

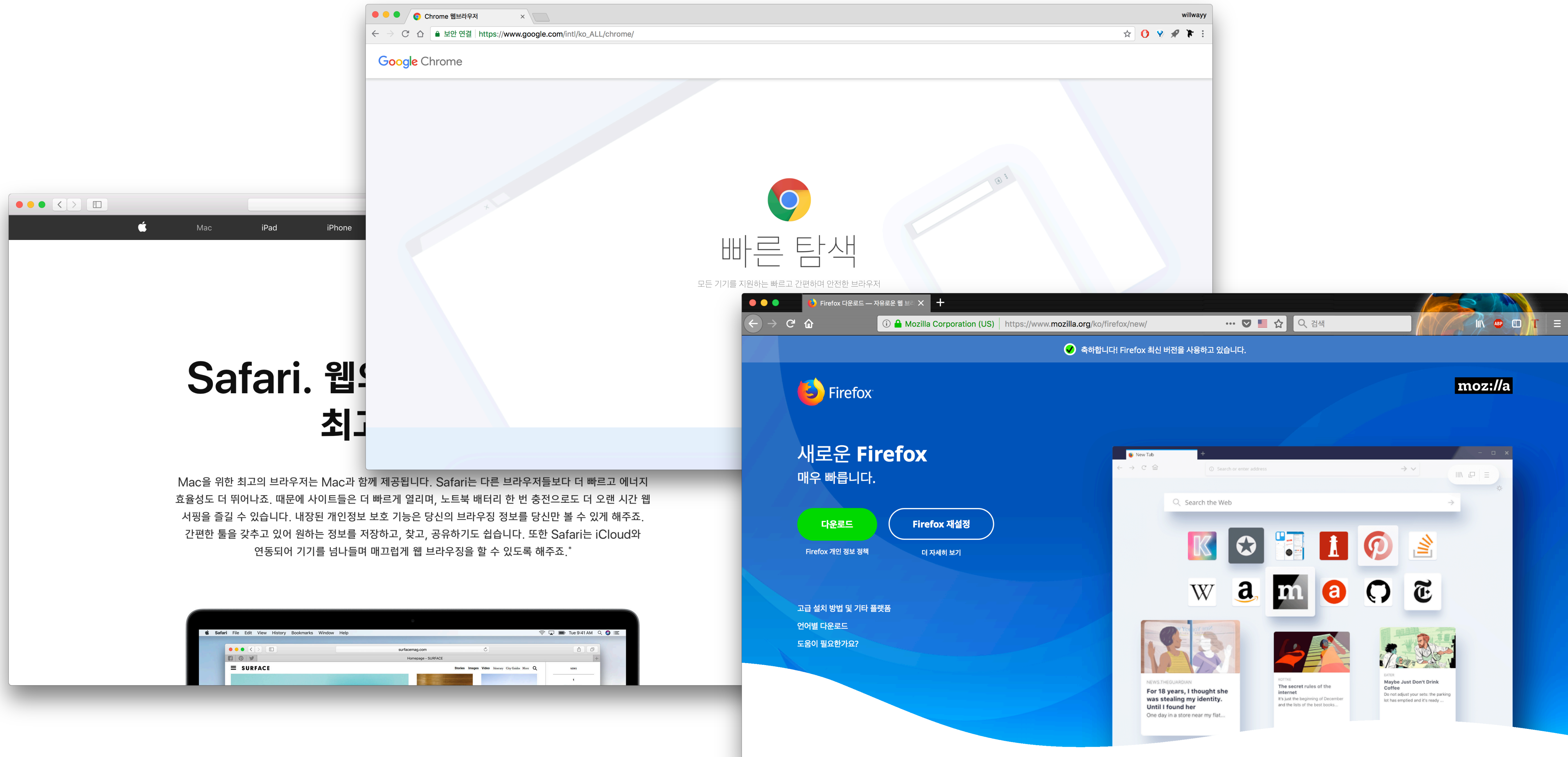


Browser Vulnerabilities Analysis

이정모 @IanLee

2019.01.15

Browser



Safari. 웹의 최고

Mac을 위한 최고의 브라우저는 Mac과 함께 제공됩니다. Safari는 다른 브라우저들보다 더 빠르고 에너지 효율성도 더 뛰어나죠. 때문에 사이트들은 더 빠르게 열리며, 노트북 배터리 한 번 충전으로도 더 오랜 시간 웹 서핑을 즐길 수 있습니다. 내장된 개인정보 보호 기능은 당신의 브라우징 정보를 당신만 볼 수 있게 해주죠. 간편한 툴을 갖추고 있어 원하는 정보를 저장하고, 찾고, 공유하기도 쉽습니다. 또한 Safari는 iCloud와 연동되어 기기를 넘나들며 매끄럽게 웹 브라우징을 할 수 있도록 해주죠.*

Chrome 웹브라우저

빠른 탐색

모든 기기를 지원하는 빠르고 간편하며 안전한 브라우저

Firefox 다운로드 — 자유로운 웹 브라우징

축하합니다! Firefox 최신 버전을 사용하고 있습니다.

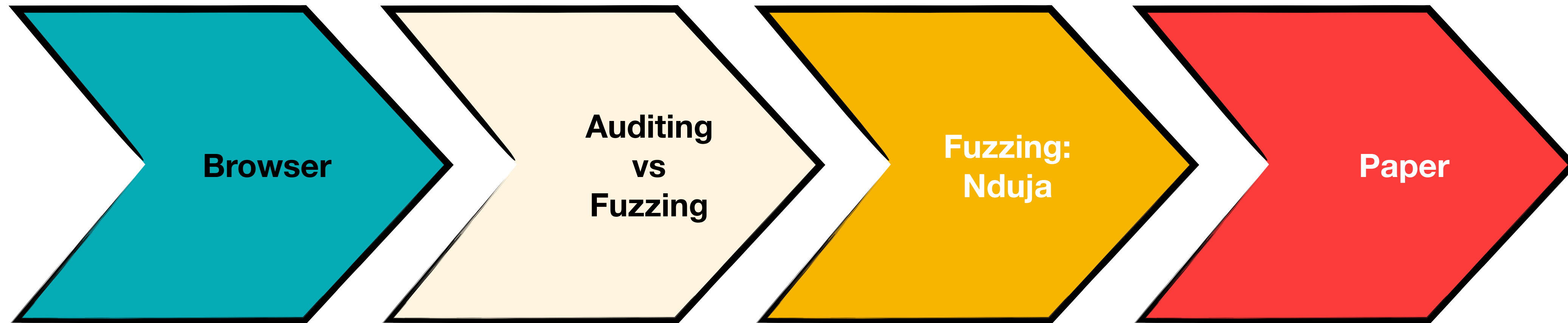
새로운 Firefox 매우 빠릅니다.

다운로드 Firefox 재설정

Firefox 개인 정보 정책 더 자세히 보기

고급 설치 방법 및 기타 플랫폼 언어별 다운로드 도움이 필요한가요?

Abstract



Browser Component Auditing

Browser Engine Fuzzing

...

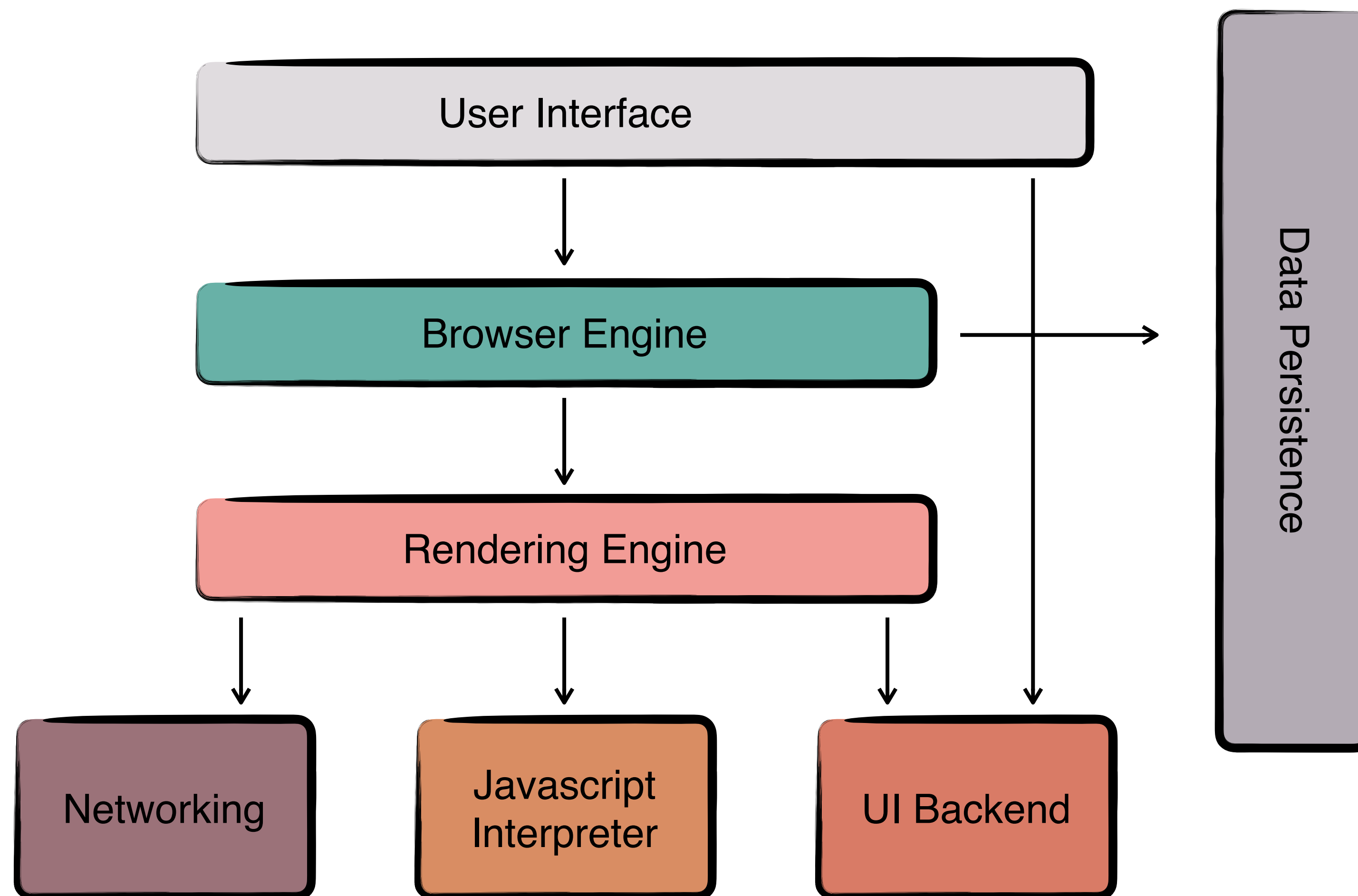
Nduja, fileja

grinder

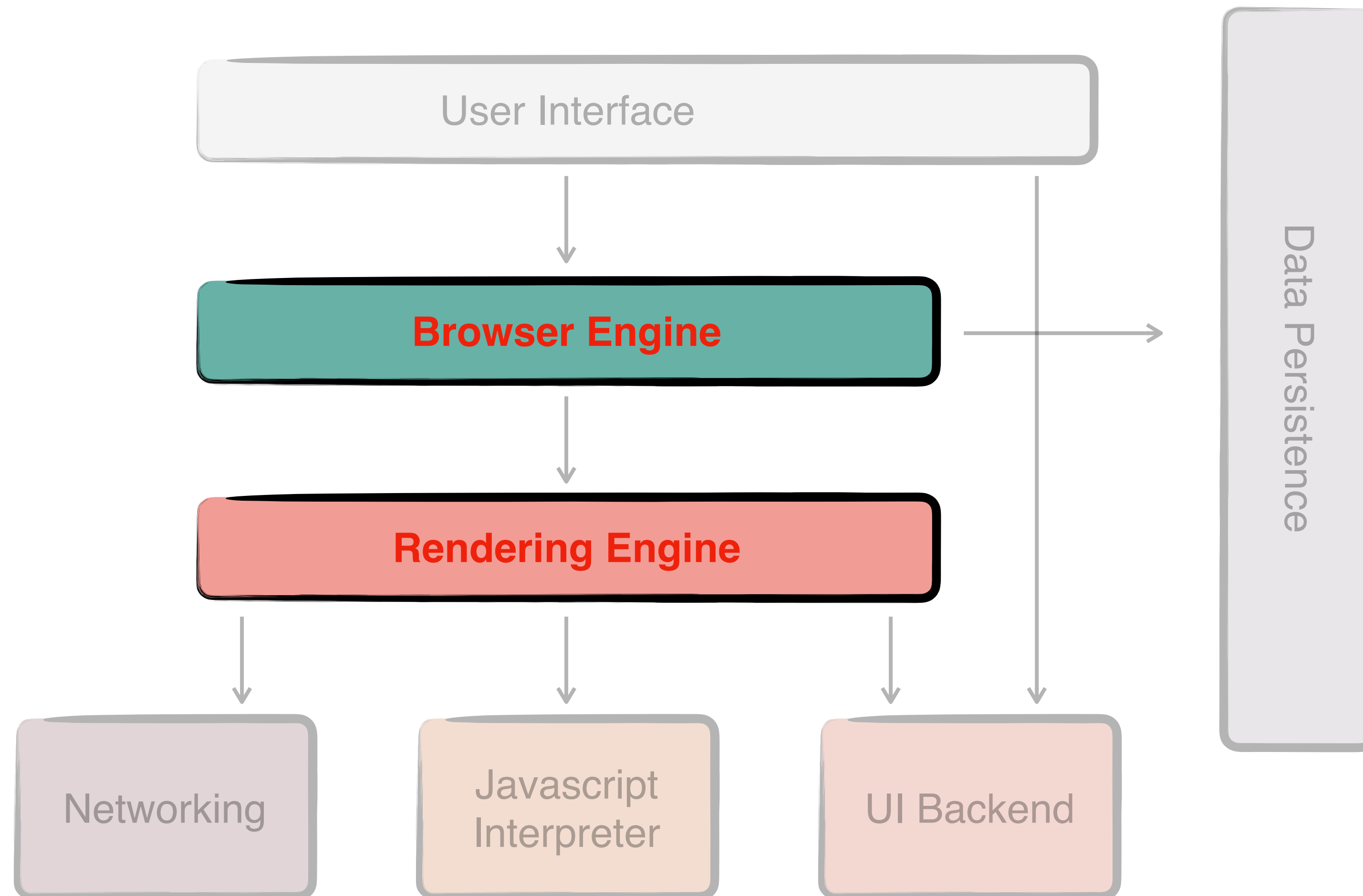
result

2019 CISC

Browser Component



Browser Component





- **Rendering Engine**

- Show content requested by the user
- For example, if you request HTML, it parses HTML and CSS And displays them on the screen.

- **JavaScript Engine**

- Interpreter role and JIT compilation utilize and have many functions, mainly used for web browsers

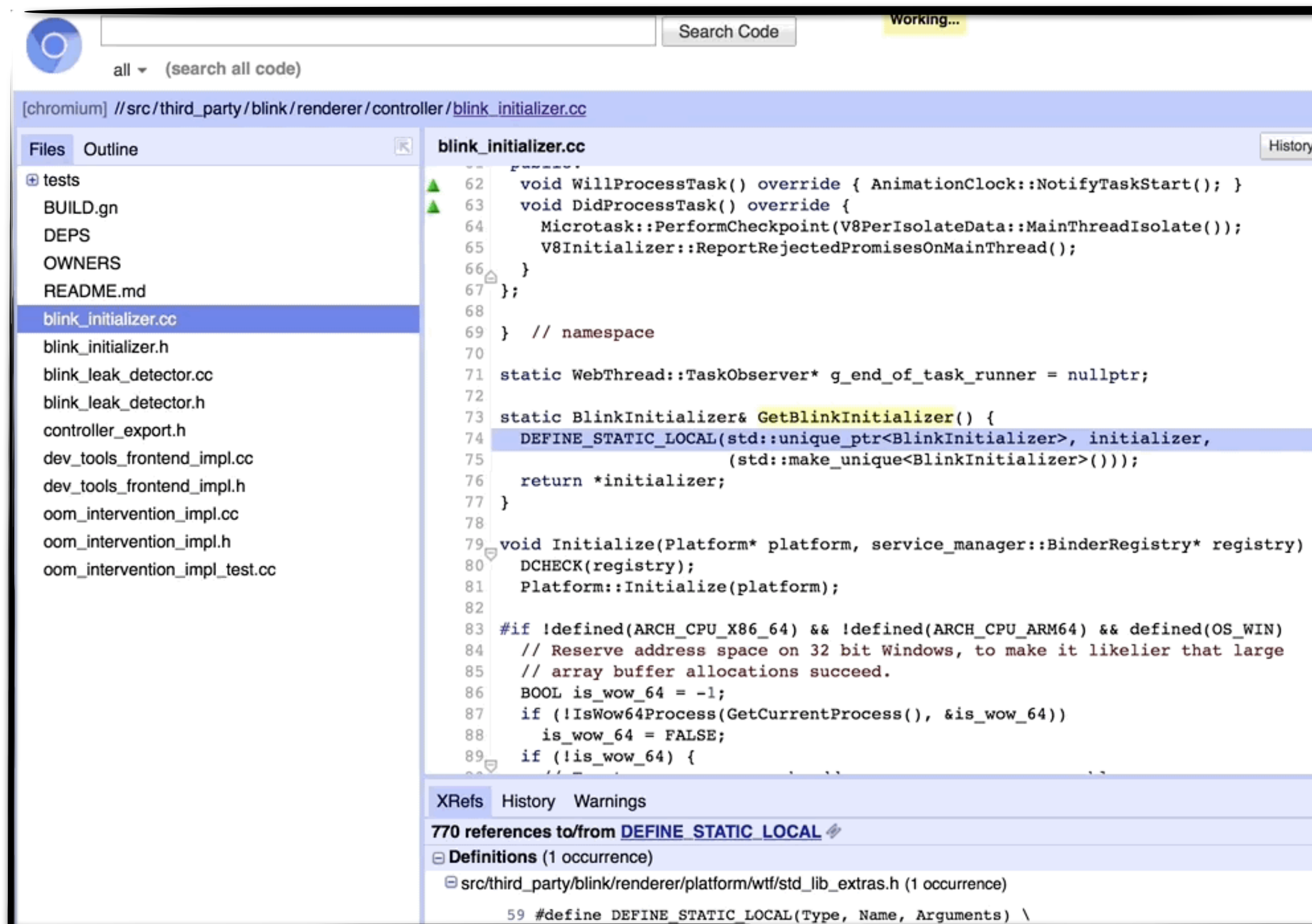
Method



Auditing

Fuzzing

- Auditing

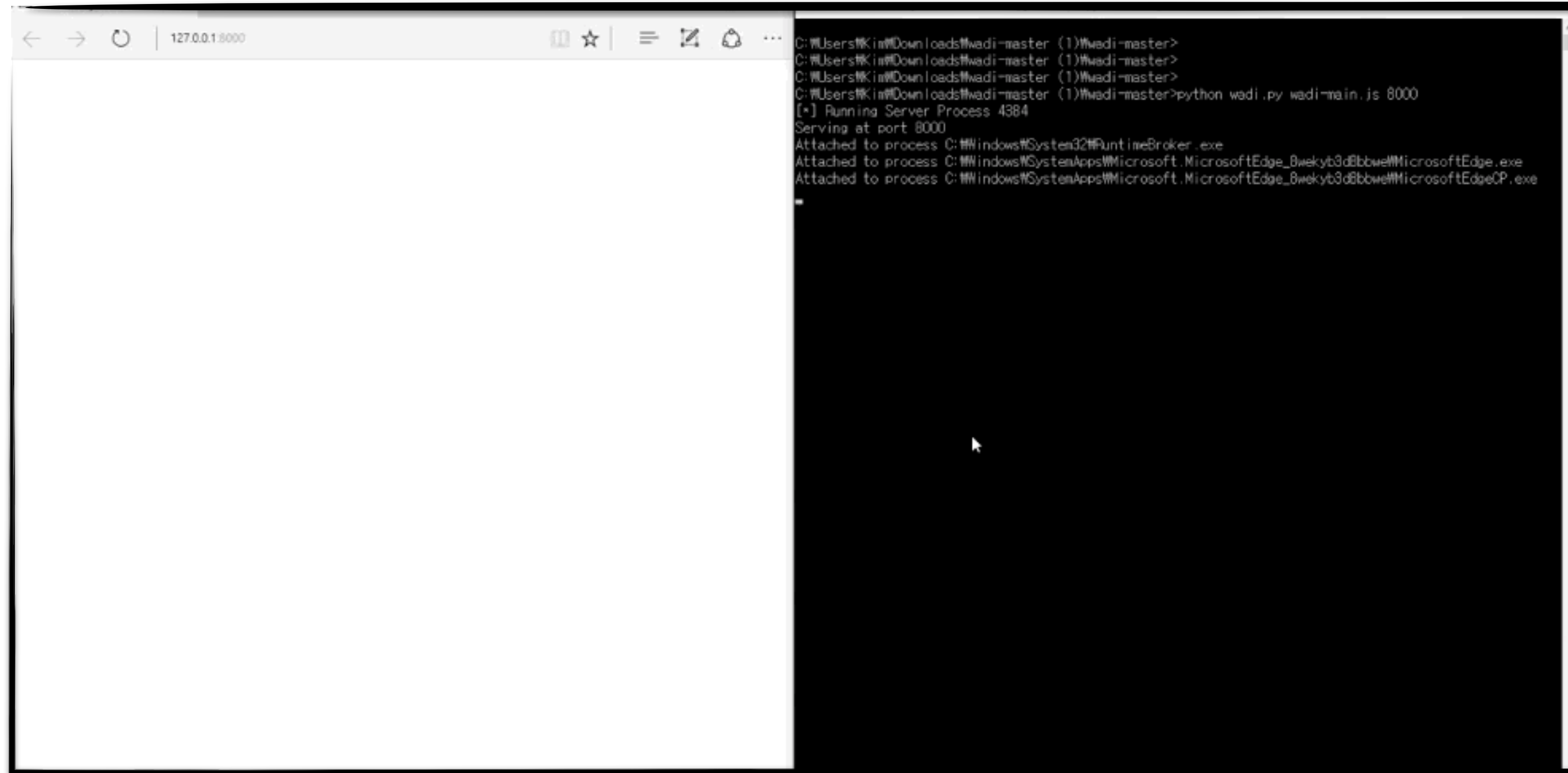


The screenshot shows a code editor interface with a search bar at the top containing "Search Code" and a "Working..." status. The search results show "all (search all code)". The main editor displays the file `blink_initializer.cc` with the following code snippet:

```
62 void WillProcessTask() override { AnimationClock::NotifyTaskStart(); }
63 void DidProcessTask() override {
64     Microtask::PerformCheckpoint(V8PerIsolateData::MainThreadIsolate());
65     V8Initializer::ReportRejectedPromisesOnMainThread();
66 }
67 };
68
69 } // namespace
70
71 static WebThread::TaskObserver* g_end_of_task_runner = nullptr;
72
73 static BlinkInitializer& GetBlinkInitializer() {
74     DEFINE_STATIC_LOCAL(std::unique_ptr<BlinkInitializer>, initializer,
75                         (std::make_unique<BlinkInitializer>()));
76     return *initializer;
77 }
78
79 void Initialize(Platform* platform, service_manager::BinderRegistry* registry) {
80     DCHECK(registry);
81     Platform::Initialize(platform);
82
83     #if !defined(ARCH_CPU_X86_64) && !defined(ARCH_CPU_ARM64) && defined(OS_WIN)
84         // Reserve address space on 32 bit Windows, to make it likelier that large
85         // array buffer allocations succeed.
86         BOOL is_wow_64 = -1;
87         if (!IsWow64Process(GetCurrentProcess(), &is_wow_64))
88             is_wow_64 = FALSE;
89         if (!is_wow_64) {
```

The search results at the bottom show "770 references to/from DEFINE_STATIC_LOCAL" and "Definitions (1 occurrence)" in `src/third_party/blink/renderer/platform/wtf/std_lib_extras.h` at line 59: `#define DEFINE_STATIC_LOCAL(Type, Name, Arguments) \`

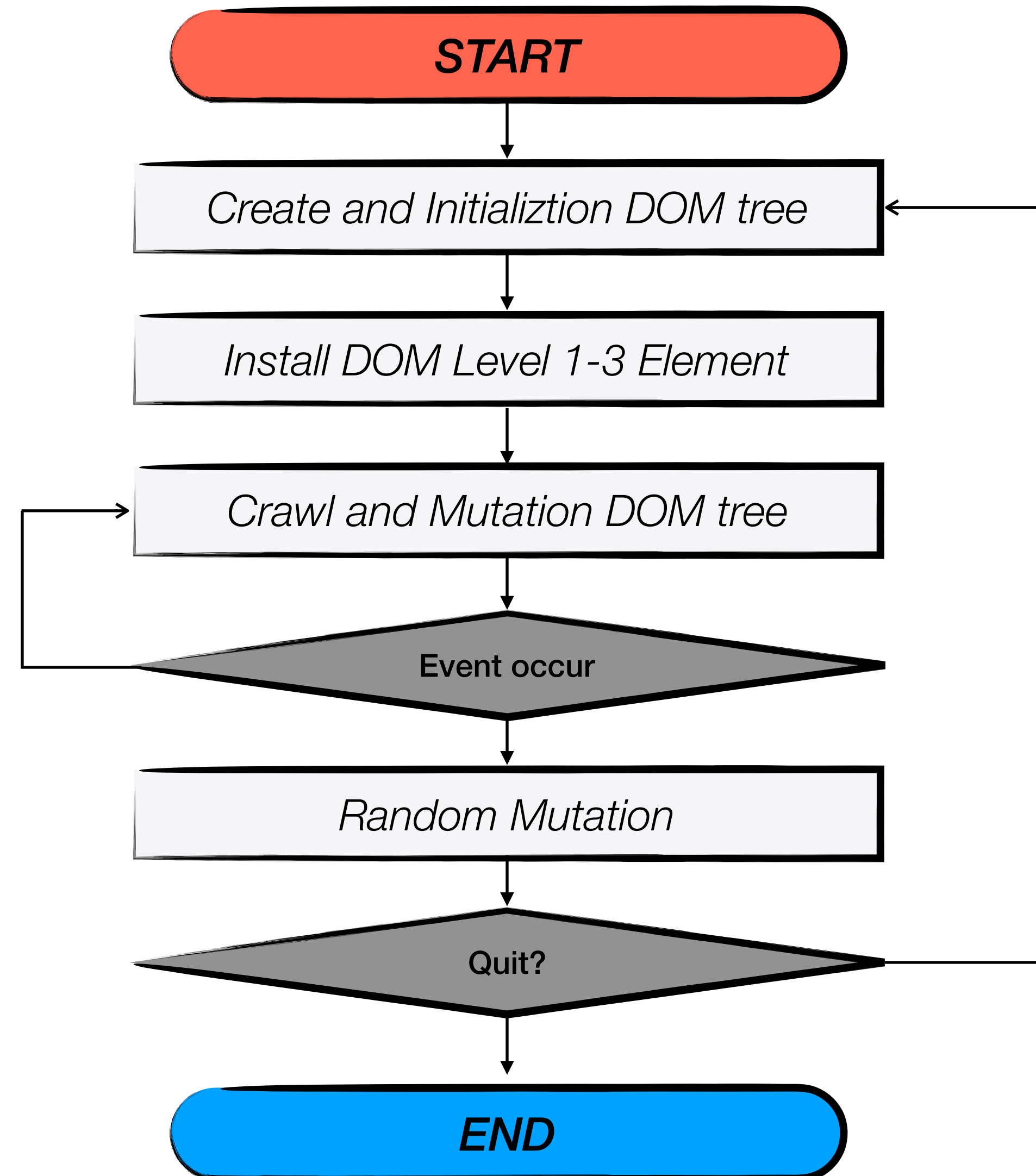
- Edge Fuzzing



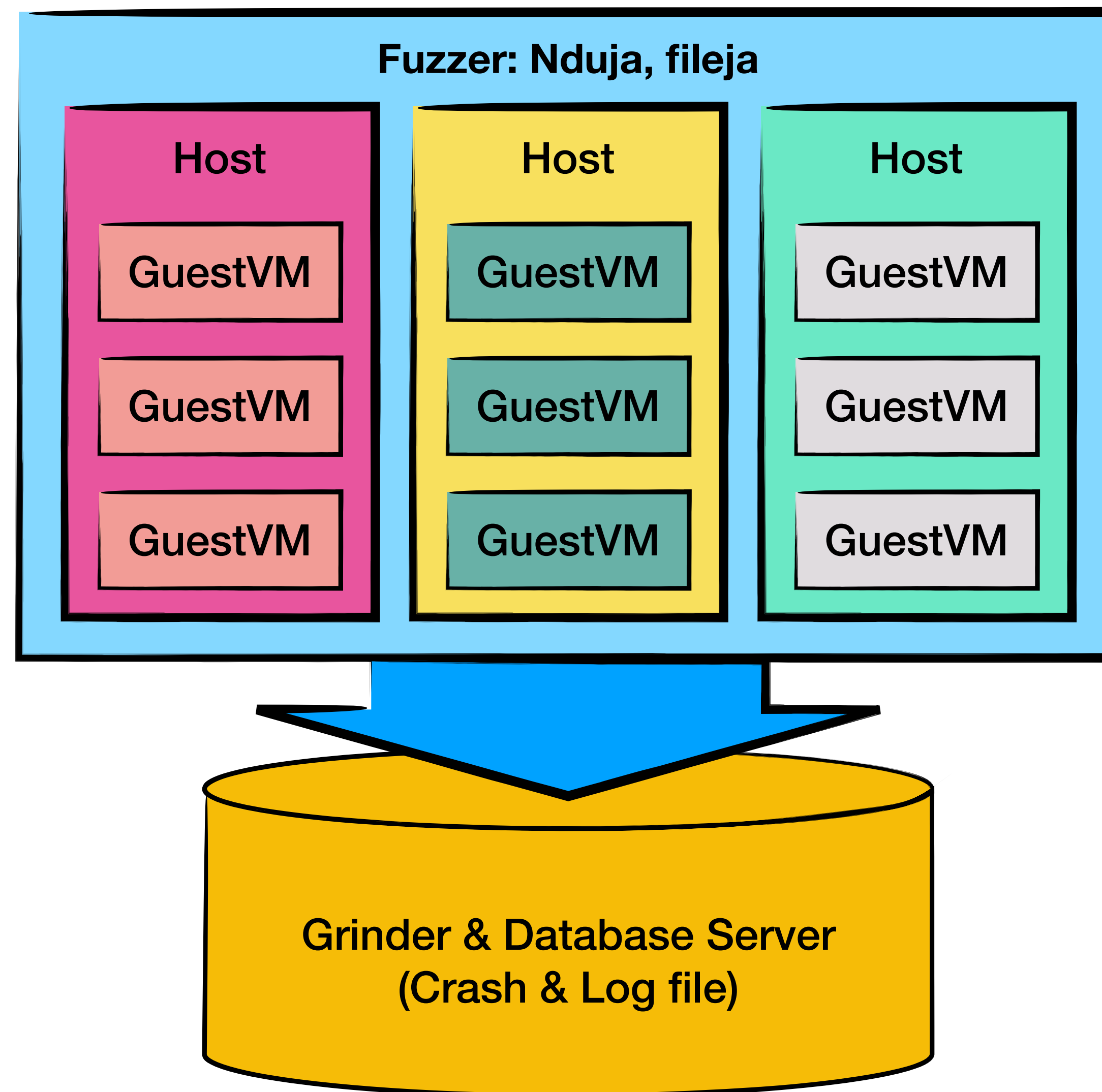
The screenshot shows a browser window on the left with the address bar set to 127.0.0.1:8000. On the right, a terminal window displays the following commands and output:

```
C:\Users\Kim\Downloads\wadi-master (1)\wadi-master>  
C:\Users\Kim\Downloads\wadi-master (1)\wadi-master>  
C:\Users\Kim\Downloads\wadi-master (1)\wadi-master>  
C:\Users\Kim\Downloads\wadi-master (1)\wadi-master>python wadi.py wadi-main.js 8000  
[*] Running Server Process 4384  
Serving at port 8000  
Attached to process C:\Windows\System32\FunTimeBroker.exe  
Attached to process C:\Windows\SystemApps\MicrosoftEdge_Bwkyb3d8bbwe\MicrosoftEdge.exe  
Attached to process C:\Windows\SystemApps\MicrosoftEdge_Bwkyb3d8bbwe\MicrosoftEdgeCP.exe
```

Nduja Fuzzing Mechanism



Fuzzing: Nduja, fileja




Fuzzing



Grinder - Mozilla Firefox

Grinder Preferences localhost Search



System Crashes Fuzzers Settings My Account

V NODE	TARGET	FUZZER	TYPE	HASH	TIME	COUNT
G2	IE11	nduja11	Stack Overflow	63ED2100.E8823A4F	2017-02-22 20:22:03	1
G2	IE11	nduja11	Stack Overflow	E51F6DC3.EE26A066	2017-02-22 19:18:42	1
G2	IE11	nduja11	Read Access Violation	89CB1AA1.B0408A2D	2017-02-20 19:09:55	1
G1	CM	nduja11	Write Access Violation	454078CF.88262F82	2017-02-17 17:48:21	1
G2	IE11	fileja0.4	Read Access Violation	D9A89B05.*	2017-02-12 13:06:05	6
G2	IE11	fileja0.4	Read Access Violation	A514E24F.EC370F85	2017-02-12 13:05:35	1
G2	IE11	nduja11	Stack Overflow	2D9C7F41.ECB36538	2017-02-12 07:04:31	1
G2	IE11	nduja11	Stack Overflow	E0931AB9.*	2017-02-12 03:16:52	3
G2	IE11	nduja11	Read Access Violation	4A97EDC2.*	2017-02-12 02:36:26	6
G1	CM	nduja11	Read Access Violation	A3FF996D.*	2017-02-11 19:18:00	25
G1	CM	nduja11	Read Access Violation	F574AA6F.*	2017-02-11 18:50:52	18
G1	CM	nduja11	Execute Access Violation	AE043A75.*	2017-02-11 18:23:04	27
G1	CM	nduja11	Read Access Violation	CEEA22E4.*	2017-02-11 18:19:15	18
G1	CM	nduja11	Execute Access Violation	A906BEE1.*	2017-02-11 17:34:59	537
G1	CM	nduja11	Read Access Violation	44F835BC.*	2017-02-11 17:33:57	46
G2	IE11	nduja11	Read Access Violation	B05C21B2.*	2017-02-11 15:34:27	6
G2	IE11	nduja11	Read Access Violation	050BB496.95EA35A7	2017-02-11 14:20:30	1
G2	IE11	nduja11	Read Access Violation	B2351D65.*	2017-02-11 13:21:07	339
G1	IE8	nduja11	Read Access Violation	86BEA3F1.95FB6EAA	2017-02-08 19:12:48	1

<< < Page 1 of 1 > >>
Jump To Page:

Options
Filters
Alerts

New Alert

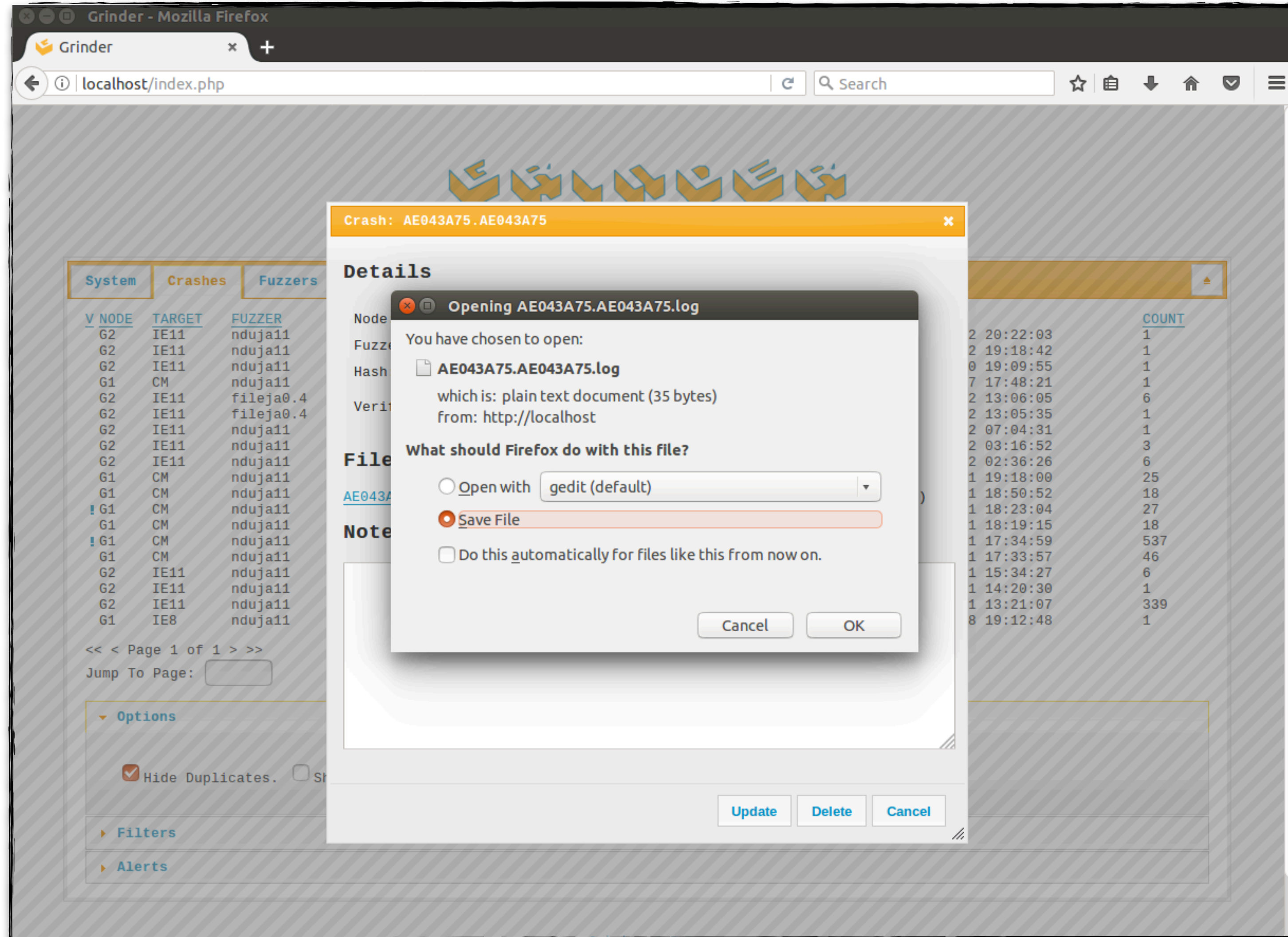
- Send an e-mail alert upon a new crash where the is equal to

Create

- Send an e-mail alert upon a new unique crash being generated.

Create

Fuzzing



Crash: AE043A75.AE043A75

V	NODE	TARGET	FUZZER	COUNT
	G2	IE11	nduja11	1
	G2	IE11	nduja11	1
	G2	IE11	nduja11	1
	G1	CM	nduja11	1
	G2	IE11	fileja0.4	1
	G2	IE11	fileja0.4	6
	G2	IE11	nduja11	1
	G2	IE11	nduja11	1
	G2	IE11	nduja11	3
	G2	IE11	nduja11	6
	G1	CM	nduja11	25
	G1	CM	nduja11	18
!	G1	CM	nduja11	27
	G1	CM	nduja11	18
!	G1	CM	nduja11	537
	G1	CM	nduja11	46
	G2	IE11	nduja11	6
	G2	IE11	nduja11	1
	G2	IE11	nduja11	339
	G1	IE8	nduja11	1

<< < Page 1 of 1 > >>
Jump To Page:

Options
 Hide Duplicates. Show Duplicates

Filters
Alerts

Opening AE043A75.AE043A75.log

You have chosen to open:
AE043A75.AE043A75.log
which is: plain text document (35 bytes)
from: http://localhost

What should Firefox do with this file?

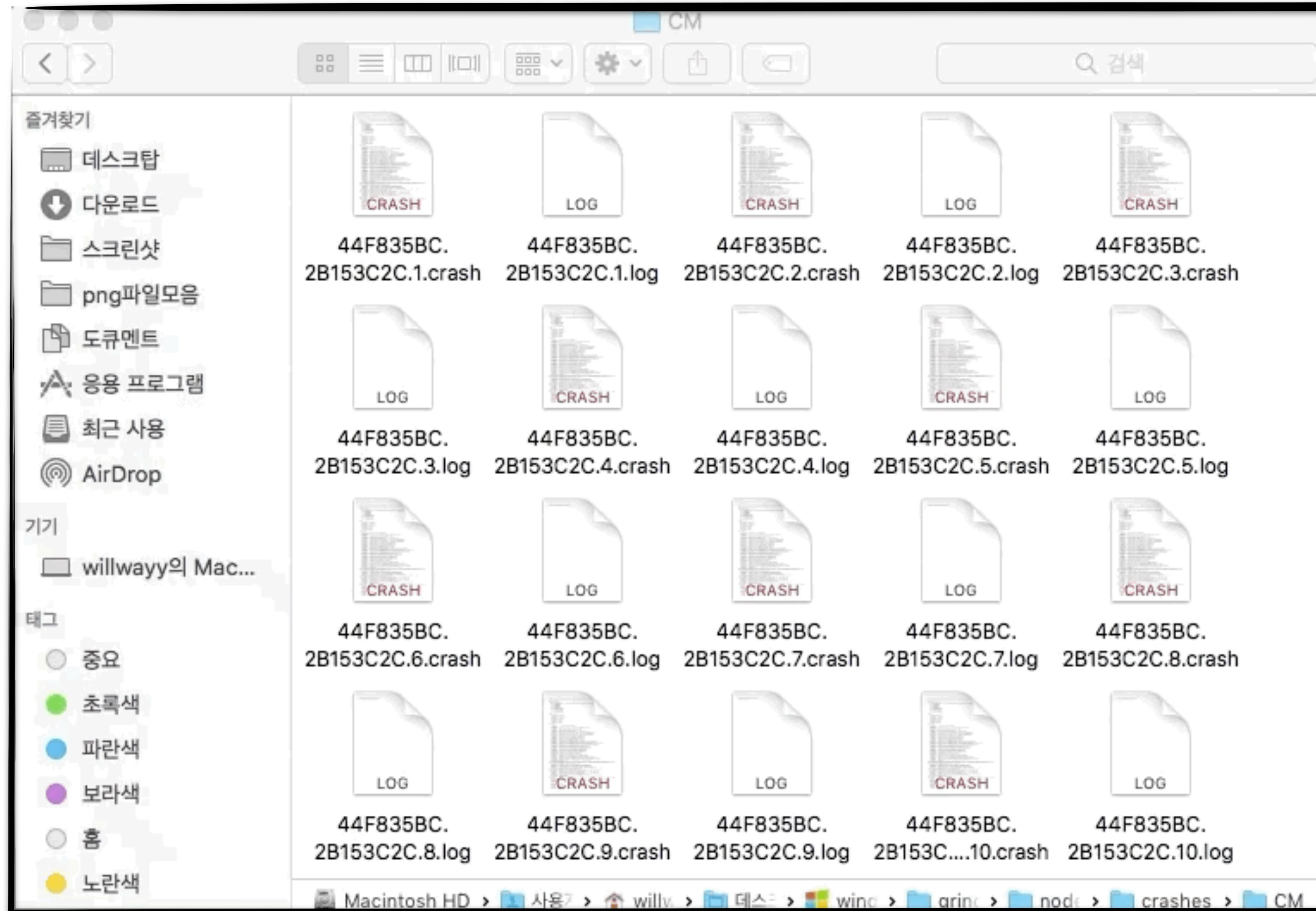
Open with gedit (default)

Save File

Do this automatically for files like this from now on.

Cancel OK

Crash



※.gif

Crash



NODE	TARGET	FUZZER	TYPE	COUNT
G2	IE11	nduja11	Stack Overflow	1
G2	IE11	fileja0.4	Read Access Violation	6
G1	Chrome	nduja11	Stack Overflow	9
G1	Chrome	nduja11	Read Access Violation	56
G1	Chrome	nduja11	Execute Access Violation	564
G1	Chrome	nduja11	Read Access Violation	43

Conclusion



- **So...**

- **Just crash?**
- Analyze crash dump file with windbg, troublizer, etc...
- Bypass process mitigation
- Kernel exploit to achieve EoP

- **Food for Thought**

- **Browser exploitation is getting harder and harder. But exploitation may still be possible using high quality vulnerabilities.**
- **If you go through this process, commercial programs can easily be hacked.**

Conclusion



- **So...**

- Just crash?
- **Analyze crash dump file with windbg, troublizer, etc...**
- **Bypass process mitigation**
- **Kernel exploit to achieve EoP**

This is the next project I'm doing right now.

- **Food for Thought**

- **Browser exploitation is getting harder and harder. But exploitation may still be possible using high quality vulnerabilities.**
- **If you go through this process, commercial programs can easily be hacked.**

Paper



구글 크롬 분석 방법론

이정모*

*중부대학교 정보보호학과

Google Chrome Analysis Methodology

Jung-Mo Lee*

*Division of information security, Joongbu University.

요 약

전 세계 많은 사람들이 컴퓨터와 인터넷을 사용하고 있고 거의 모든 컴퓨터에는 브라우저가 설치되어 있다. 특히 대다수 크롬 브라우저를 사용하고 있다. 많은 사람들이 크롬 브라우저를 사용하고 있는 만큼 크롬 이용자들을 위한 보안은 필수적이다. 더군다나 Google Chrome 의 목적 중 하나는 모든 작업을 특별한 설치 과정 없이 브라우저 내에서 모든 걸 이용하게끔 개발 중이기에 보안은 더욱 중요하다. 하지만 몇몇 해커들에 의해 Exploitation 이 현재도 많이 이루어지고 있고 해당 IT 기업 관계자들 합의하에 이루어지는 해킹대회인 Pwn2Own 에서도 구글 크롬이 항상 주요 타겟이 되어 Exploit 되고 있다.

본 논문에서는 이러한 해킹 과정들이 이루어지기 위해서는 분명히 취약점 분석 과정을 거쳐야 하는데 이러한 과정을 어떻게 접근해야 하며, 좀 더 효율적으로 chromium을 분석하기 위해서는 어떠한 Auditing 과정과 Fuzzing을 작업해야 하는지에 대해 다뤄볼 예정이다.

I. 서론

가구당 인터넷과 컴퓨터 보급, 보유량이 많아지면서 수많은 사람들은 브라우저를 이용해 인터넷을 사용하고 있다. 그리고 몇몇 브라우저 중 Google에서 Chromuim을 기반으로 만든 Chrome은 가장 많은 이용자를 가지고 있다. 그리고 이에 따라 악의적인 목적을 가진 해커들은 브라우저에 존재하는 취약점을 이용해 많은 문제를 일으키고 있다.

시만텍(Symantec)에서 발표한 2017년 주요 사이버 범죄 및 보안 위협 동향에 대한 분석을 담은 인터넷 보안 위협 보고서(ISTR) 제 23호에 따르면 웹에 관련된 악성코드 및 피싱이 가장 많이 이루어지고 있으며 이는 대부분 웹 브라우저에서 구동되는 자바 스크립트의 취약점을 노린 공격이다. 현재 지금도 Chrome 자바 스크립트 엔진인 V8에서 발생한 Use-after-Free 버그(CVE-2018-17465), BigInt64Array Out-

of-Bound Write (CVE-2018-16065) 취약점 등 철저한 보안과 거의 완벽에 가까운 소스코드를 가지고 있는 크롬이라 할지라도 최근에도 취약점이 활발하게 나오고 있고 이중 몇몇 브라우저 취약점들은 블랙마켓에서 거액으로 판매가 되고 있고 해당 취약점을 가지고 악의적인 스크립트 실행 및 사용자들의 컴퓨터를 위협하고 있다.

따라서 브라우저 취약점이 악의적인 의도를 가진 해커에 의해 악용되기 전에 미리 우리같은 화이트해커가 배포되기 전 개발자가 분석을 진행하는 버전인 Debug 버전과 일반 사용자들에게 배포되는 버전인 Release 버전 단계에서 취약점을 미리 발견해 보고해야한다. 이러한 취약점을 발견하는 방법에는 눈으로 소스코드를 읽고 수동적으로 취약점을 발견하는 방법인 Auditing 방법과 소프트웨어에 무작위 데이터를 반복하여 입력해 소프트웨어의 로직의 크래시를 유발함으로써 보안상의 취약점을 자동으로

찾아내는 Fuzzing을 이용해 찾는 방법이 존재한다.

본 논문에서는 Auditing을 진행하기 위해서는 어떠한 작업이 이루어져야하고 수많은 소스코드에서 효율적으로 Auditing을 진행하는 방법과 무작위 데이터를 반복하여 입력해 타겟 소프트웨어의 로직에서 크래시를 유발함으로써 보안상의 취약점을 찾아내는 Fuzzing을 이용할 것이고 본문에서는 크게 두 가지 방법인 Dump Fuzzing과 Mutation Fuzzing에 대해 다룰 예정이다.

현재 Nduja를 사용해 수백개의 크래시를 발견하였고 Google Project Zero 팀에서 개발한 Mutation Fuzzer인 Domato Fuzzer를 기반으로 제작한 특화된 Fuzzer를 만들어 다양한 취약점들을 확인하고 있다.

II. Auditing, Fuzzing

2.1 Auditing

Google의 Chromium은 모든 기능과 소스코드를 공개해 났으므로 쉽게 White Box Test를 진행해 별다른 작업 구성없이 cs.chromium.org에서 손쉽게 원하는 클래스나 템플릿 등 소스코드를 찾아보고 분석할 수 있다. 또한 구글에서는 항상 배포되는 버전에 대해 날짜를 공개해놓고 있는데 아래의 Table1에 나와 있다.

Table1. Chromium Development Calendar

Release	Week of Branch Point
65	Jan 18th, 2018
66	Mar 1st, 2018
67	Apr 12th, 2018
68	May 24th, 2018
69	Jul 19th, 2018
70	Aug 30th, 2018
71	Oct 11th, 2018
72	Nov 29th, 2018

이 표를 이용해 미리 분석해야하는 버전의 Chromium 파일을 다운로드 받아 소스코드를 분석할 수 있다.

2.1.1 DOM Bug CVE-2017-5052

이 취약점은 Blink에서 오디오를 통해 발견되었고 캐스팅되는 과정에서의 프로그래밍 오류가 있었다. 또한 디버그 모드에서만 작동하는 DCHECK 함수를 릴리즈 모드에서 작동하는 CHECK 함수로 작동시키기만 해도 Root 권한을 탈취할 수 있는 트리거가 된다. 해당 PoC코드는 다음과 같다.

```
PoC1. CVE-2017-5052

<html>
<head>
  <head>
    <title>::: reproduce-14fc2a :::</title>
  </head>
  <script>
    function start()
    {
      //make dom objects.
      o13 = document.createElement('frameset');
      o13.id = 'o13';

      o25 = document.createElement('time');
      o25.id = 'o25';

      o28 = document.createElement('listing');
      o28.id = 'o28';

      o161 = document.createElement('applet');
      o161.id = 'o161';

      o25.appendChild(o28.cloneNode(true));
      o161.appendChild(o25.cloneNode(true));
      document.body.appendChild(o161);
      document.body.appendChild(o13);
    }
  </script>
</head>
<body onload="start();">
</body>
</html>
```

2.2 Fuzzing

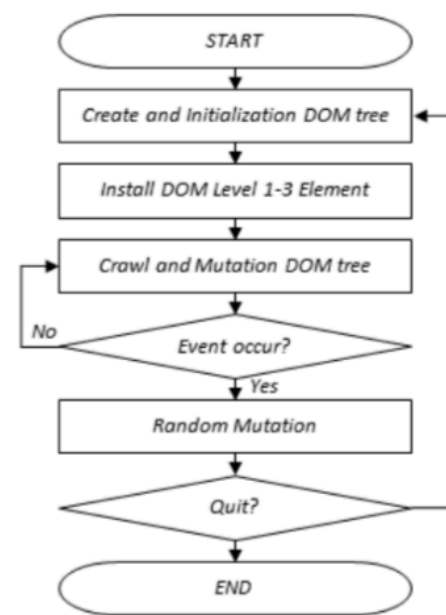
퍼징에는 크게 두가지 방법이 존재한다. 첫 번째로 덤 퍼징(Dump Fuzzing)은 무작위로 인풋(Input)을 대입하여 취약점을 유발시키는 것을 말하며 여기에서 파생되 이미 있는 데이터를 조금씩 수정하는 것을 Mutation Fuzzing이라고 한다. 또한 퍼징 대상을 제대로 이해하고 적절한 입력값을 넣어주는 방식을 스마트 퍼징(Smart Fuzzing)이라고 하며 마찬가지로 이것에 파생되 퍼징 데이터를 새롭게 만들어주며 대입시키는 것을 제너레이션 퍼징(Generation Fuzzing)이라고 한다. 그리고 두 방식을 결합한 방식의 퍼징인 하이브리드 퍼징(Hybrid Fuzzing)이 존재한다. 본 논문에서는 두가지 방식을 결합한 하이브리드 방식의 퍼징인 Nduja를 사용했다.

2.2.1 Nduja

Nduga는 Rosario Valotta가 개발한 Grinder 기반의 웹 브라우저 퍼징 도구이다. 기존에 존재하는 퍼저들은 대부분 DOM Level 1 또는 Level 2 에 대한 API를 이용한 퍼징을 수행하는 반면, Nduja에서는 DOM Level 3에 해당하는 이벤트 요소를 추가로 활용하여 좀 더 복잡하고 새로운 형태의 취약점을 발견하고자 하였다. 전체적인 퍼징 메커니즘은 Flow1과 같다.

Nduja의 주요 퍼징 알고리즘은 기본적으로 초기화 단계와 크롤링 및 DOM 트리 변형 단계, 이벤트 트리거 단계 등으로 이루어진다. 초기화 단계에서는 DOM 트리를 생성하고, 각 노드에 대한 속성에 임의의 값을 입력한다. 그리고 DOM 레벨 1의 노드 이터레이터(iterator), 레벨 2의 트리워커(treewalker), 레벨 3의 이벤트 리스너(listener)를 각각 설치한다. 크롤링 단계에서는 생성된 DOM 트리를 순회하며 무작위로 변형을 수행한다. 만약 설치한 이벤트가 발생할 경우, 해당 이벤트에 대한 핸들러에서는 다양한 변형 작업이 이루어짐으로서 크래시를 발생시키도록 한다. 이러한 퍼징 과정은 사용자가 임의로 종료할 때까지 페이지 새로 고침을 통해 반복적으로 수행된다.

Flow1. Nduja Fuzzing Mechanism



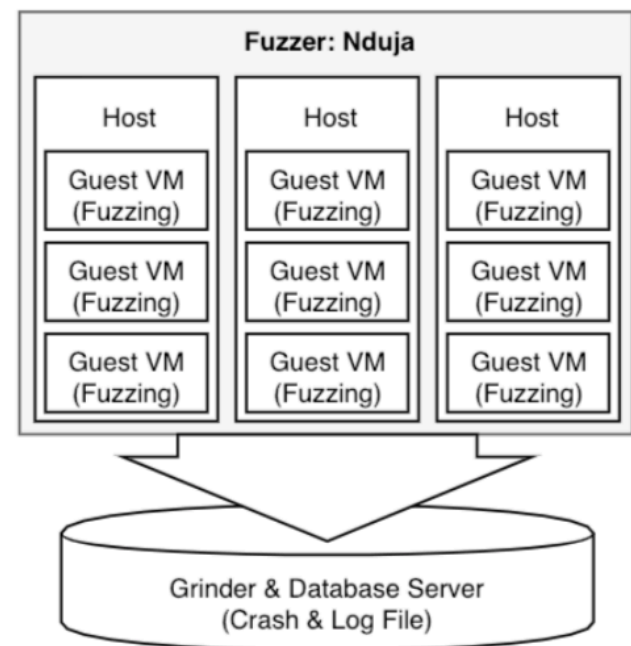
III. Nduja 환경 구성 및 방법

Nduja는 루비(Ruby) 언어로 개발된 웹 브라우저 퍼징 프레임워크인 글라이더(Grinder)를 기반으로 작동하며 전체적인 구성은 노드(Node)와 서버(Server)로 이루어져있다. 노드는 퍼징을 수행하는 자체를 의미하며, 서버는 글라이더를 통해 퍼징을 관리하고 작동시키는 역할을 한다. 각각의 시스템 사양은 Table2와 같으며 전체적인 구성은 아래의 Flow2와 같다.

Table2. System Specification of Experimental Environments

	Host	GuestVM	Server
OS	Win7	Win7(32bit)	Ubuntu14
S/W	Win7	VMware	Grinder
CPU	Intel i5	Intel i5	Intel i7
RAM	8GB	1GB	8GB

Flow2. Development Environment Map



각각의 노드는 브라우저에서 Crash가 발생되고 이에 관한 정보 및 결과를 생성하게 되는데 이는 grinder\node\crashes 디렉토리 내에 두개의 파일이 생성된다. 그리고 이 파일은 Grinder 서버로 전송되게 되고 .crash 파일에는 유용한 디버깅 정보가 담겨져 있다. (call stack, disassembly, register info 등) .log 파일에는 재사용이 가능한 테스트 케이스를 생성하는데 사

Paper



구글 크롬 분석 방법론

이정모*

*중부대학교 정보보호학과

Google Chrome Analysis Methodology

Jung-Mo Lee*

*Division of information security, Joongbu University.

요약

전 세계 많은 사람들이 컴퓨터와 인터넷을 사용하고 있고 거의 모든 컴퓨터에는 브라우저가 설치되어 있다. 특히 대다수 크롬 브라우저를 사용하고 있다. 많은 사람들이 크롬 브라우저를 사용하고 있는 만큼 크롬 이용자들을 위한 보안은 필수적이다. 더군다나 Google Chrome 의 목적 중 하나는 모든 작업을 특별한 설치 과정 없이 브라우저 내에서 모든 걸 이용하게끔 개발 중이기에 보안은 더더욱 중요하다. 하지만 몇몇 해커들에 의해 Exploitation 이 현재도 많이 이루어지고 있고 해당 IT 기업 관계자들 합의하에 이루어지는 해킹대회인 Pwn2Own 에서도 구글 크롬이 항상 주요 타겟이 되어 Exploit 되고 있다.

본 논문에서는 이러한 해킹 과정들이 이루어지기 위해서는 분명히 취약점 분석 과정을 거쳐야 하는데 이러한 과정을 어떻게 접근해야 하며, 좀 더 구체적으로 Chromium을 분석하기 위해서는 어떠한 Auditing 과정과 Fuzzing을 작업해야 하는지에 대해 설명한다.

I. 서론

가구당 인터넷과 컴퓨터 보급, 보유량이 많아지면서 수많은 사람들은 브라우저를 이용해 인터넷을 사용하고 있다. 그리고 몇몇 브라우저 중 Google에서 Chromium을 기반으로 만든 Chrome은 가장 많은 이용자수를 가지고 있다. 그리고 이에 따라 악의적인 목적을 가진 해커들은 브라우저에 존재하는 취약점을 이용해 많은 문제를 일으키고 있다.

시만텍(Symantec)에서 발표한 2017년 주요 사이버 범죄 및 보안 위협 동향에 대한 분석을 담은 인터넷 보안 위협 보고서(ISTR) 제 23호에 따르면 웹에 관련된 악성코드 및 피싱이 가장 많이 이루어지고 있으며 이는 대부분 웹 브라우저에서 구동되는 자바 스크립트의 취약점을 노린 공격이다. 현재 지금도 Chrome 자바 스크립트 엔진인 V8에서 발생한 Use-after-Free 버그(CVE-2018-17465), BigInt64Array Out-

of-Bound Write (CVE-2018-16065) 취약점 등 철저한 보안과 거의 완벽에 가까운 소스코드를 가지고 있는 크롬이라 할지라도 최근에도 취약점이 활발하게 나오고 있고 이중 몇몇 브라우저 취약점들은 블랙마켓에서 거액으로 판매가 되고 있고 해당 취약점을 가지고 악의적인 스크립트 실행 및 사용자들의 컴퓨터를 위협하고 있다.

따라서 브라우저 취약점이 악의적인 의도를 가진 해커에 의해 악용되기 전에 미리 우리같은 화이트해커가 배포되기 전 개발자가 분석을 진행하는 버전인 Debug 버전과 일반 사용자들에게 배포되는 버전인 Release 버전 단계에서 취약점을 미리 발견해 보고해야한다. 이러한 취약점을 발견하는 방법에는 눈으로 소스코드를 읽고 수동적으로 취약점을 발견하는 방법인 Auditing 방법과 소프트웨어에 무작위 데이터를 반복하여 입력해 소프트웨어의 로직의 크래시를 유발함으로써 보안상의 취약점을 자동으로

찾아내는 Fuzzing을 이용해 찾는 방법이 존재한다.

본 논문에서는 Auditing을 진행하기 위해서는 어떠한 작업이 이루어져야하고 수많은 소스코드에서 효율적으로 Auditing을 진행하는 방법과 무작위 데이터를 반복하여 입력해 타겟 소프트웨어의 로직에서 크래시를 유발함으로써 보안상의 취약점을 찾아내는 Fuzzing을 이용할 것이고 본문에서는 크게 두 가지 방법인 Dump Fuzzing과 Mutation Fuzzing에 대해 다룰 예정이다.

현재 Nduja를 사용해 수백개의 크래시를 발견하였고 Google Project Zero 팀에서 개발한 Mutation Fuzzer인 Domato Fuzzer를 기반으로 제작한 특화된 Fuzzer를 만들어 다양한 취약점들을 확인하고 있다.

II. Auditing, Fuzzing

2.1 Auditing
Chromium은 모든 기능의 소스코드를 공개한 오픈 소스 소프트웨어로 개발이 진행된다. 따라서 작업 구성없이 es.chromium.org에서 손쉽게 원하는 클래스나 템플릿 등 소스코드를 찾아보고 분석할 수 있다. 또한 구글에서는 항상 배포되는 버전에 대해 날짜를 공개해놓고 있는데 아래의 Table1에 나와 있다.

Table1. Chromium Development Calendar

Release	Week of Branch Point
65	Jan 18th, 2018
66	Mar 1st, 2018
67	Apr 12th, 2018
68	May 24th, 2018
69	Jul 19th, 2018
70	Aug 30th, 2018
71	Oct 11th, 2018
72	Nov 29th, 2018

이 표를 이용해 미리 분석해야하는 버전의 Chromium 파일을 다운로드 받아 소스코드를 분석할 수 있다.

2.1.1 DOM Bug CVE-2017-5052

이 취약점은 Blink에서 오디오를 통해 발견되었고 캐스팅되는 과정에서의 프로그래밍 오류가 있었다. 또한 디버그 모드에서만 작동하는 DCHECK 함수를 릴리즈 모드에서 작동하는 CHECK 함수로 작동시키기만 해도 Root 권한을 탈취할 수 있는 트리거가 된다. 해당 PoC코드는 다음과 같다.

```
PoC1. CVE-2017-5052

<html>
<head>
<title><!-- reproduce-14fc2a --></title>
</head>
<script>
function start()
{
//make dom objects.
o13 = document.createElement('frameset');
o13.id = 'o13';

o25 = document.createElement('time');
o25.id = 'o25';

o28 = document.createElement('listing');
o28.id = 'o28';

o161 = document.createElement('applet');
o161.id = 'o161';

o25.parentNode.appendChild(o28);
o25.parentNode.appendChild(o161);
}
</script>
</body>
</html>
```

2.2 Fuzzing

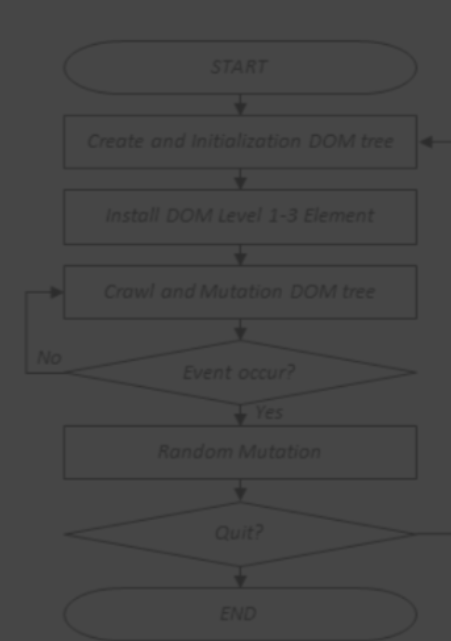
피징에는 크게 두가지 방법이 존재한다. 첫 번째로 덤 피징(Dump Fuzzing)은 무작위로 인풋(Input)을 대입하여 취약점을 유발시키는 것을 말하며 여기에서 파생되 이미 있는 데이터를 조금씩 수정하는 것을 Mutation Fuzzing이라고 한다. 또한 피징 대상을 제대로 이해하고 적절한 입력값을 넣어주는 방식을 스마트 피징(Smart Fuzzing)이라고 하며 마찬가지로 이것에 파생되 피징 데이터를 새롭게 만들어주며 대입시키는 것을 제너레이션 피징(Generation Fuzzing)이라고 한다. 그리고 두 방식을 결합한 방식의 피징인 하이브리드 피징(Hybrid Fuzzing)이 존재한다. 본 논문에서는 두가지 방식을 결합한 하이브리드 방식의 피징인 Nduja를 사용했다.

2.2.1 Nduja

Nduja는 Rosario Valotta가 개발한 Grinder 기반의 웹 브라우저 피징 도구이다. 기존에 존재하는 피저들은 대부분 DOM Level 1 또는 Level 2 에 대한 API를 이용한 피징을 수행하는 반면, Nduja에서는 DOM Level 3에 해당하는 이벤트 요소를 추가로 활용하여 좀 더 복잡하고 새로운 형태의 취약점을 발견하고자 하였다. 전체적인 피징 메커니즘은 Flow1과 같다.

Nduja의 주요 피징 알고리즘은 기본적으로 초기화 단계와 크롤링 및 DOM 트리 변형 단계, 이벤트 트리거 단계 등으로 이루어진다. 초기화 단계에서는 DOM 트리를 생성하고, 각 노드에 대한 속성에 임의의 값을 입력한다. 그리고 DOM 레벨 1의 노드 이터레이터(iterator), 레벨 2의 트리워커(treewalker), 레벨 3의 이벤트 리스너(listener)를 각각 설치한다. 크롤링 단계에서는 생성된 DOM 트리를 순회하며 무작위 피징을 수행한다. 피징이 완료된 후 이벤트가 발생하면, 해당 노드의 속성에 임의의 값을 입력한다. 이 과정에서 크래시를 유발할 수 있다. 이 과정을 반복하여 사용자 기기의 중요할 때까지 피징을 새로 고침을 통해 반복적으로 수행된다.

Flow1. Nduja Fuzzing Mechanism



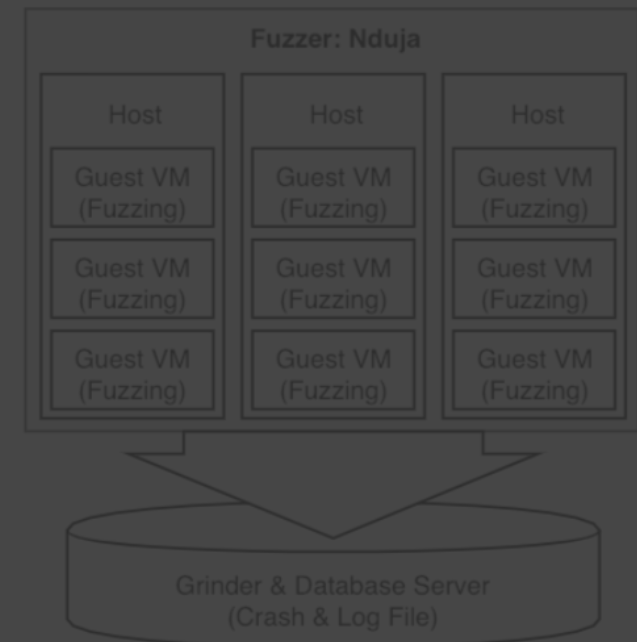
III. Nduja 환경 구성 및 방법

Nduja는 루비(Ruby) 언어로 개발된 웹 브라우저 피징 프레임워크인 글라인더(Grinder)를 기반으로 작동하며 전체적인 구성은 노드(Node)와 서버(Server)로 이루어져있다. 노드는 피징을 수행하는 자체를 의미하며, 서버는 글라인더를 통해 피징을 관리하고 작동시키는 역할을 한다. 각각의 시스템 사양은 Table2와 같으며 전체적인 구성은 아래의 Flow2와 같다.

Table2. System Specification of Experimental Environments

	Host	GuestVM	Server
OS	Win7	Win7(32bit)	Ubuntu 14
S/W	Win7	VMware	Grinder
CPU	Intel i5	Intel i5	Intel i7
RAM	8GB	1GB	8GB

Flow2. Development Environment Map



각각의 노드는 브라우저에서 Crash가 발생되고 이에 관한 정보 및 결과를 생성하게 되는데 이는 grinder\node\crashes 디렉토리 내에 두개의 파일이 생성된다. 그리고 이 파일은 Grinder 서버로 전송되게 되고 .crash 파일에는 유용한 디버깅 정보가 담겨져 있다. (call stack, disassembly, register info 등) .log 파일에는 재사용이 가능한 테스트 케이스를 생성하는데 사

Thank you



QnA