A study on Evolutionary Fuzzing

Bofin Babu
2013H313085H
BITS G540 RESEARCH PRACTICE
Instructor: Prof. Chittaranjan Hota

December 3, 2015

Abstract

Fuzzing is a software testing method in which a series of random input is given to the software under test until it gets crashed. The purpose of this is to find coding errors or security vulnerabilities in the software. Evolutionary fuzzing is a new method of fuzzing where evolutionary algorithms are used to generate small, better set of test cases. Therefore evolutionary fuzzing is more effective and efficient fuzzing technique than bind fuzzing.

1 Introduction

Evolutionary fuzzing is a fuzzing technique which makes use of genetic algorithms to generate smart test data. In this section I explain Genetic Algorithms and Fuzzing. In the following sections I will describe the process of evolutionary fuzzing and cover a detailed section about using evolutionary fuzzing to detect XSS vulnerabilities of the web.

1.1 Genetic Algorithms

Genetic Algorithms use techniques inspired based on the evolutionary ideas of natural selection and genetics such as,
Inheritance: The ability of modeled objects to mate, mutate and propagate their problem solving genes to the next generation, in order to produce an evolved solution to a particular problem. The selection of objects that will be inherited from in each successive generation is determined by a fitness function.
Mutation:Mutation alters one or more gene values in a chromosome from its initial state Used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next.
Selection: Selection The stage of a genetic algorithm in which individual genomes are chosen from a population for later breeding (using crossover operator).

Figure 1: The process of fuzzing

Crossover: Crossover A genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. Analogous to reproduction - taking more than one parent solutions and producing a child solution from them.

1.2 Fuzz testing

Fuzz testing or fuzzing is a software testing technique that involves providing invalid, unexpected, or random data to the inputs of a computer program, in order to check it against for coding error and security vulnerabilities.

Figure 1 is a high level fuzzing flow chart. The process consist of four steps. In step 1, we generate test data or make use of the test data available in hand. In step 2, we deliver this in to the application. Depending on the type of fuzzing and nature of the application, the way by which we deliver the test data varies a lot. To fuzz a web application, we generally use their API's, while a fuzzing a command-line application involves feeding the data as commands in expected format. In step 3, we observe the effect of a test case on thee application. If it fails or shows a problem, ee'll move to step 4 and save the details of this problem. Otherwise we go back to step 1 and repeat the process.

Given below is a python implementation of a basic fuzzer to fuzz FTP servers. The fuzzer generates random strings and use it with the standard FTP commands to fuzz a local or remote FTP server. This implementation is minimal and more features can be added.

```
import socket as skt
```

```
buffer = ["B"]
counter = 2

while len(buffer) <= 30:
        buffer.append("B"*counter)
        counter = counter+100
            # Some FTP commands
commands = ["MKD","GET","STOR","SYST", "XYZS"]

for command in commands:
        for string in buffer:
                # Sending commands
                s = skt.socket(skt.AF_INET, skt.SOCK_STREAM)
                                    #Target machine
                connect=s.connect(('172.16.6.69',21))
                s.recv(1024)
                s.send('USER ftp\r\n')
                s.recv(1024)
                s.send(command+' '+string+'\r\n')
                s.recv(1024)
                s.send('QUIT FTP \r\n')
                s.close()
```

2 Related Research

A lot of researchers working in the area software testing have moved their attention to evolutionary fuzzing in the recent years.

In 2007, J DeMott et.al proposed the idea of evolutionary fuzzing[2]. In 2008, S Bratus et.al made LZFuzz[1], a fast evolutionary fuzzing inspired fuzzer for poorly documented protocols. In 2010, S Rawat et.al showed[5] the effectiveness of evolutionary fuzzing approach in detecting buffer over-flow vulnerabilities. In 2012, Fabien Duchene et.al showed [3] how evolutionary fuzzing can be applied to extend the performance of traditional XSS fuzzing.In 2014, they extended their work and made a web fuzzing framework called KameleonFuzz [4].

3 Evolutionary fuzzing

In evolutionary fuzzing approach, we use test cases generated by genetic algorithms in step 1, shown in figure 1. Evolutionary algorithms are adaptable in ways that humans are not and may discover new or better test cases.

Figure 2 shows how this is being accomplished in step 1. Here essentially A population of individuals is evolved depending of their fitness score and fuzzing operators. Here what we meant by an individual is a single application input. From the initial population, we evolve the population in three steps. First we submit the individual (test case). Then we analyse the test verdict and the fitness. These two steps will be carried out for all individuals. Then we

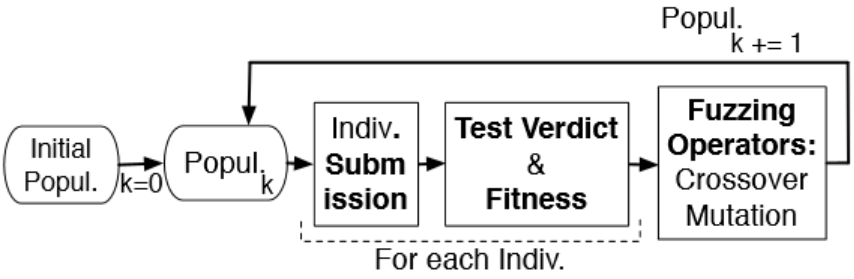

Figure 2: Evolutionary test case generation

apply crossover and mutation, which are explained earlier to generate better individuals.

4 XSS Vulnerability Detection

In this section I discuss a recent study about evolutionary fuzzing used to detect XSS vulnerabilities. Note that this section is highly referenced from the paper "XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing" [3] by Duchene et.al.

Cross site scripting(XSS) allows attackers to inject browser side script (usually JavaScript) into web pages accessed by other users. By running malicious JavaScript (or any other browser side script), attackers can retrieve cookies, session tokens and sensitive information of the victim.

An overview of the evolutionary XSS fuzzer under discussion is shown in Figure 3.

Web Application: We define the web application under test, usually referred to as SUT(System Under Test) in the following way.

A transition(u) is defined as a mapping of n user inputs from $i^u_l \in \Sigma^* : I_u = i^u_1, ..., i^u_n$ to output $q = q_1.q_2....q_k, q \in \Sigma^*$. Each q_j is either a web-server filtered input parameter i^u_l or a string q_h surrounding one or more q_j.

An individual is a sequence $I = (I_1, ..., I_m)$ where each I_u satisfies the above definition.

The value if I_3 (kalimu) is observed in q, the output of the transition S2 to S3. We assume a possibility of XSS there and start fuzzing on I_3 from S2.

XSS Fuzzing attack grammar: An an input grammar is needed to impose some restrictions on GA to generate inputs by constraining mutations and crossovers. This helps to be closer to the behaviour of an attacker who would mainly modify interesting input parameters.

$< a\, name = "[USER\_GIVE\_INPUT]" > Hello! < /a >$

Here q = q1 + q2 +q3 meaning that q2 is a result of a filtering function applied to an input parameter i^3_l

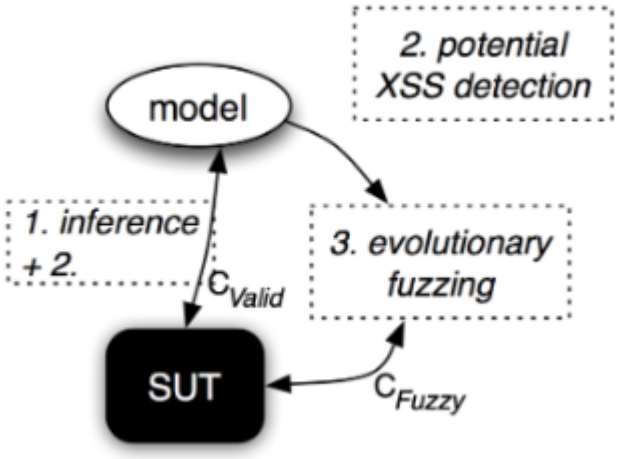

Figure 3: High level view of evolutionary XSS fuzzer

XSS Fuzzing attack grammar: Figure 5, taken from the original paper shows an extract of the written attack grammar for guiding input mutations.

Creating first generation: Individuals of the first generation are created from the attack grammar and known attack inputs.

Character classes: Exploiting an injection is about sending data and instructions to the SUT that does not use them in a safe way and assumes those inputs as only data. First submit only data for inferring a SUT formal model. Then during the fuzzing step, a combination of data and instructions is sent to the SUT.

Detecting XSS attacks: If the attacker succeeds in crafting an input i such that $q_2 = kalimu"onlclick = "alert(1)$, then q_2 is not syntactically confined w.r.t G0 (HTML)

Fitness function: The evolutionary fitess function to detect XSS is give in

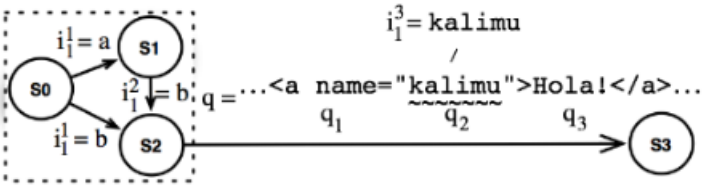

Figure 4: When the value of an input parameter I_l is observed in the output q, the fuzzing starts from that initiating state on that very same I_l

```
HTML_XSS_FIELD ::= HTML_TEXT_SIMPLE HTML_TAG_QUOTE
      HTML_TAG_SPACE HTML_TAG_EVENT HTML_TAG_EQUAL
      HTML_TAG_QUOTE JS_PAYLOAD
HTML_TAG_QUOTE ::= ' | "
HTML_TAG_SPACE ::= \n | \t | \r | ␣
HTML_TAG_EQUAL ::= =
HTML_TAG_EVENT ::= onabort | ... | onclick | ... | onwaiting
```

Grammar fragment 1. Injecting into an HTML attribute field value

shows an input parameter value i_l (thus a subset of an input sequence) generated using that grammar:

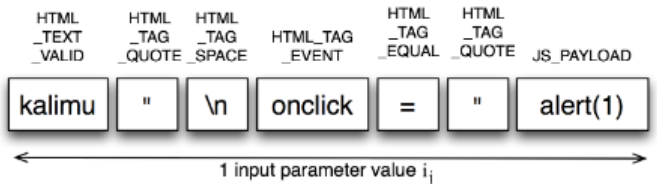

Figure 5: An extract of the attack grammar

figure 7. States reachable within few transitions from the initial state are more likely to be sanitized than deeper ones. Fitness is an increasing function of it. A well formed output will be more likely be executed by the client. Well(I) = 1 if q is well formed 0 otherwise.

Evolving the population: This is the final part of the model. Following is the definition of mutation and crossover operations that tend to respect the Attack Input Grammar G_{AI}
Let two individuals with potential reflection $I = (I_1, ..., I_m)$ and $J = (J_1, ..., J_m)$. Crossover performed and a child would be: $I = \left(I_1, ..., I_m, \left(i^{1^m}, ..., i^{u}_{x-1}, j^{w}_{y}\right)\right)$
Where i,j are input parameters.

5 Future of Evolutionary Fuzzing

Evolutionary fuzzing is still a relatively new area in software testing. Most of thee research came out so far have not added much novel ideas from thee works of Je Demott et.al[2] in 2007. One one challenge still in front of us is about properly dealing with obfuscated and encrypted files. An other problem with traditional evolutionary fuzzing is the complexity it adds up in-terms of implementation. For the same reason, even though this idea has been here for a decade, there are only a handful of tools which actually make use of evolutionary fuzzing. Also most of the tools now-a-days are written in scripting languages like python or pearl, to reduce the implementing complex, which results in considerable performance reduction. An open source project called afl-fuzz[6] released by Michał Zalewski (a.k.a lcamtuf) is written in C++ and so bar the

- C_0: HTML Spaces: ␣ || \r || \t || \n
- C_1: HTML Attribute delimiter: " || ` || '
- C_2: HTML Tag delimiter: < || > || />
- C_3: HTML Equal sign: =
- C_4: JavaScript code: (||); ||{||}
- C_5: URL related: /|| : ||?||&
- C_6: Escaping character: \
- C_7: HTML_TEXT_SIMPLE: [a-Z] ∪ [0-9]

Figure 6: Attack grammar class

$$Fit(I) = \frac{S(I)}{S_{total}} + \frac{C_{injected}(I)}{C_{sent}(I)} + W_{ell}(I) + N(I)$$

Figure 7: Fitness function

best evolutionary fuzzer implemented interms of speed and complexity. I believe that, in the coming years, more fuzzer which makes use of evolutionary fuzzing will come especially specific for areas of web and databases.

References

[1] Sergey Bratus, Axel Hansen, and Anna Shubina. Lzfuzz: a fast compression-based fuzzer for poorly documented protocols. 2008.

[2] Jared DeMott, Richard J Enbody, and William F Punch. Revolutionizing the field of grey-box attack surface testing with evolutionary fuzzing. *BlackHat and Defcon*, 2007.

[3] Fabien Duchene, Roland Groz, Sanjay Rawat, and Jean-Luc Richier. Xss vulnerability detection using model inference assisted evolutionary fuzzing. In *SECTEST 2012-3rd International Workshop on Security Testing (affiliated with ICST)*, pages 815–817. IEEE Computer Society, 2012.

[4] Fabien Duchene, Sanjay Rawat, Jean-Luc Richier, and Roland Groz. Kameleonfuzz: evolutionary fuzzing for black-box xss detection. In *Proceedings of the 4th ACM conference on Data and application security and privacy*, pages 37–48. ACM, 2014.

[5] Sanjay Rawat and Laurent Mounier. An evolutionary computing approach for hunting buffer overflow vulnerabilities: A case of aiming in dim light. In *Computer Network Defense (EC2ND), 2010 European Conference on*, pages 37–45. IEEE, 2010.

[6] Michał Zalewski. American fuzzy lop. 2012.