하이브리드 퍼저를 이용한 브라우저 취약점 분석

이정모, 이병천

*중부대학교 정보보호학과

Analysis of Browser Vulnerabilities using Hybrid Fuzzer

Jungmo Lee, Byoungcheon Lee

*Department of Information Security, Joongbu University

요 약

최근 웹 기반 서비스가 증가하면서 브라우저의 취약점을 이용한 웹서비스 공격이 증가하고 있어서 브라우저의 취약점에 대한 선제적인 대응이 필요하다. 브라우저 취약점을 탐색하기 위한 작업에는 Auditing과 Fuzzing이 존재하는데 본 논문에서는 하이브리드(Hybrid) 퍼저(Fuzzer)인 Nduja를 이용해 브라우저의 취약점을 분석해본다. 퍼징을 진행하면서 여러 다양한 크래시를 수집하고 수집한 결과들을 분석하는 과정을 제시한다.

I. 서론

최근 웹을 기반으로 하는 서비스가 증가하면서 많은 사람들은 브라우저를 이용해 다양한 종류의 인터넷 서비스들을 사용하고 있다. 그중에서 구글이 Chromuim 엔진을 기반으로 만든 크롬 브라우저는 현재 가장 많은 이용자수를 가지고 있다[1]. 악의적인 목적을 가진 해커들은 브라우저에 존재하는 취약점들을 이용해 많은 문제를 일으키고 있다.

시만텍(Symantec)에서 발표한 2017년 사이버 범죄 및 보안 위협 동향에 대한 분석을 담은 인터넷 보안 위협 보고서(ISTR)[2]에 따르면 웹에 관련된 악성코드 및 피싱이 가장 많이 이루어지고 있으며 이는 대부분 웹 브라우저에서 구동되는 자바스크립트의 취약점을 노린 공격이다. 철저한 보안과 완벽에 가까운 소스코드를 가지고 있다고 공언하는 크롬 브라우저이지만 지금도 Chrome 자바 스크립트 엔진인 V8에서 발생한 Use-after-Free 버그(CVE-2018-17465), BigInt64Array Out-of-Bound Write (CVE-2018-16065) 등 취약점이 많이 발견되고 있다. 이중 몇몇 취약점들은 블랙마켓에서 거액으로 판매가 되고 있고 공격자들은 해당 취약점을 가지고 악의적인 스크립트를 실행하는 등 사용자들의 컴퓨터를 위협하고 있다.

따라서 브라우저 취약점이 악의적인 의도를 가진 해커에 의해 악용되기 전에 개발자가 분석을 진행하는 버전인 Debug 버전과, 일반 사용자들에게 배포되는 버전인 Release 버전 단계에서 크래시(Crash) 및 취약점을 미리 발견해 개선하도록 해야 한다. 이러한 취약점들을 발견하는 방법에는 눈으로 소스코드를 읽고 수동적으로 취약점을 발견하는 방법인 오디팅(Auditing) 방법과 소프트웨어에 무작위 데이터를 반복하여 입력해 소프트웨어의 로직에 크래시를 유발함으로써 보안상의 취약점을 자동으로 찾아내는 퍼징(Fuzzing)을 이용하는 방법이 있다.

본 논문에서는 하이브리드(Hybrid) 퍼저(Fuzzer)인 Nduja를 이용해 브라우저의 취약점을 분석하는 과정을 살펴보고 그 대응책을 제시한다.

II. 퍼징

2.1 퍼징 기법의 종류

퍼징에는 Table. 1에서 제시한 바와 같이 다양한 기법들이 사용된다. 본 논문에서는 효율성이 높은 하이브리드 퍼징 방식을 사용한다.

Table 1. Classification of Fuzzing Techniques

퍼징 기법	설 명
Dumb Fuzzing	입력의 형태를 모르는 환경에서 임의의 값으로 입력 데이터를 생성 또는 변이하는 퍼징 기법
Smart Fuzzing	입력의 형태를 하는 환경에서 입력의 타입 또는 형태에 따라 입력 데이터를 생성 또는 변이하는 퍼징 기법
Mutation Fuzzing	기존에 정상적인 입력 데이터의 일부를 조작 하여 새로운 입력 데이터를 만들어 내는 퍼징 기법
Generation Fuzzing	분석된 입력의 형태에 따라 적합한 새로운 입력 데이터를 만들어 내는 퍼징 기법
Hybrid Fuzzing	Mutation과 Generation 퍼징 방법의 장점을 결합한 방식으로 비교적 높은 커버리지를 가지므로 예측하기 어려운 상황들을 다양하게 이끌어내는 퍼징 기법

2.2 퍼징 도구 Nduja

Nduja[3]는 Rosario Valotta가 개발한 웹브라우저 퍼징 도구이다. 기존의 퍼져들은 DOM Level 1 또는 Level 2에 대한 API를 이용한 퍼징을 수행했지만 Nduja에서는 DOM Level 2 및 Level 3 API를 많이 활용하고 좀 더 복잡한 퍼징 알고리즘을 이용해 다양한 크래시와 새로운 형태의 취약점을 발견할 수 있다. 전체적인 퍼징 메커니즘은 Fig. 1과 같다[7].

Nduja는 Javascript로 만들어진 퍼저이며 이를 이용하여 무작위로 DOM 요소들을 만들고 DOM 처리 함수를 무작위로 호출하며 임의로 DOM 요소를 삭제하는 등 다양하고 복잡한 과정을 뒤섞어 브라우저를 분석한다. 이런 과정을 거치게 되면 다양한 객체가 할당되고 해제되는 과정을 반복하는데 이 과정에서 어떠한 객체가 이미 메모리가 해제된 객체에 접근하게 되는 Use-After-Free(UAF) 버그가 발생할 수 있으며 이는 브라우저 해킹에서 가장 많이 이용되는 버그이다. 이 도구는 많은 UAF 유형의 취약점을 발견했으며 2012 년경 브라우저 취약점 패치와 악용에 가장 많이 사용된 퍼저로 알려졌다.

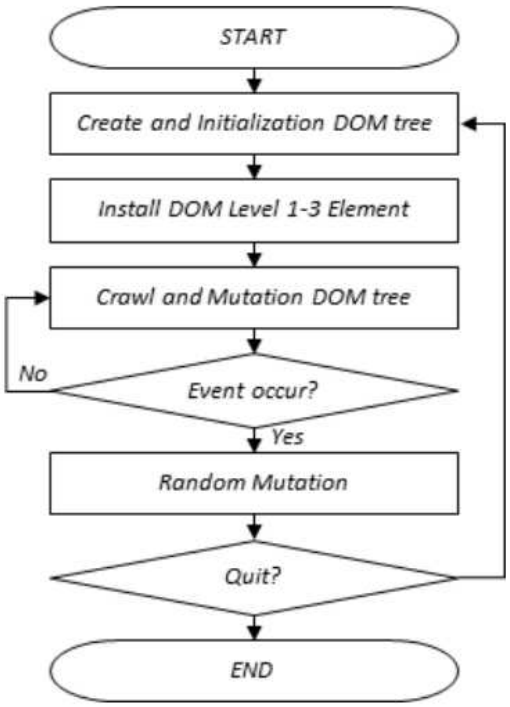

Fig. 1 Nduja Fuzzing Mechanism

III. Nduja 환경 구성 및 방법

Nduja는 루비(Ruby) 언어로 개발된 웹 브라우저 퍼징 프레임워크인 그라인더(Grinder)[8]를 기반으로 작동하며 전체적인 구성은 노드(Node)와 서버(Server)로 이루어져 있다. 노드는 퍼징을 수행하는 단위 객체를 의미하며, 서버는 그라인더를 통해 퍼징을 관리하고 크래시를 분류해 수집하는 등 관리 역할을 한다.

전체적인 운영 환경을 Table 2 및 Fig. 2에 제시하였다. 여기에서 호스트는 여러 PC를 돌리고 있는 것처럼 구성하기 위해 VMware가 설치된 PC를 의미하며, 노드는 하나의 호스트 컴퓨터에 설치된 여러개의 GuestVM에 설치된 그라인더 노드를 의미한다. 서버는 노드에서 생

성된 크래시를 분류하고 관리하기 위해 그라인더 서버가 설치된 우분투 환경의 PC를 말한다.

Table 2. System Configuration of Experimental Environments

	Host	GuestVM	Server
OS	Win7	Win7(32bit)	Ubuntu14
CPU	Intel i5	Intel i5	Intel i7
RAM	8GB	1GB	8GB

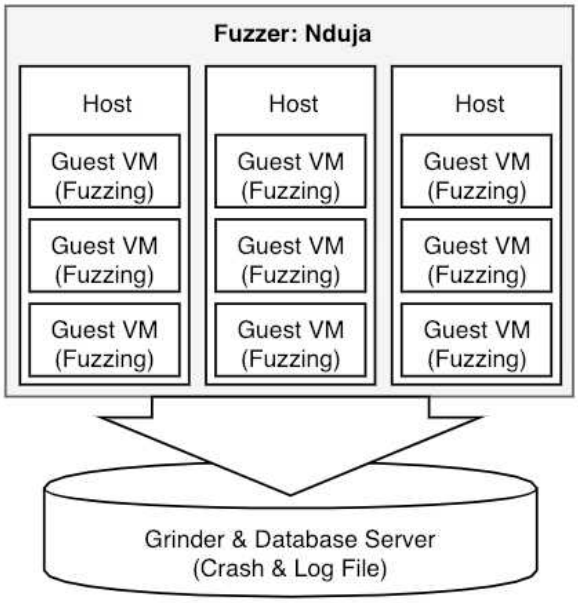

Fig. 2 Operations in Developed Environment

각각의 노드에 있는 브라우저에서 크래시가 발생하면 노드 내의 ~/grinder/node/crashes 디렉토리 내 2개의 파일(.crash, .log)에 이들 정보가 저장된다. 그리고 이 파일들은 그라인더 서버로 전송된다. .crash 파일에는 call stack, disassembly, register info 등 유용한 디버깅 정보가 담겨져 있으며 .log 파일에는 재생성(reproduce)이 가능한 테스트 케이스 정보가 저장된다. 이것은 브라우저를 퍼징하는 과정에서 생성된 로깅(logging) 정보가 들어 있으므로 크래시를 트리거 할 수 있다. 또한 크래시 파일을 이용한 브라우저 공격의 성공 가능성은 디버거 모듈이 대상 브라우저 프로세스를 후킹하고 Windbg에 있는 !exploitable API를 사용하여 판단할 수 있다. 서버로 전송되는 이러한 정보들은 보안상 민감한 내용이므로 암호화되어 전송하도록 되어 있다. 그라인더의 장점 중 하나는 웹 서버를 구축해 놓은 상태이므로 외부에서 로그인하여 퍼저들이 정상적으로 작동하고 있는지, 어떠한 취약점이 발생되고 있는지 외부에서도 분석이 가능하도록 크래시 파일과 로그 파일을 다운로드 및 상세보기가 가능하다.

IV. 실험 및 결과

퍼저는 Nduja를 사용했으며 하나의 호스트 내에 여러 퍼저를 구동시키기 위하여 가상으로 환경을 구성할 수 있는 VMware를 이용해 여러 노드를 구성하고 IE11, Chrome 61을 대상으로 퍼징을 진행하였다. 해당 버전들은 실험 당시 가장 최신 버전을 대상으로 진행했다. 이렇게 구축한 시스템으로 약 7일간 퍼징을 수행하였고 퍼징한 결과는 Table 3 와 같다.

Table. 3. Experimental Results

Target	Type	Count
IE11	Stack Overflow	6
Chrome	Read Access Violation	61
Chrome	Write Access Violation	1
Chrome	Execute Access Violation	564

이 실험에서는 IE와 크롬을 포함해 약 630여 개의 크래시가 발생하였으며 다양한 취약점을 탐지할 수 있었다. 오류로 분류될 수 있는 크래시에 한해서만 수집하였으며 이것은 공격자가 Heap Feung Shui[9], Heap Spray[10] 공격 등을 통해 원하는 코드를 주입한 후, 해당 코드로 실행흐름을 변경할 수 있는 가능성이 높기 때문이다.

V. 크래시 사례 분석

위에서 생성된 덤프 파일과 크래시 로그는

매우 많기 때문에 그중에 콜스택(Call Stack)을 볼 수 있는 하나의 크래시 로그파일을 사례로 분석해본다.

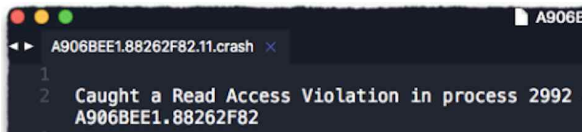

Fig. 3 Chrome Crash File

Fig. 3의 사례는 Read Access Violation의 사례로 해당 프로세스가 읽기(Read) 권한이 없는 메모리 영역에 접근해 발생한 오류이다. 이는 메모리 주소가 누출되어 권한이 없는데도 필요한 주소나 정보를 읽어올 수 있다는 오류이다.

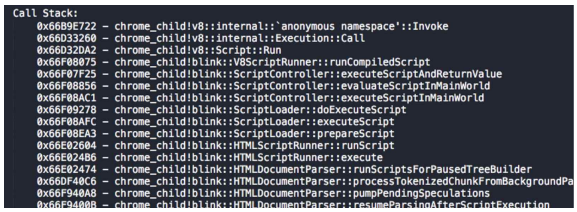

Fig. 4 Call Stack of Chrome Crash

위의 사례에 대한 콜스택을 보면 맨 아래에 있는 함수가 최초로 실행된 곳이며 맨 위에 있는 함수가 결론적으로 문제가 발생한 함수 지점을 나타낸다. Windbg를 사용해 이러한 덤프 파일을 분석한다. 크로미움은 모든 함수에 대해 소스가 공개 되어 있으므로 Windbg로 디버깅 시 심볼 서버를 크로미움으로 설정한다[11]. 그리고 kb(Display Stack Backtrace) 명령어를 이용해 콜스택을 확인하고 Windbg 기능 중 하나인 Local view(로컬 창으로 변수 보기)와 소스코드를 보며 최초 혹은 중간 지점부터 호출한 함수에서 시작해 맨 위에 있는 오류 발생 지점 함수까지 따라가며 분석을 진행하게 된다.

VI. 결론 및 향후 연구

본 논문에서는 현재 보안 시장의 뜨거운 이슈로 부상하고 있는 웹브라우저의 취약점에 대해 분석하는 방법 중 대표적인 방법인 퍼징에 대해 소개하였다. 그 중 Nduja11을 이용해 하이브리드 퍼징을 진행하였고, 그 결과 많은 크래시 데이터를 얻을 수 있었다.

그런데 취약점 제보를 목적으로 브라우저 취약점 분석을 하고 있는 연구자들이라면 모두 이러한 퍼져들을 돌리고 있을 것으로 생각되는데 같은 퍼징 알고리즘 및 엔진을 이용해 취약점을 점검하면 비슷하거나 똑같은 취약점이 나올 것이다. 또한 구글에서는 이미 브라우저를 구성하고 있는 전체적인 클래스를 대상으로 대규모로 퍼징 테스트를 진행하고 있다. 그러므로 우리가 발견한 크래시를 해당 소프트웨어 벤더사에 제보를 해도 중복될 가능성이 높게 된다. 따라서 이러한 문제점들을 극복하기 위해서는 좀 더 다양한 방법론을 사용하는 퍼저 엔진을 제작하기 위한 연구개발이 필요하며, 여러 웹 API 중 소수의 선택된 클래스를 대상으로 취약점을 집중적으로 찾을 수 있는 퍼저를 개발하여 이용할 필요가 있다.

또한 취약점 발견에만 몰두하지 말고 크래시 결과를 분석하는 과정에서 존재하는 메모리 보호기법에 대한 연구도 필요하다. 즉 특정 프로세스가 사용하는 메모리 주소를 무작위로 부여하여 익스플로잇(Exploit)을 방어하기 위한 ASLR(Address Space Layout Randomization)이나 외부 접근 및 영향을 차단하여 제한된 영역 내에서만 프로그램을 동작시키는 Sandbox와 같은 기법들을 우회할 수 있는 공격기술을 연구해야 할 필요가 있다.

[참고문헌]

[1] Browser Market Share Worldwide http://gs.statcounter.com/

[2] Symantec Internet Security Threat Report https://www.symantec.com/content/dam/symantec/ko/docs/infographics/istr-23-infrastructure-attacks-stealthy-mining-threats-go-big-and-small-ko.pdf

[3] Rosario Valotta, “Taking Browsers Fuzzing To The Next (DOM) Level”, DeepSec 2012.

[4] Document Object Model (DOM) Level 1 Specification https://www.w3.org/TR/DOM-

Level-1/

[5] Document Object Model (DOM) Level 2 Core Specification https://www.w3.org/TR/DOM-Level-2-Core/

[6] Document Object Model (DOM) Level 3 Core Specification https://www.w3.org/TR/DOM-Level-3-Core/

[7] Event and Command based Fuzzing Method for Verification of Web Browser Vulnerabilites http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.836.8133&rep=rep1&type=pdf

[8] Grinder, https://github.com/stephenfewer/grindjer.

[9] Alexander Sotirov 'Heap Feng Shui in JavaScript' https://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf

[10] Heap Spraying https://en.wikipedia.org/wiki/Heap_spraying

[11] Debugging Chromium on Windows https://www.chromium.org/developers/how-tos/debugging-on-windows