Fuzzing in the (Amazon) Cloud – A worthy alternative to classical fuzzing?

Wilfried Kirsch[1], Hartmut Pohl[2]

Abstract

Fuzzing is a method to identify software bugs and vulnerabilities in executables. The current development shows a trend to move fuzzing into the cloud (cloud fuzzing), that allows a dramatic fuzzing speed increase up to factor 100 compared to classic fuzzing running on a typical personal computer. This paper shows benefits and drawbacks of cloud fuzzing for companies. With the softScheck Cloud Fuzzing Framework, a feasible software solution for fuzzing in the Amazon cloud is presented and its architecture sketched.

1 The evolution of fuzzing

Fuzzing is a technique to test the robustness of software, which was developed in 1988 by Barton Miller (Miller B 1988). A fuzzer is a program which feeds the target program with generated data. It then monitors the target program to check if given input leads to a crash a hang or other unexpected behavior.

Lots of those anomalies are in fact vulnerabilities, which a sophisticated attacker could exploit. For security experts fuzzing is basic equipment, because it’s automatable and can be used on every software that in some way reads data, which is nearly every software that exists. The source code is not needed, although if available it can speed things up.

The first fuzzers worked by randomly mutating data and sending it to the target program. Those fuzzers are called dumb fuzzers. Current generation fuzzers sometimes are aware of the protocols the target program expects and are able to create data that is within the specifications of the protocol. Such fuzzers are known as smart fuzzers. Assuming a program reads a file which has a checksum attached to it, a dumb fuzzer will fail to calculate the correct checksum, while a smart fuzzer will automatically generate and append the correct checksum, and so the target program does not exit immediately but continues to run through more program code until it will eventually exit or crash. The more code gets executed (code coverage), the higher is the probability to hit an error. Using techniques like source code instrumentation (Bär R, Prof. Dr. Fischer D) and symbolic execution (Hicks, M 2013) the code coverage per input can be further increased.

It is expected that cloud fuzzing will be used more frequently in the future (Godefroid P, Molnar D 2010). In fact, some companies like Google or Microsoft already use cloud fuzzing to identify security vulnerabilities (Evans C, Moore M, Ormandy T 2001).

[1] Wilfried Kirsch, IT-Security-Consultant softScheck GmbH, Sankt Augustin - Germany

[2] Prof. Dr. Hartmut Pohl, geschäftsführender Gesellschafter softScheck GmbH, Sankt Augustin - Germany

2 Cloud fuzzing

The difference between cloud fuzzing and local fuzzing is, that in cloud fuzzing the fuzzer as well as the target program do not run on a local machine, but on a remote machine. The remote machine can be in the same office building (private cloud) or in a data center of a cloud service provider. Because the user of the cloud usually does not have physical access to the machines used there, protocols such as SSH, RPC or VNC are used to control the cloud machines remotely. Well-known cloud service providers, whose machines can be used for cloud fuzzing, are Amazon (Amazon Web Services), Microsoft (Azure) and Google (Google Cloud Platform). All these service providers have a pay for use payment model, which means you pay for the time you used a certain machine. Slow machines are only a few Cents per hour while the fast ones are over a Dollar.

3 Comparison of cloud fuzzing and local fuzzing

This Chapter compares cloud fuzzing with local fuzzing. At the end of the chapter, the benefits and drawbacks of cloud fuzzing will be presented in a table. A rating is not given as this very much dependents on the actual field of application, the company as well as personal convictions. The reader must, therefore, decide for himself whether the advantages or disadvantages prevail.

3.1 Disadvantages of cloud fuzzing

With cloud fuzzing, it is impossible to fuzz real hardware because that hardware simply cannot connect to the instance running on a server far away. Also, while possible, infrastructures are hard to fuzz with cloud fuzzing, due to the high amount of work it takes to reproduce the infrastructure in the cloud.

Another drawback, at least for some companies, is the fact that the data lies unencrypted on the instances and therefore on Amazon’s data centers. This means, it is possible for everyone who has access to the data servers (amazon staff, governments, a skilled hacker) to retrieve the companies program, which might be not even released at the time.

At last, cloud fuzzing can be far more expensive than local fuzzing. More about the cost of cloud fuzzing can be found in chapter 4.

3.2 Advantages of cloud fuzzing

A computer that currently fuzzes has a very high CPU utilization so it cannot be used productively for other tasks. Therefore, a new Computer has to be bought, with the exception that the company only fuzzes at a time when no one works on that computer. A new computer has some drawbacks in comparison to a cloud machine:

It costs money, whereas cloud instances only cost when they are actually used.

A computer needs maintenance, can break, must be repaired or even replaced. If a cloud machine has a hardware issue, it’s the problem of the cloud service provider. They have to fix the machine. A possible failure can, of course, affect the cloud fuzzer. However, big cloud service providers have SLAs with high availability guarantees. Amazon states that their instances have 99.95% uptime per month, otherwise the user can get up to 30% of their expenses back to compensate the downtime.[3]

[3] https://aws.amazon.com/de/ec2/sla/

Also, computers need space. If the company buys only one PC for fuzzing that might be not important, but the more PCs there are the more space is required. The company might need an additional room, just for the fuzzing computers or even an extra building. The cloud computer on the other side is located many miles away from the business.

Next computers have to be set up for cloud computing. This costs time, which employees could spend more meaningful. In the cloud, a user can choose out of many images with the most diverse combination of operating systems and pre-installed software. On Amazon there a more than 25.000 images available.[4] It is also possible to create an own image, which comes with pre-installed fuzzers. So fuzzing can begin within seconds after an image has been chosen.

Most machines that can be bought are limited in their performance.

From these benefits, a last and perhaps most important advantage of cloud fuzzing can be defined: Flexibility. Almost everything can be exchanged within a very short time. A program can be fuzzed on different operating systems at the same time, to check if it might behave differently on them. If one project requires a high data throughput, an instance that utilizes many SSDs in a RAID0 configuration can be used. When an application requires lots of RAM, a corresponding instance can be selected. If a web application or a network protocol should be tested, hundreds of low end, and therefore cheap, machines can be created to get the job done.

3.3 Summary

Finally, all arguments for and against cloud fuzzing, are presented in a table. The arguments should be weighted accordingly to use case, company and available budget. Thus, it is impossible to say that cloud fuzzing is better or worse than classical fuzzing without knowing the surrounding conditions.

Pros	**Cons**
No initial costs	Not applicable for all areas
Does not need physical space	Privacy
Maintenance free	More expensive, in the long run.
Quick setup	
Fast machines	
Flexibility	

Table 1: Advantages and disadvantages of cloud fuzzing

[4] aws ec2 describe-images | grep ami | wc -l

4 Cost

One of the primary factor of making a decision for or against cloud fuzzing is the price. Hence it gets its own chapter. The following figure compares a desktop PC (Intel Core i7 4790K CPU and 32GB RAM) to a m4.2xlarge Amazon instance. Both are equally powerful. The desktop PC costs about 1100 Dollar. The electricity costs for continuous utilization are about 30 Euro per month.

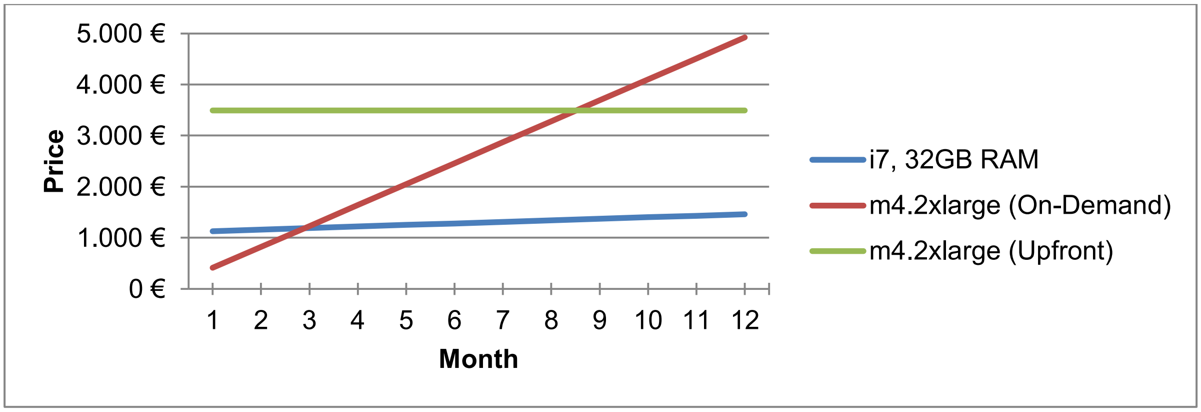


Diagram 1: Cost comparison of cloud and local machines over a period of one year.

After three months the costs of the desktop PC have been amortized. In this time it would have run the softScheck test program about 350 billion times[5] – a, for the most software projects, unnecessary high amount! This shows that in most cases it is more cost-effective to use cloud fuzzing. For companies that want to fuzz continuously for more than three months, e.g. big IT-Security companies, the acquisition of local computers might be worthwhile. It should be noted, however, that Amazon also offers "Reserved" instances these costs, as opposed to the On-Demand Instances also money when they do not run. Reserved Instances have a runtime between one and 36 months, while instances with a runtime longer than 12 months are cheaper if broken down to costs per hour. Overall, Reserved Instances can be up to 75% cheaper compared to on-demand instances, provided they are paid up front. Still, after not even a year the PC is cheaper.

There is a third type of instances, the Spot-Instances.[6] For Spot-Instances the user bids on unused EC2 instances. The highest N bids get a Spot-Instance. N is a number of available Spot-Instances of a certain machine type. N is dependent on Amazons free computing capacity. If Amazon's computing capacity runs out or there is a higher bid, it can happen that your own bid is no longer within the N-highest bids. In this case, the acquired Spot-Instance gets terminated. If enough other users terminate their own Spot-Instance or Amazon again has enough computing power available, it might happen that the user's bid is in N again, which means the user is able to start a Spot-Instance again. Another disadvantage is that the very fast instance types are not available in this case. A shutdown of the instance is also not possible (it is equal to terminating the machine), but a reboot is.[7] If the user is flexible and needs the machine just for a short period, he can save a lot of money using Spot-Instances (up to 90% compared to On-Demand Instances).

[5] 5500 executions per second * 8 CPU cores * 60 seconds * 60 minutes * 24 hours * 30 days * 3 months

[6] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html

[7] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/how-spot-instances-work.html

5 Amazon Web Services – Elastic Compute Cloud

The Amazon Web Services (AWS) is a collection of various online services of the company Amazon.com. AWS is currently the biggest player in cloud computing (Meyer A). One component of AWS is Elastic Compute Cloud (EC2). EC2 allows the user to set up virtual machines, which are fully configurable and act as servers. EC2 provides the foundation for the cloud fuzzing framework presented in the next chapter, as the fuzzers run inside an EC2 virtual machine. In order to understand the technical side of sCFF, this chapter will briefly present the functioning and the nomenclature of EC2. EC2 allows the creation and execution of so-called instances. An instance is a virtual server in the cloud, consisting of an operating system, including the software installed on it, assigned resources, as well as various settings, which are selected during the creation. The operating system can be chosen from a huge pool of Amazon Machine Images (AMI).[8] An AMI consists of the following three things:

1. A Template for the root volume (usually the operating system)
2. A set of Permissions
3. Block Device Mapping (Determines the mount points)

It is possible to create an AMI from any (running) instance, making it possible to clone an instance. Self-created AMIs are private by default and thereby can't be found by other AWS users.

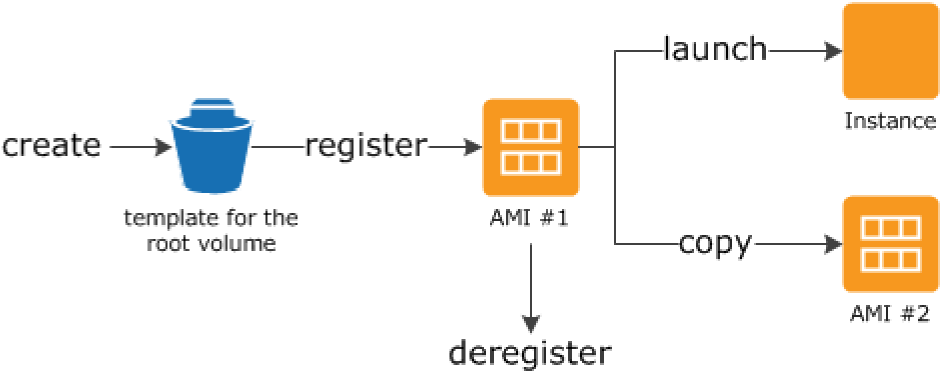


Figure 1: AMI Lifecycle

If an AMI is no longer needed, it can be deleted (deregister).

After choosing an AMI, the machine type on which the image will be executed must be selected. The machine types are divided into categories: General purpose, compute optimized, memory optimized, storage optimized, etc... Every category consists of multiple virtual machines, with different performance. The slowest instances are cheaper than 1 Cent per hour while the fastest cost more than 18 Dollar per hour, but offer 2TB RAM and 256 CPU cores. If you choose a proprietary operating system, the price per hour increases. To make it easy to compare the performances between different instances, Amazon introduced a unit called Elastic Compute Unit (ECU)[9]. If an instance has twice as much ECUs than another instance, is also about twice as fast. Within the same category, price and ECUs correlate. Once AMI and a machine are selected, the instance can be started. It is advisable to configure them before doing so through. It is possible to configure the network, add volumes, activate monitoring and tinker with a lot of other things. Some settings cost extra. Important, especially in connection with the fuzzing framework presented in the next chapter, are tags as well as the security group. Tags are key-value pairs that can be read over an API, even when the instance is stopped.

[8] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html
[9] https://aws.amazon.com/de/ec2/faqs/

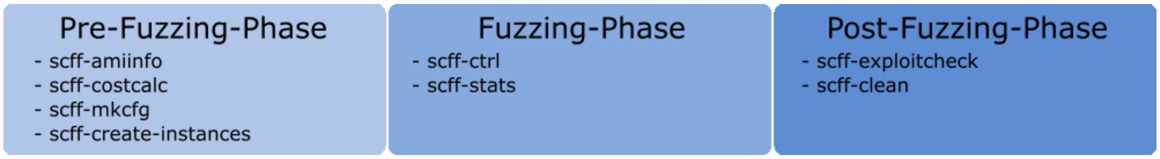
Security groups in the context of an instance can be compared with a simple firewall, they allow the restriction of network flow (blocking certain ports, IP addresses, protocols etc...)

There are various ways to interact with AWS. A user can login with a username and password at aws.amazon.com or use the aws-cli tool (authentication via API key). If AWS should be used within a program, the developer must use one of the AWS APIs. The Python AWS API is called Boto, currently available in version 3. Boto authenticates itself to the Amazon Web Services with the Amazon User Key and the Amazon Secret Key, which the user must retrieve from the Amazon Management Console. Instead of those two keys, it is advisable to use Amazons Identity and Access Management (IAM)[10] for greater security. IAM allows the creation of user and groups, which can be linked to policies. With those policies, it's possible to granularly restrict what a certain user or groups rights. If an attacker steals an IAM key pair, he can only do what this key pair permits, compared to the Root key pair, where he has full permissions. IAMs can also prevent accidents that can happen from wrongly used commands which can be destructive or expensive, simply because the user linked to the IAM key pair does not have the required rights to execute that command.

6 softScheck Cloud Fuzzing Framework (sCFF)

While companies like Google and Microsoft have their own fuzzing facility there is no product available for the masses, even though Project Springfield is expected to be publicly available soon.[11] Therefore, softScheck has created their own tool for cloud fuzzing. It is called softScheck Cloud Fuzzing Framework – short sCFF. In the future it should be capable of fuzzing on all major cloud infrastructures; in the current state it only supports the Amazon cloud.[12] There were three main reasons for using the Amazon cloud in this context. First: User can choose to save their data in Amazon computing center in Germany (Frankfurt), were German laws are applied. Another reason is the well documented Python API for AWS called Boto 3.[13] And the third reason is the fact that new AWS user have 750 free compute hours per month for the first year, as long as they use a small t2 machine. Those are ideal conditions to develop and test a lot without spending much money.

sCFF is written in Python3 and uses the Boto 3 API to communicate with the Amazon cloud and SSH for the communication with single AWS instances. sCFF is split up in multiple subprograms following the Unix paradigm one tool for one job. Every subprogram is used in a different fuzzing phase.



Pre-Fuzzing-Phase
- scff-amiinfo
- scff-costcalc
- scff-mkcfg
- scff-create-instances

Fuzzing-Phase
- scff-ctrl
- scff-stats

Post-Fuzzing-Phase
- scff-exploitcheck
- scff-clean

Figure 2: The sCFF subprograms

`scff-mkcfg` asks the user how many machines he want, how fast they should be, what AMI should be used and most important, where the program that should be fuzzed is stored. It then writes a project file, which `scff-create-instances` uses to tell Amazon to create the correct EC2 instances. It also writes the following tags: Name, GID and Role. Name should be self-explanatory, GID stands for Group ID. All instances with the same GID can be controlled

[10] https://aws.amazon.com/de/iam/
[11] https://www.microsoft.com/en-us/springfield/
[12] https://aws.amazon.com
[13] https://boto3.readthedocs.io/en/latest/

at the same time. The first instance of a group gets the Role tag with the value „Master“. This will later be used to determine and identify the distributed fuzzing server.

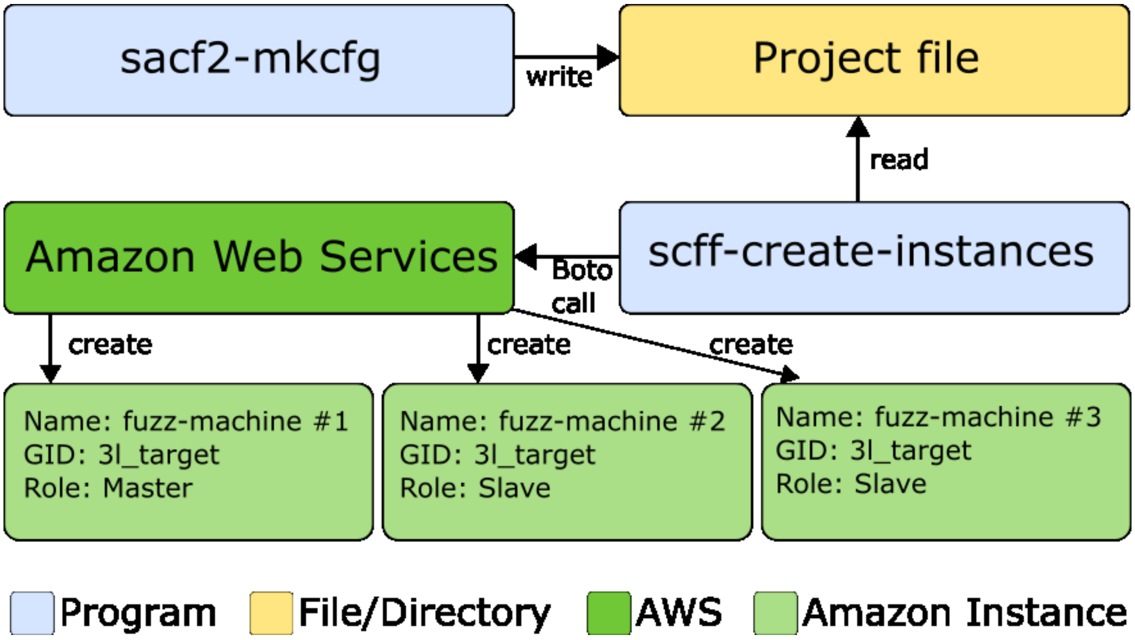
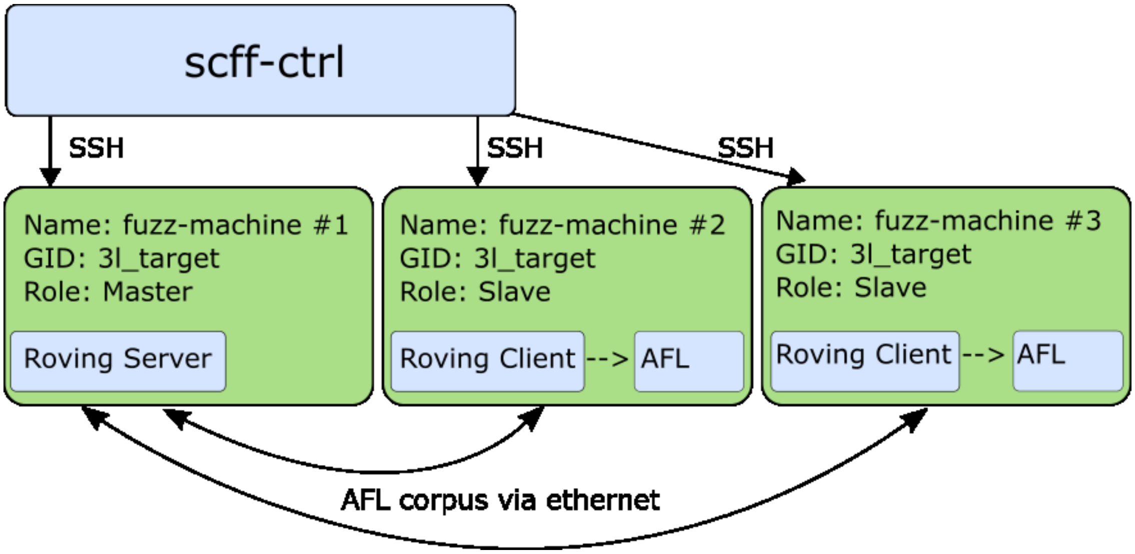


Figure 3: Pre-Fuzzing Phase

Depending on whether a sCFF AMI was used during the instance creation, the instance must still be prepared for fuzzing. Otherwise only the project definition, as well as the test program must be copied to the instance and the fuzzing can begin. By default, sCFF uses American Fuzzy Lop (AFL) [14] as fuzzing program. AFL is known for its high performance as well as the ability to instrument programs, which leads to the generation of better test data and thus increases the speed even further.



[14] http://lcamtuf.coredump.cx/afl/

Figure 4: Fuzzing Phase in distributed mode

If there are multiple instances of the same group running, fuzzing is also possible in distributed mode. In distributed mode, the fuzzers send their AFL corpus to a server (master role) which then sends the corpus to all other fuzzers registered to the server. Sharing the corpus shares the found program branches of one fuzzer to all the others, which ultimately leads to finding vulnerabilities faster. The table below shows the speed gain for five machines, fuzzing the softScheck test software using distributed mode.

Input length	**Single mode [h]**	**Distributed mode [h]**	**Factor**
2	< 0.01	~ 0.01	-
3	< 0.01	~ 0.01	-
4	~ 0.15	~ 0.06	2,50
5	9	4	2,25
6	44	16	2,75
7	> 50	> 50	-

Table 2: Time 5 AFL instances needed to find passwords with a given length running in single mode and in distributed mode.

The test program crashes when certain inputs from a certain input length are made. Hence the fuzzer must identify these strings. After the first half of the required string has been read by the target program, it enters a new branch. AFL detects this branch and if running in distributed mode, shares it with the other instances. Inputs shorter than three characters are found nearly in an instant, so both modes are on par. The longer the required input, the faster is distributed fuzzing compared to the normal fuzzing process. Distributed mode is implemented with a modified Roving[15] version.

The fuzzers on the machines are monitored. On events such as a new finding, crash of a sCFF component. If the user finds that the program was fuzzed long enough, sCFF can send the findings to the local computer and halts the instances. `scff-exploitcheck` then filters duplicates and false-positives. The remaining finds are automatically checked for exploitability by the debugger GDB. If the program contains debugging symbols or if the source code is available, the code line in which the program crashed can be displayed, which makes it easier to derive the error as well as the patching.

[15] https://github.com/richo/roving

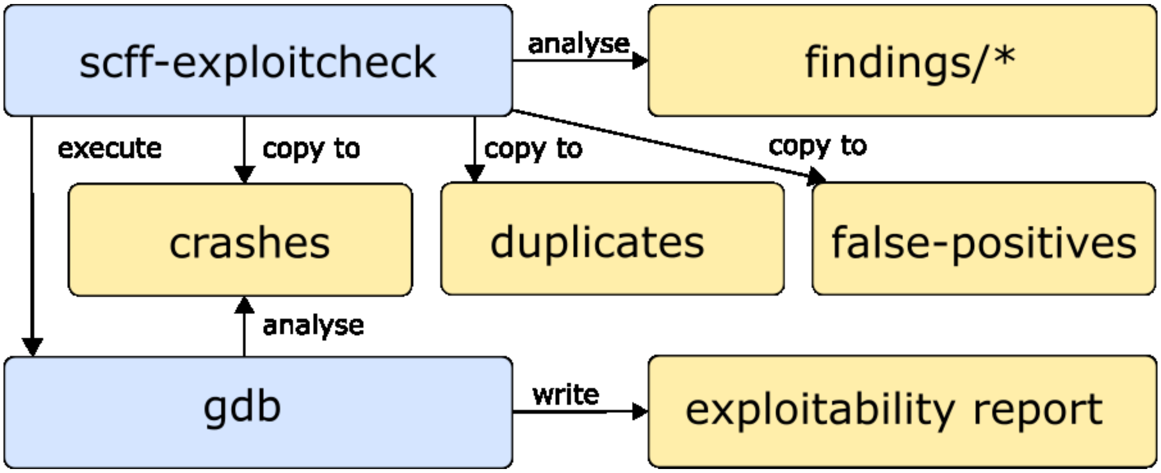


Figure 5: Check for false positives, duplicates and exploitability

7 Conclusion

Cloud fuzzing has many advantages over classical fuzzing. Developers can cost-effectively test their product for errors and vulnerabilities. IT-Security companies, which fuzz a lot, probably have to pay more in comparisons to classic fuzzing, but can save a lot of time using a cloud fuzzing framework like sCFF, as well as gaining flexibility and finish fuzzing assignments faster than before. Whether the advantages outweigh the disadvantages, is up for the company to decide for itself. However, the fact that cloud fuzzing is the future, is undeniable.

8 References

Bär R, Fischer D (2012) Code-Coverage auf kleinen Targets http://www.elektroniknet.de/embedded/entwicklungstools/artikel/92028/2/

Blodget H (2011) Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data http://www.businessinsider.com/amazon-lost-data-2011-4?IR=T

Evans C, Moore M, Ormandy T (2001) Fuzzing at scale https://security.googleblog.com/2011/08/fuzzing-at-scale.html

Godefroid P, Molnar D (2010) Fuzzing in The Cloud (Position Statement) https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/paper-80.pdf

Hicks, M (2013) Symbolic Execution for finding bugs https://www.cs.umd.edu/~mwh/se-tutorial/symbolic-exec.pdf

Meyer A (2015) Vier Konzerne dominieren den Cloud-Markt https://www.heise.de/ix/meldung/Vier-Konzerne-dominieren-den-Cloud-Markt-2763042.html

Miller B (1988) Original fuzz project assignment http://pages.cs.wisc.edu/~bart/fuzz/CS736-Projects-f1988.pdf

Author’s bio

Wilfried Kirsch has been working as a consultant for IT-Security at softScheck GmbH since 2016. His professional focus is on IT security audits of networks, soft and hardware products as well as the development, testing and evaluation of security requirements, security concepts and security architectures.
Contact: wilfried.kirsch@softscheck.com | Tel: +49 (2241) 25 43 -21.

Prof. Dr. Hartmut Pohl is the CEO of softScheck with the focus on:

Tactical and strategic security consulting, among others. Based on BSI basic protection, ISO 27000 family, COBIT, NIST SP 800, ITIL etc. And second: Secure software - Secure Software: Security Consulting with the ISO 27034 based development process of secure software and the security testing process for the identification of unidentified security gaps (zero-day vulnerabilities): Security Requirements Analysis, Threat Modeling, Static Source Code Analysis, Dynamic Analysis (Fuzzing), Penetration Testing

As a test partner of TÜV Saarland, softScheck guarantees all test results with a proprietary certificate or a TÜV certificate for software, firmware, apps and systems, control systems, websites, networks etc.