The Infinite Game of Vulnerability Research

Introduction

$ whoami

- David Calligaris @daviddiaul
- 2002 - 2016 Emaze Networks
  - IpLegion Security Scanner
  - CTO
- 2017 Huawei German Research Center
  - Fuzzing Kernel Space / User Space
  - Huawei Mobile Phones Bug Bounty Program
- Talk with me about
  - #security #fuzzing #memorycorruptions #martiallarts #calistenics #aj1 #memetics #gametheory

Huawei calls hackers to Munich for secret bug bounty meeting

Zack Whittaker @zackwhittaker / 5:25 PM GMT+1 • November 5, 2019

Disclaimer

All the content of these slides represents

<u>my personal view</u> not that of my employer

I will talk about

The Infinite Game of
Vulnerability Research

Game Theory

Variant Analysis
Fuzzing
Manual Code
Analysis

Game Theory

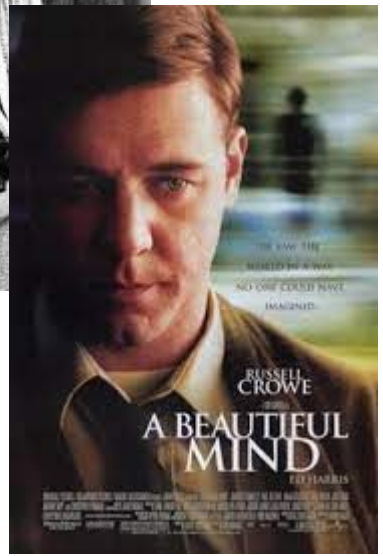
Game Theory

John Von Neumann

Game Theory - Science of decision making

A game is any interaction between multiple people in which each person's payoff is affected by the decisions made by others

John Nash



Game Theory - Finite / Infinite Games

Influential thinkers in Game Theory

- James Carse
  - *Finite and Infinite Games (1986)*
- Simon Sinek
  - *The infinite game (2019)*

James Carse

Simon Sinek

Finite Game VS Infinite Game

Finite Game	**Infinite Game**
Played by known players	Played by known and unknown players
Fixed rules	Not exact agreement on rules
There are objectives, when reached ends the game	Players can change how they play the game
There is a beginning, a middle and an end	Infinite time horizon
There is a clear winner and a loser	There is no such thing as winning or losing an infinite game, the goal is to stay in the game as longer as possible, outlasting the other participants

Vulnerability Research - What?

Vulnerability research is a process used to find flaws in software (hardware) that could lead to security issues. This process could include static code analysis, dynamic testing etc

Vulnerability Research - Who?

Internal to the organization

- DAST / SAST Teams
- Internal Penetration Testing Team
- ...

External to the organization

- Bug Bounty hunters
- Bad Actors
- Academia
- ...

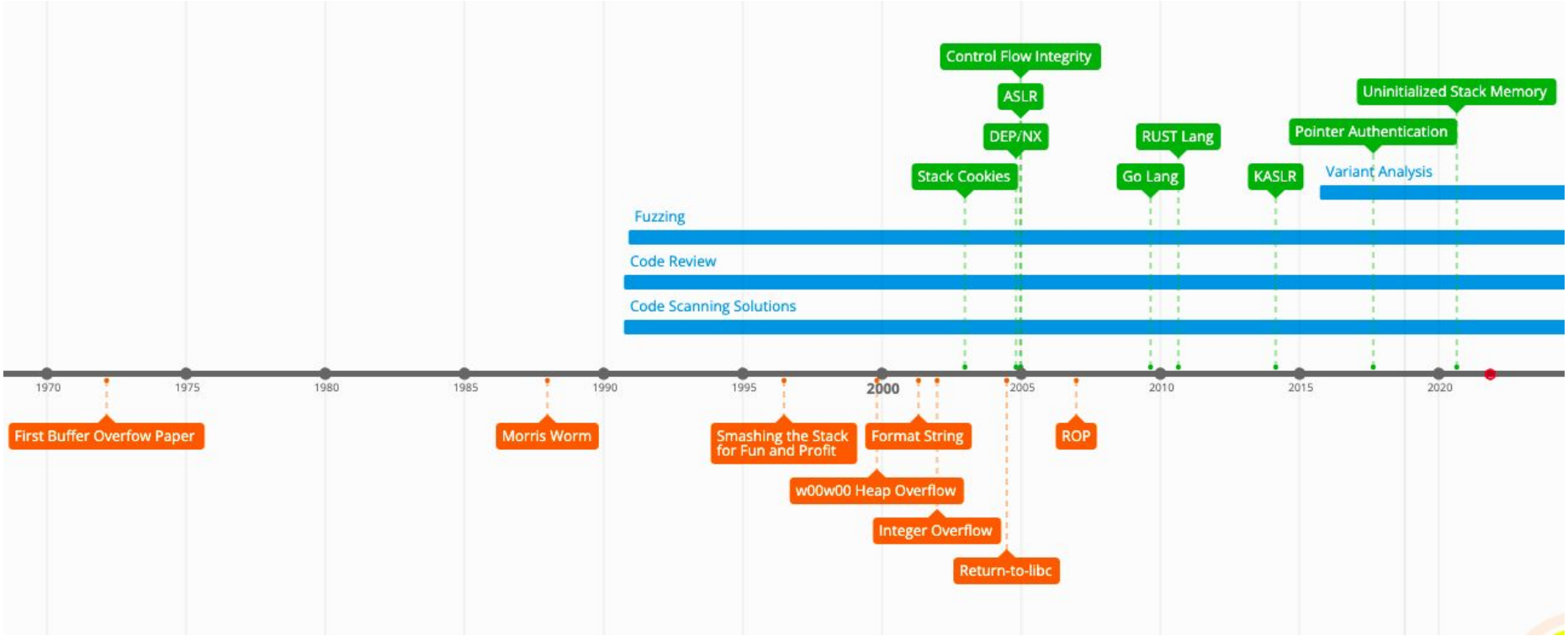
Vulnerability Research - Why?

- Finding (and patching) vulnerabilities is still an “effective” way to make software more secure, making Internet a safer place for all of us
- Can be quite remunerative (e.g. Bug Bounties, others)
- Fame & Glory

The Infinite Game of Vulnerability Research

Infinite Game	**Vulnerability Research Game**
Played by known and unknown players	New vulnerability researchers New players on the defensive side
Not exact agreement on rules	Only software? What about hardwear?
Players can change how they play the game	Introduction of grey box fuzzing Introduction of memory safe languages
Infinite time horizon	Do we know how/when/if the game will end?
There is no such thing as winning or losing an infinite game, the goal is to stay in the game as longer as possible, outlasting the other participants	Who is the absolute winner in cybersecurity of last year?

How was the game played?



Vulnerability Research - How?

Several methods:

- Variant Analysis
- Fuzzing
- Manual Code Review

American Fuzzy Lop plus plus (AFL++)

Release Version: 3.14c

Github Version: 3.15a

Repository: https://github.com/AFLplusplus/AFLplusplus

AFL++ is maintained by:

- Marc "van Hauser" Heuse mh@mh-sec.de,
- Heiko "hexcoder-" Eißfeldt heiko.eissfeldt@hexco.de,
- Andrea Fioraldi andreafioraldi@gmail.com and
- Dominik Maier mail@dmnk.co.

Originally developed by Michał "lcamtuf" Zalewski.

FRIDA

Variant Analysis

Variant Analysis - Sources

How do you become aware of a vulnerability in your product?

Internal to the organization

- Code scanning solution
- Dynamic testing
- Internal security team
- Honeypots
- Threat Intelligence
- ...

External to the organization

- External security team (e.g pentesting with a company)
- Responsible - Coordinated Disclosure / Bug Bounty Program
- Full Disclosure
- Exploited in the wild
- ...

Variant Analysis

- Patch the vulnerability
- How can we prevent similar bugs from happening again?
- Can we kill the vulnerability class?
- Can we introduce exploit mitigations?
- Do we have “variation” of the same vulnerability in other parts of the project or in other codebases?

Variant Analysis

Patch the vulnerability

How can we prevent similar bugs from happening again?

Can we kill the vulnerability class?

Can we introduce exploit mitigations?

Do we have “variation” of the same vulnerability in other parts of the project or in other codebases?

Variant Analysis

Variant analysis is the process of using known vulnerabilities as a starting point to find similar problems in the code

Introduction

After BleedingTooth, which was the first time I looked into Linux, I wanted to find a privilege escalation vulnerability as well. I started by looking at old vulnerabilities like CVE-2016-3134 and CVE-2016-4997 which inspired me to grep for `memcpy()` and `memset()` in the Netfilter code. This led me to some buggy code.

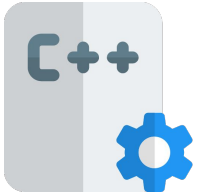
https://github.com/google/security-research/blob/master/pocs/linux/cve-2021-22555/writeup.md

Introduction and Methodology

As a researcher, it's important to add new techniques and software to your bug hunting methodology. A year ago, I started using CodeQL for my own research on open source projects and decided to compile the Linux kernel with it and try my luck.

https://www.sentinelone.com/labs/tipc-remote-linux-kernel-heap-overflow-allows-arbitrary-code-execution/

Variant Analysis

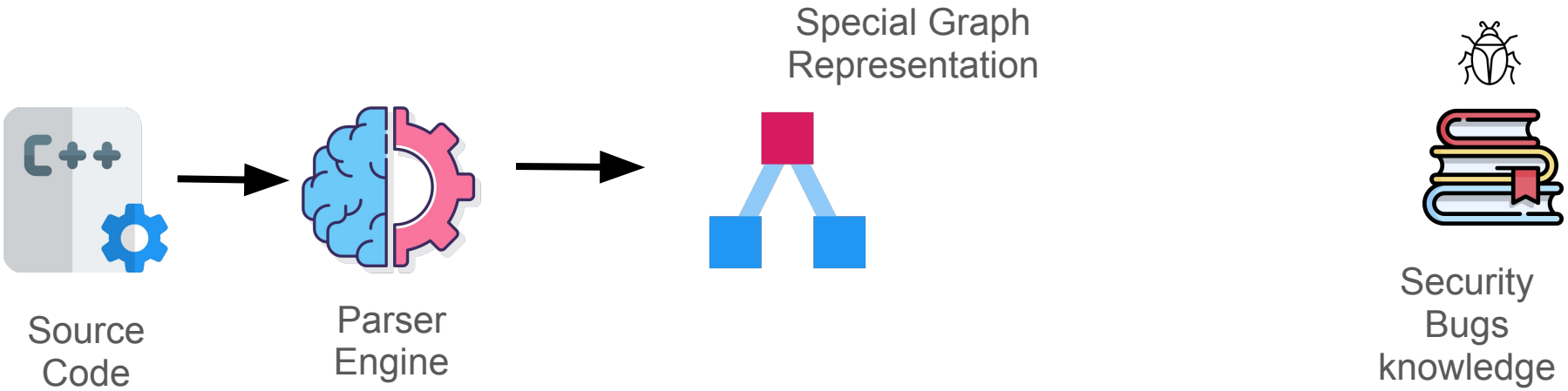


Source
Code

Security
Bugs
Knowledge

We start with our Source Code and with our Security Bugs Knowledge

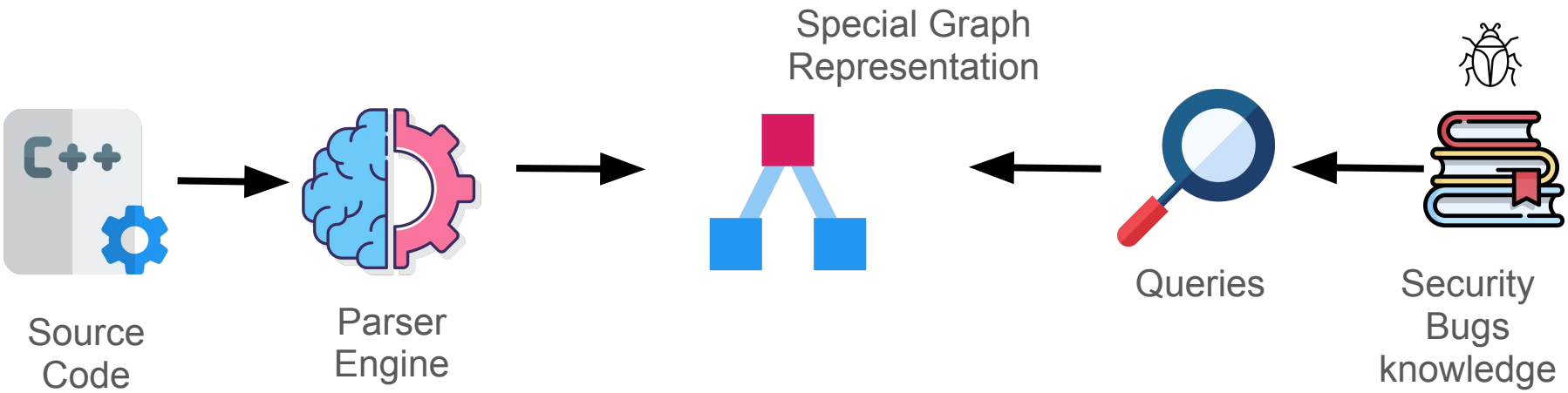
Variant Analysis



We analyze our Source Code with a Parsing Engine

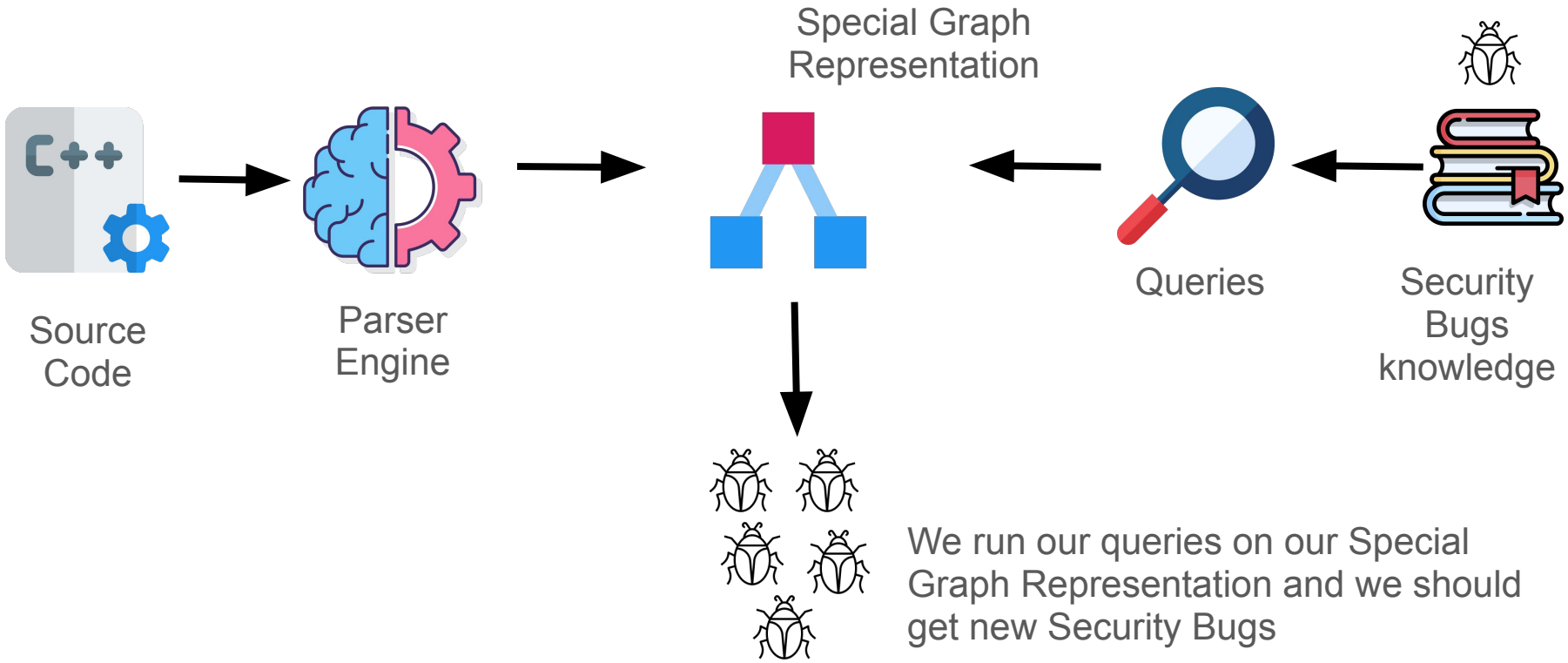
and we store it in a Special Graph Representation structure

Variant Analysis



We model our Security Bugs Knowledge in Queries / Templates

Variant Analysis



Variant Analysis

There are several projects aiming for this type of analysis:

- GitHub Semmle CodeQL
- SemGrep
- Coccinelle
- Google P0 Weggli
- ShiftLeft Joern

CodeQL

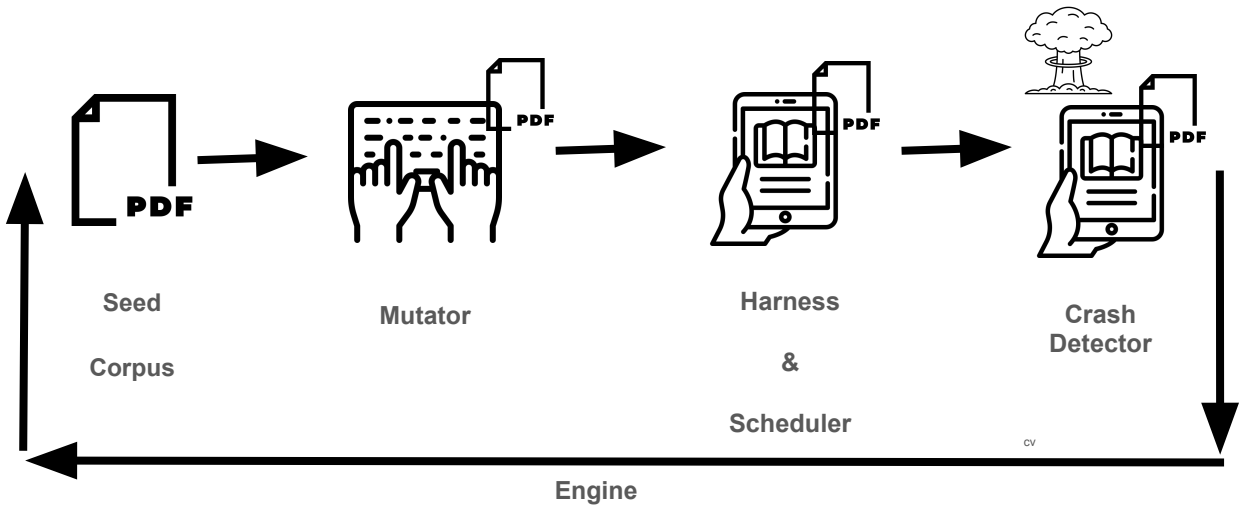
Semgrep

Fuzzing

Fuzzing - Explained for your grandparents

Dynamic software testing methodology that aims to find bugs by feeding with (random) data the SUT (Software Under Test) and observes its behaviour

cv



cv

Fuzzing - Pseudo Code

```
while(0) {
        fp = mutate_corpus(FILE);
        r = SUT.open_file(fp);
        If (r == SUT.crash):
                print("Crash Identified\n");
}
```

Engine

Mutator

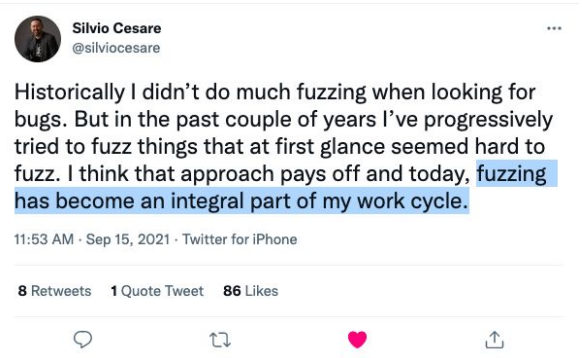
Harness - Scheduler

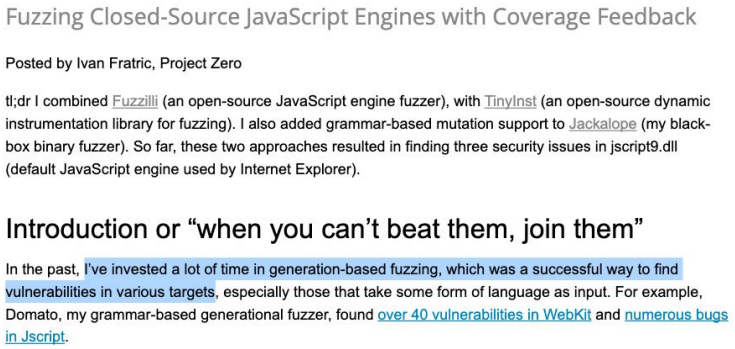
Crash Detector

Fuzzing

Fuzzing is an extremely effective way to identify bugs in software

Google OSS Fuzz / Cluster Fuzz project identified more than 30.000 bugs in over 500 Open Source Software projects (June 2021)





Fuzzing Closed-Source JavaScript Engines with Coverage Feedback

Posted by Ivan Fratric, Project Zero

tl;dr I combined Fuzzilli (an open-source JavaScript engine fuzzer), with TinyInst (an open-source dynamic instrumentation library for fuzzing). I also added grammar-based mutation support to Jackalope (my black-box binary fuzzer). So far, these two approaches resulted in finding three security issues in jscript9.dll (default JavaScript engine used by Internet Explorer).

Introduction or "when you can't beat them, join them"

In the past, I've invested a lot of time in generation-based fuzzing, which was a successful way to find vulnerabilities in various targets, especially those that take some form of language as input. For example, Domato, my grammar-based generational fuzzer, found over 40 vulnerabilities in WebKit and numerous bugs in Jscript.

https://googleprojectzero.blogspot.com/2021/09/fuzzing-closed-source-javascript.html

Fuzzing VS Something Else

Sometimes the term “fuzzing” is used to discover hidden resources on web servers. *Personally* I call it:

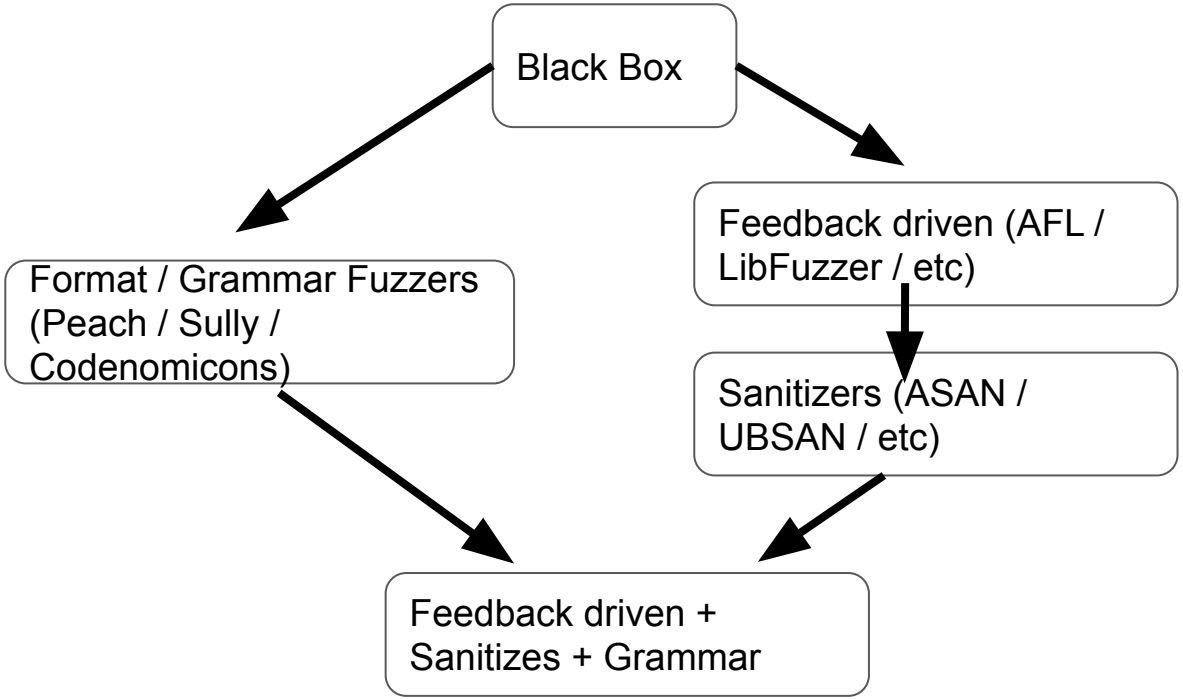
- Web server Resource Enumeration
- Directory Bruteforcing

Today we are not talking about this.

Fuzzing Concepts

A **Harness** is a piece of software that allows your fuzzing engine to interact with your SUT (Software Under Test)

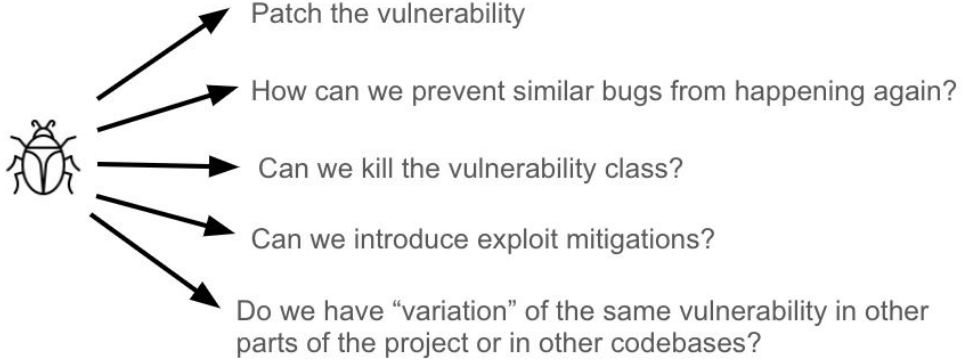
SUT Harness Engine



Vulnerability Rediscovery

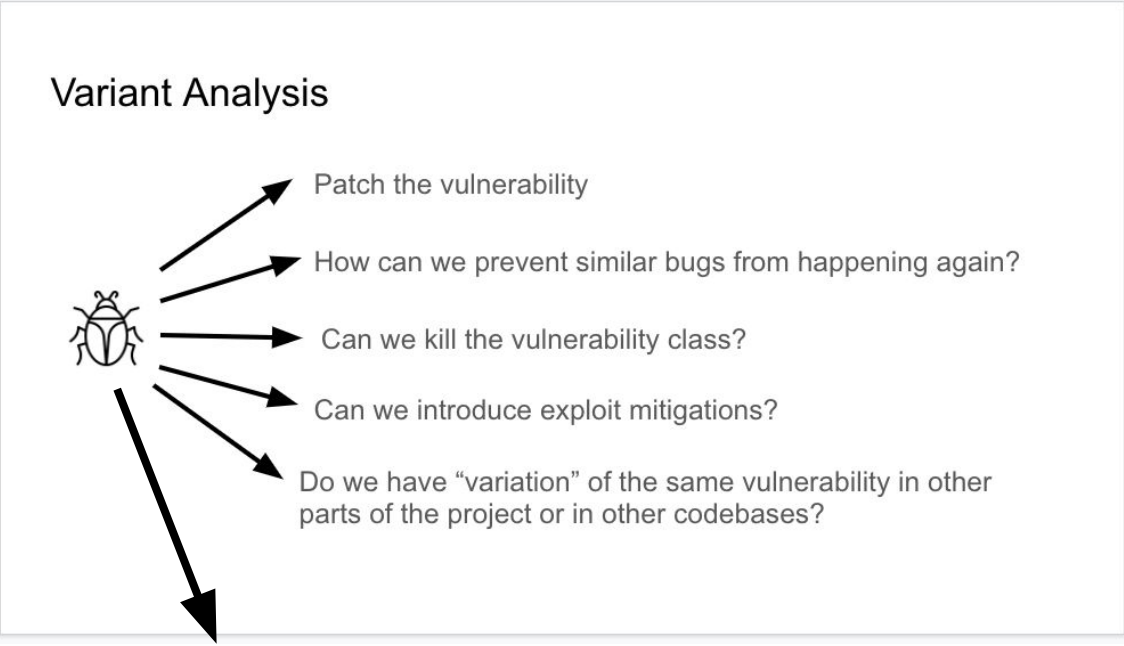
Do you remember this slide?

Variant Analysis



Vulnerability Rediscovery

Do you remember this slide?

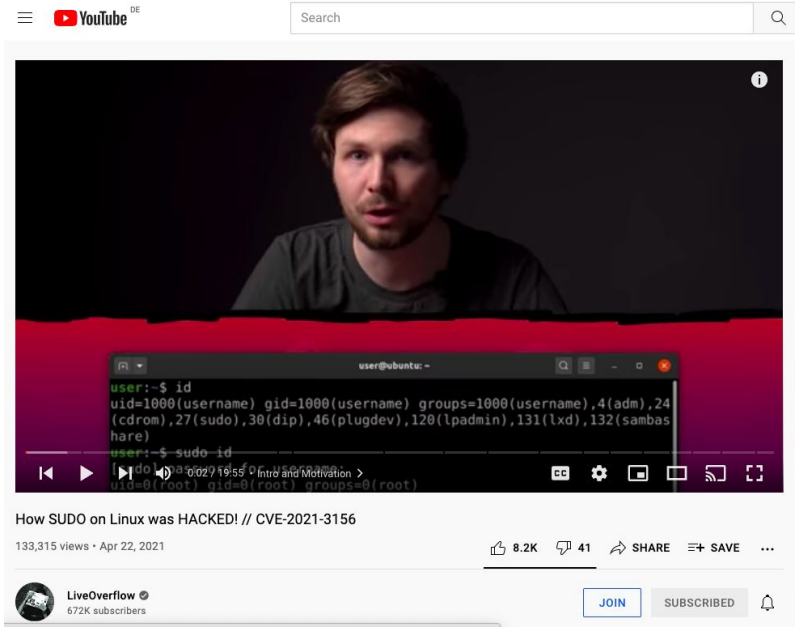


Why didn’t our fuzzers catch this vulnerability before?

Vulnerability Rediscovery

Taming your fuzzers:

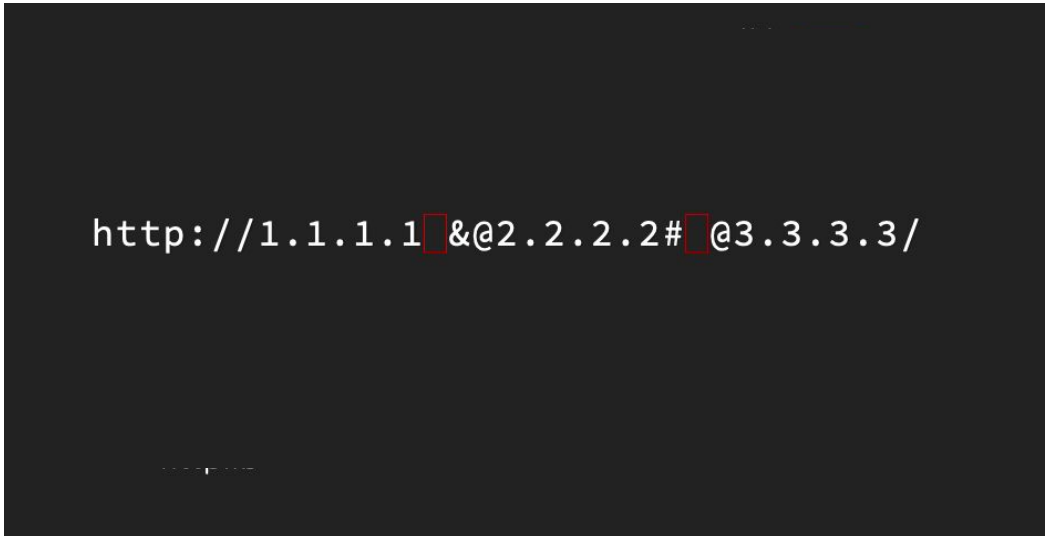
- Why didn’t my fuzzer spot the vulnerability reported?
  - Missing harness
  - Not exhaustive initial corpus seed
  - Not appropriate mutator
  - Not appropriate sanitizer
  - ...
- Are there other parts in the code where I did not “tame” the fuzzer correctly?



Differential Analysis

Differential analysis is the evaluation of the different outcomes that would arise from alternative solutions to a particular problem e.g.

- Parsing (URL, email, etc)
- Processing (archive extraction)
- ...

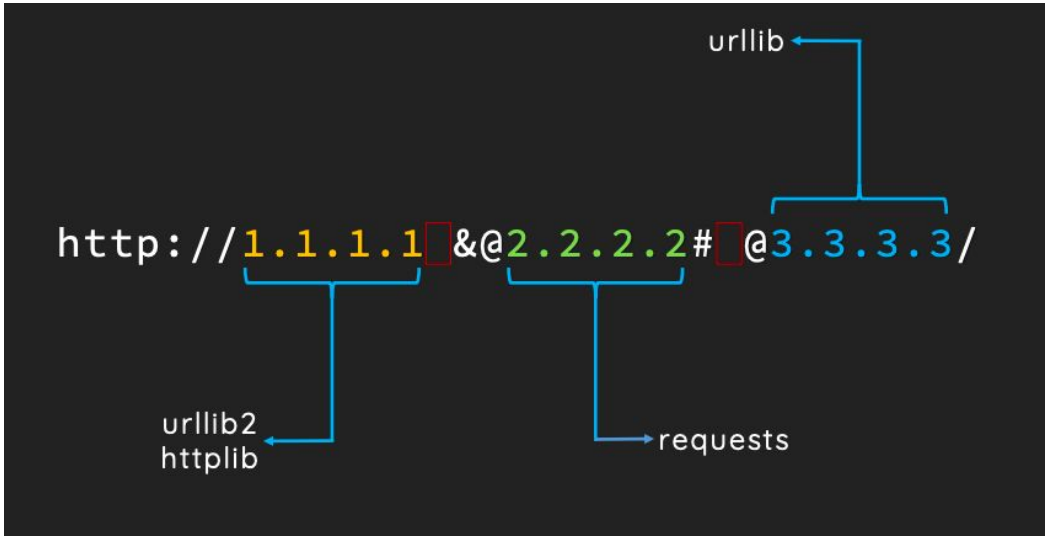


https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf

Differential Analysis

Differential analysis is the evaluation of the different outcomes that would arise from alternative solutions to a particular problem e.g.

- Parsing (URL, email, etc)
- Processing (archive extraction)
- ...



https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf

Combining Differential Analysis & Fuzzing

There is an increasing amount of research trying to combine Differential Analysis with Fuzzing

Financially motivated actor breaks certificate parsing to avoid detection

Sep 23, 2021 · 2 min read

Share

N

Neel Mehta
Threat Analysis Group

https://blog.google/threat-analysis-group/financially-motivated-actor-breaks-certificate-parsing-avoid-detection/

Detecting semantic bugs in the Linux kernel using differential fuzzing

Wednesday, 22 September 2021 07:10 (25 minutes)

Many bugs are easy to detect: they might cause assertions failures, crash our system, or cause other forms of undefined behaviour detectable by various dynamic analysis tools. However, certain classes of bugs, referred to as *semantic bugs*, cause none of these while still resulting in a misbehaving faulty system.

Differential fuzzing is a way to automate detection of semantic bugs by providing the same input to different implementations of the same systems and then cross-comparing the resulting behaviour to determine whether it is identical. In case the systems disagree, at least one of them is assumed to be wrong.

https://linuxplumbersconf.org/event/11/contributions/1033/contribution.pdf

Manual Code Review

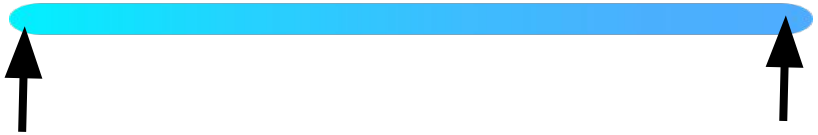
Manual Code Review

By discussing with several security researchers, most of them acknowledge that manual code review is still one of the best ways to catch novel new bugs

For some researchers, automation allows only to spot low hanging fruits or some complex bugs, but there is a whole area that is not covered by it

Variant Analysis and Fuzzing are tools that should be used to support manual code review, but not replace it

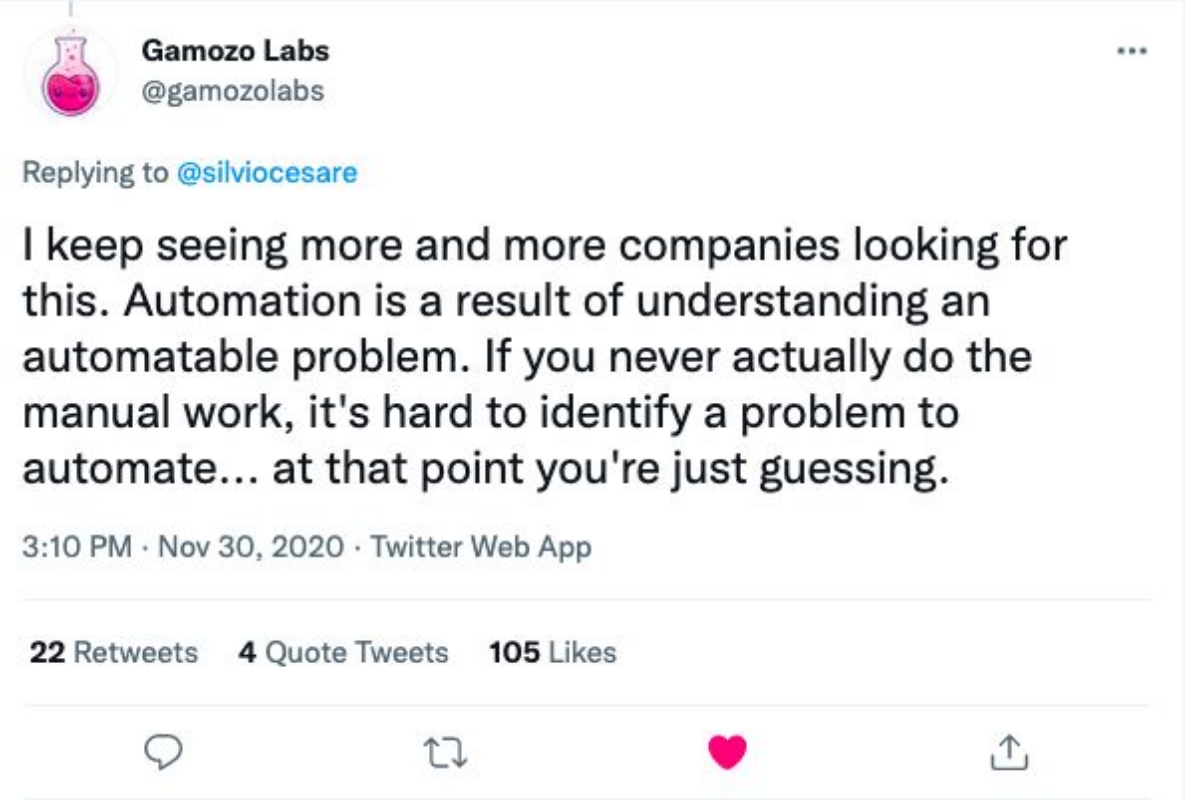
```
faultin_page
  handle_mm_fault
    __handle_mm_fault
      handle_pte_fault
        do_fault <- pte is not present
          do_cow_fault <- FAULT_FLAG_WRITE
            alloc_set_pte
              maybe_mkwrite(pte_mkdirty(entry), vma) <- mark the page dirty
                                                        but keep it RO
# Returns with 0 and retry
follow_page_mask
  follow_page_pte
    (flags & FOLL_WRITE) && !pte_write(pte) <- retry fault

faultin_page
  handle_mm_fault
    __handle_mm_fault
      handle_pte_fault
        FAULT_FLAG_WRITE && !pte_write
          do_wp_page
            PageAnon() <- this is CoWed page already
            reuse_swap_page <- page is exclusively ours
            wp_page_reuse
              maybe_mkwrite <- dirty but RO again
              ret = VM_FAULT_WRITE
((ret & VM_FAULT_WRITE) && !(vma->vm_flags & VM_WRITE)) <- we drop FOLL_WRITE

# Returns with 0 and retry as a read fault
cond_resched -> different thread will now unmap via madvise
follow_page_mask
  !pte_present && pte_none
faultin_page
  handle_mm_fault
    __handle_mm_fault
      handle_pte_fault
        do_fault <- pte is not present
          do_read_fault <- this is a read fault and we will get pagecache
                           page!
```



Low Hanging Fruits

Hard to Spot Vulnerabilities

Manual Code Review

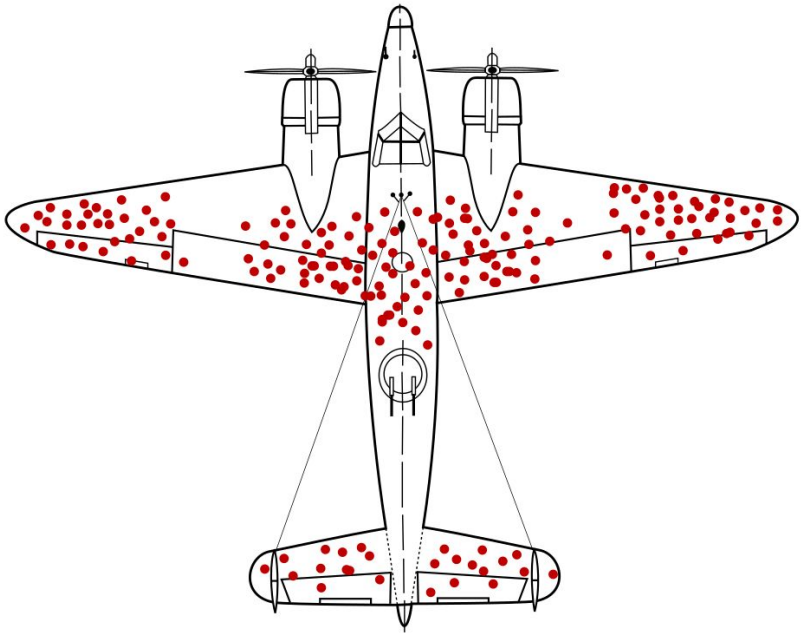


Conclusions

Survivorship Bias

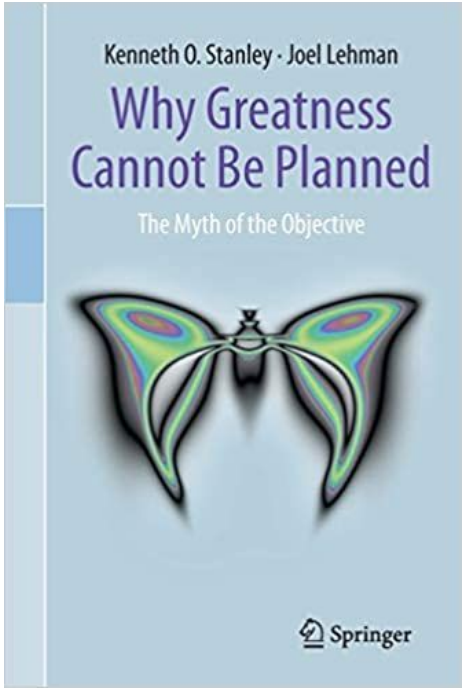
Automated security analysis in crucial but is not enough:

- What are the vulnerabilities we are missing?
- How to avoid a “research scope bias” in vulnerability research?



Avoiding Bias in Vulnerability Research Teams

- Don’t focus on goals/wrong KPIs (e.g. find a X number of vulnerabilities) but instead push more for Novelty Search strategy in your team
- If you work for a vendor don’t limit yourself to work with source code. Perform vulnerability research as bad actors are doing: this could trigger new ideas and spot new security bugs. Engineers love challenges!
- On top of process and tools hire smart people and allow them to do independent research
- Don’t focus only on automation, manual code analysis is what drives innovation



Takeaway


Rado RC1 @RabbitPro · 2h
Internal product security efforts promote finding as many bugs as possible that often just scratches the surface.
Attacker is goal oriented. Digging deep until the goal is reached ignoring useless vulns along the way.
Apply both methodologies as they expose different vulns.
2
17

Questions

@daviddiaul

Thanks to: @_pox_ @RabbitPro @pedrib1337 @m0n0sapiens @Lady_Librarian_1