}^i$ function as explorers for fuzzers in the class $\mathcal{F}^j$, i.e. fuzzers in F^i attract fuzzers in $\mathcal{F}^j$ according to the guidance algorithm as depicted in Figure 1. This yields the directed graph

$$\mathcal{G} = \left(\bigcup_{i \in \{1,\ldots,n_K\}} \mathcal{F}^i, A \right). \tag{7}$$

For example, setting $n_K = 2$ yields the previously described situation of a single colony of explorers $\dot{\mathcal{F}} = \cup_{i \in \{1,\ldots,n_E\}} \dot{F}^i$ attracting a single colony of workers $\mathcal{F} = \cup_{j \in \{1,\ldots,n_W\}} F^j$, i.e.

$$\mathcal{G}_2 = \left(\dot{\mathcal{F}} \cup \mathcal{F}, (\dot{\mathcal{F}}, \mathcal{F}) \right). \tag{8}$$

Such a graph definition is especially useful when distributing fuzzing instances on actual computer clusters. If the fuzzing campaign is executed by a heterogenous hardware infrastructure the graph should be adapted according to bandwidth and latency of the respective interconnection network. In particular, if two fuzzing classes in the graph are connected with an arrow, they are suited to be placed in the same high bandwidth and low latency interconnection network. Vice versa, if two fuzzer classes have a large geodesic distance in the graph (i.e. a relatively high number of arrows in the shortest path between them), they may be distributed accordingly in computing clusters that are interconnected with lower bandwidth and higher latency.

3.5 Choices for f and g

In choosing the spraying function f and the attraction function g we are guided by the following considerations. f determines the actual attracting fraction of the explorer traces. As determined by Equation (2) a fast decay of f leads to short attracting traces and vice versa. In the extreme, f distributes the attractant nowhere on the explorer trace except on the most recently computed center (corresponding to $f(0) \neq 0$ and $f(t) = 0$ for all $t > 0$). For simultaneous strong force of attraction such a choice is almost equivalent to direct corpus synchronization. However, we want the black-box workers $F^j (j = 1, \ldots, n_W)$ to be guided along the explorer paths for two reasons: Close proximity to actually all regions roamed by the explorers and enough time for black-box worker exploration. To be more precise, for large periods t' of pure fuzzing (as indicated in Figure 1) corpus extraction provides only discrete snapshots of current explorer positions in time. During fuzzing for time t' the workers also diffuse their corpora through input space. Too high attractivity of the most currently generated explorer corpus would tend to ignore fuzzing the whole path between extracted corpus snapshots. Since we want the black-box workers to follow the explorer paths as closely as possible while simultaneously give them enough time to generate corner cases not discovered by the explorers, we distribute the mass of f accordingly. As shown in our evaluation in Section 4 we achieved good results with different Gaussian functions for f. Regarding the attracting function g in Equation (6) we borrow from the law of gravity and propose higher attraction forces for higher concentrations and closer distances. We implement a sigmoid function made of two logistic functions in Section 4.

4 Implementation and Evaluation

To show the feasibility of our approach we implemented a prototype of the algorithm as depicted in Figure 1 with one dedicated group of explorers guiding a colony of high throughput worker fuzzers. In this section we first present our choices for functions f and g, and subsequently evaluate our method.

For spraying function f we implemented Gaussian functions

$$f(t) = \begin{cases} c_1 e^{-\frac{(t-c_2)^2}{2c_3^2}} & 0 \leq t < t_z \\ 0 & t \geq t_z \end{cases} \tag{9}$$

parameterized by $c_1, c_2, c_3 \in \mathbb{R}_{>0}$. While c_1 determines the total amount of attractant, c_3 controls the decomposition rate of attractant concentration on the traces T^i. A nonzero value of $c_2 > 0$ translates to an attractant that unfolds its full attractive potential only with a time delay, but we set $c_2 = 0$ for the following benchmarks.

Function g assigns the force of attraction dependent on attractant concentration and distance to the attractant. Shorter distance and higher concentration should result in stronger attraction. We implemented g: $\mathbb{R}^2_{\geq 0} \to [0, 1]$

$$g(f, \delta) \left((1 + a_1 e^{a_3 - a_2 f})(1 + d_1 e^{d_2 \delta - d_3}) \right)^{-1}, \tag{10}$$

where $(f, \delta) = (f(t - \tau_i), \delta(\hat{c}^i, x))$ denote attractant concentration and distance, respectively, and $a_1, a_2, a_3 \in \mathbb{R}_{>0}$.

As testing target we chose the command line tool `djpg` for decompressing `JPEG` files to image files (in `BMP` and `GIF` format). All explorers are slow moving versions of the fuzzer presented in [2] and [1]. We initialized both the explorers and black-box workers with seed corpora containing image files of size 100 kB. Then we measured the distance δ between most recently generated centers (corresponding to the end of trace $\bar{T}$) of a selected explorer and the respective corpus centers of a successfully attracted black-box worker.

Figure 2 depicts attraction behavior of a single explorer ($n_E = 1$). After each of the first 100 iterations of the *do-while* loop of our algorithm (as depicted in Figure 1) we indicate distance δ on the z-axis. After 100 iterations we stop the fuzzing campaign and increase the force of attraction by increasing d_2. After 10 fuzzing campaigns ($d_2 = 4, \ldots, 14$) we receive the surface depicted in Figure 2. For strong forces of attraction (corresponding to high values of d_2) the single explorer successfully attracts all workers and reduces the mean distance δ from averaged 400 kbit to 330 bit. Weak forces of attraction (corresponding to low values of d_2) do not lead to attraction. This is due to the diffusivity of worker corpora in input space: With a black-box fuzzer mutation rate of $r = 4 \times 10^{-5}$ the workers diffuse their test case corpora stronger than the explorer attracts them.

In a second experiment we increase the number of explorers to $n_E = 5$ as well as the black-box mutation ratio to $r = 8 \times 10^{-4}$. The resulting benchmark is depicted in Figure 3. Analog to the previous setting we measure distance δ (on the z-axis) in each of the first 100 iterations for 10 fuzzing campaigns with respectively increasing force of attraction $d_2 = 4, \ldots, 14$. After successful initial attraction the distance δ reaches an equilibrium state depending on the force of attraction. We can successfully reduce the distance

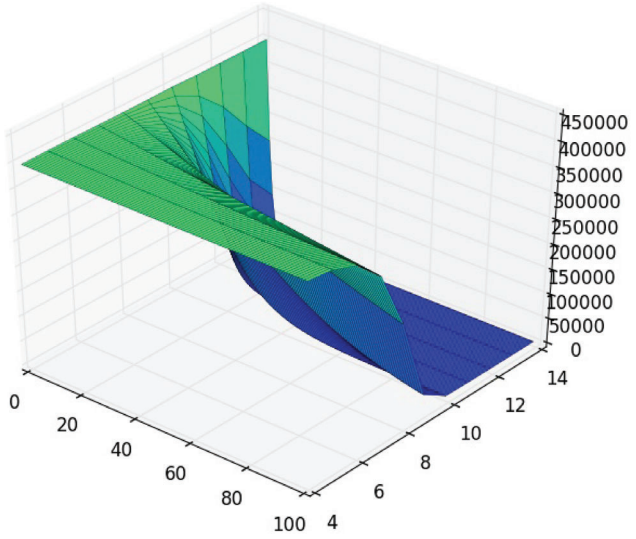

Figure 2 Attraction of a single explorer ($n_E = 1$) within the first 100 iterations, resulting in a decrease of δ from averaged 400 kbit down to 330 bit, where mutation ratio for the measured black-box fuzzer is $r = 4 \times 10^{-5}$. Increasing d_2 from 4 to 14 causes a significant stronger attraction.

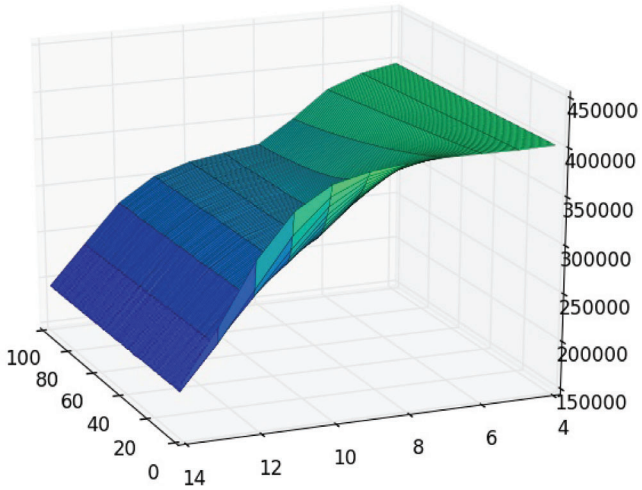

Figure 3 Attraction for $n_E = 5$ within the first 100 iterations and increasing $d_2 = 4, \ldots, 14$, causing a decrease of δ from averaged 400 kbit down to 190 kbit, where mutation ratio is $r = 8 \times 10^{-4}$.

and guarantee proximity of the workers to the explorer traces by increasing the force of attraction. The equilibrium of distance between an attracted worker and its explorer guide is caused by three antagonizing forces: Attraction by its guide, attraction by all competing explorers, and mutation ratio of the worker. Increasing the force of attraction simultaneously for all explorers binds the worker further to its guide (because of the sigmoid form of g as defined in Equation (10) and additionally overcomes even high mutation ratios.

5 Discussion

In this section we discuss characteristics, possible modifications, and expansions of our approach.

As shown in our evaluation once a black-box worker fuzzer has joined an explorer it will remain there most probably for the rest of the fuzzing campaign. However, the attraction mechanism of the group of explorers after each period of time t' brings in a small fraction of valuable fresh input from the surrounding explorers. This is due to the construction of our attraction mechanism as described in Section 3.2, where actual attraction is lowering the Hamming distance by flipping bits to match the attracting center (which is the test case that matches the majority of test cases of a current explorer corpus regarding the bit string). Therefore, we achieve mixing of test cases between essentially isolated explorers.

Further, attraction of the trace of an explorer as sprayed by f according to Equation (2) guarantees optimal post-processing of input regions toughed by the explorers. As discussed in Section 3 trace attraction gives two vital advantages compared to simple corpus synchronization: Close proximity to actually all regions roamed by the explorers and enough time for black-box worker exploration. Since black-box workers are significantly faster and provide different mutation engines, their concentration around explorer traces often reveals new side paths and corner cases that the explorers did not discover.

We put much emphasis on out-of-the-box deployment of existing fuzzing frameworks to avoid any possibly time-consuming or (in case of closed source fuzzers) impossible modifications. However, access to information inside the explorer fuzzers would allow us to adapt attraction behavior for each explorer individually. Our presented spraying mechanism as determined by function f in Equation (2) treats each explorer equally: It assumes the explorer has found a region of quality test cases (e.g. regarding code coverage), sprays the corpus center, and lets the concentration descent over time. If an explorer

discovers significantly more new basic blocks than all other explorers, we should be able to assign a higher force of attraction to respective test cases. In other words, comparing the numbers of newly discovered basic blocks found by the individual explorers would allow us to allocate higher attractant concentrations to centers of higher quality corpora, enabling strongest attraction to the currently best performing explorer. Further, we could introduce a repellent inducing negative chemotaxis for test cases that for example consume too much time to process or enter code regions that are not relevant for testing.

So far we do not provide any feedback from the black-box fuzzers back to the explorers. This is motivated by the nature of basically blind black-box fuzzers which do not obtain any information from the targeted binary during runtime, except a program crash. But especially this crash information could be used to mark the corresponding test case as attractive. Such modifications could improve the overall fuzzing campaign.

6 Conclusion

Inspired by insect and animal colonies that reveal a rich diversity of scouts and explorers we introduce the first framework for large-scale random testing of binary executables based on the concept of chemotaxis. In order to maximize coverage of execution paths in the target under test we draw fast and efficient (but blind regarding runtime information) black-box workers to regions in input space discovered by feedback-driven explorers. We realize this by constructing a mechanism for distributing attractants in input space and defining the resulting force field of attraction exerted on black-box fuzzers. This approach combines the best of both worlds: The sight into runtime information from dynamic instrumentation by the explorers and the speed of black-box worker fuzzers. Next, we generalize this approach to multiple hierarchies of fuzzers to capture their attraction network in a graph. Such a graph definition is especially useful when distributing fuzzing instances on actual computing clusters, as we can adjust the graph of attraction to the hardware infrastructure. We show the feasibility of our approach by evaluating it on a real-world target with different parameter settings. Further, we discuss modifications and expansions of our algorithm. Especially customized testing frameworks would allow us to distribute attractant concentration significantly more fine-grained, which probably results in faster code coverage and is subject to future work.

References

[1] Böttinger, K. (2016). Fuzzing binaries with Lévy flight swarms. *EURASIP J. Inform. Sec*. 2016.

[2] Böttinger, K. (2016). "Hunting bugs with Lévy flight foraging," in *Proceedings of the IEEE Security and Privacy Workshops*, San Jose, CA, 111–117.

[3] Böttinger, K. (2017). "Guiding a colony of black-box fuzzers with chemotaxis," in *Proceeding of the IEEE Symposium on Security and Privacy Workshops*, San Jose, CA.

[4] Böttinger, K., and Eckert, C. (2016). "Deepfuzz: triggering vulnerabilities deeply hidden in binaries," in *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, (New York, NY: Springer), 25–34.

[5] Cadar, C., Ganesh, V., Pawlowski, P. M., Dill, D. L., and Engler, D. R. (2008). "EXE: automatically generating inputs of death." in *Proceedings of the 13th ACM Transactions on Information and System Security (TISSEC)*, eds A. Juels, Wright, and S.D.C. di Vimercati (New York, NY: ACM).

[6] Godefroid, P., Levin, M. Y., and Molnar, D. (2012). SAGE: whitebox fuzzing for security testing. *Commun. ACM* 55, 40.

[7] Hillen, T., and Painter, K. J. (2009). A users guide to pde models for chemotaxis. *J. Math. Biol*. 58, 183–217.

[8] Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., et al. (2005). "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '05*, (New York, NY: ACM), 190–200.

[9] Miller, B. P., Fredriksen, L., and So, B. (1990). An empirical study of the reliability of unix utilities. *Commun. ACM* 33, 32–44.

[10] Painter, K. J., and Hillen, T. (2002). Volume-filling and quorum-sensing in models for chemosensitive movement. *Can. Appl. Math. Quart.* 10, 501–543.

[11] Rawat, S., Jain, V., Kumar, A., Cojocar, L., Giuffrida, C., and Bos, H. (2017). "Vuzzer: application-aware evolutionary fuzzing," in *Proceedings of the Network and Distributed System Security Symposium (NDSS 2017)*, San Diego, CA.

[12] Rebert, A., Cha, S. K. Avgerinos, T., Foote, J., Warren, D., Grieco, G., and Brumley, D. (2014). "Optimizing seed selection for fuzzing,"

in *Proceedings of the USENIX Security Symposium*, (Berkeley, CA: USENIX Association), 861–875.

[13] Riley, J. R., Greggers, U., Smith, A. D., Reynolds, D. R., and Menzel, R. (2005). The flight paths of honeybees recruited by the waggle dance. *Nature* 435, 205–207.

[14] Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., et al. (2016). "SOK: (state of) the art of war: offensive techniques in binary analysis," in *Proceedings of the IEEE Symposium on Security and Privacy*, San Jose, CA, 138–157.

[15] Stephens, N., Grosen, J., Salls, C., Dutcher, A., Wang, R., Corbetta, J., et al., (2016). "Driller: augmenting fuzzing through selective symbolic execution," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA.

[16] Sumpter, D. J. T., and Beekman, M. (2003). From nonlinearity to optimality: pheromone trail foraging by ants. *Animal Behav.* 66, 273–280.

[17] Sutton, M., Greene, A., and Amini, P. (2007). *Fuzzing: Brute Force Vulnerability Discovery*, 1st Edn. Boston, MA: Addison-Wesley Professional.

[18] Takanen, A., DeMott, J., and Miller, C. (2008). *Fuzzing for Software Security Testing and Quality Assurance*. Norwood, MA: Artech House, Inc.

[19] Woo, M., Cha, S. K., Gottlieb, S., and Brumley, D. (2003). "Scheduling black-box mutational fuzzing," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS)*, (New York, NY: ACM), 511–522.

Biography

Konstantin Böttinger joined the Fraunhofer Institute for Applied and Integrated Security (AISEC) as research associate in 2011 and is currently working in the Department for Product Protection and Industrial Security. His research focuses on cryptographic protocols, software testing, and anomaly detection. Prior to joining the Fraunhofer AISEC, Konstantin Böttinger studied mathematics and physics at Heidelberg University, where he finished with a Diploma in mathematics.