공학석사 학위논문

차등 테스트를 활용한 브라우저 핑거프린팅 식별 도구 설계

Finding Browser Fingerprinting Vector Using Differential Testing

2023년 2월

서울대학교 대학원

전기·정보공학부

곽 진 한

공학석사 학위논문

차등 테스트를 활용한 브라우저 핑거프린팅 식별 도구 설계

Finding Browser Fingerprinting Vector Using Differential Testing

2023년 2월

서울대학교 대학원

전기·정보공학부

곽 진 한

차등 테스트를 활용한 브라우저 핑거프린팅 식별 도구 설계

Finding Browser Fingerprinting Vector Using Differential Testing

지도교수 이 병 영

이 논문을 공학석사 학위논문으로 제출함

2023년 2월

서울대학교 대학원

전기·정보공학부

곽 진 한

곽진한의 공학석사 학위논문을 인준함

2023년 2월

위 원 장 _____ 백 윤 흥 _____ (인)

부위원장 _____ 이 병 영 _____ (인)

위 원 _____ 심 재 웅 _____ (인)

초록

브라우저 핑거프린팅은 브라우저에서 제공하는 기능을 활용하여 웹 클라이언트를 식별하기 위한 일련의 정보들을 수집하는 행위를 의미한다. 브라우저 핑거프린팅은 브라우저에서 제공하는 기능을 통해 수행되며, 특히 브라우저의 구성 요소 중 자바스크립트 코드 수행을 위한 자바스크립트 엔진과 웹 컨텐츠를 클라이언트에게 보여주기 위한 작업을 수행하는 렌더링 엔진을 중심으로 많은 기능들을 제공하고 있다. 브라우저에서 제공하는 기능을 통해 브라우저 핑거프린팅이 수행되는 만큼 해당 기능들에 대한 전반적인 이해와 분석이 요구된다. 이에 본 논문에서는 브라우저 핑거프린팅에 활용될 수 있는 핑거프린팅 벡터를 식별할 수 있는 도구를 제안하였다. 문법적으로 올바르면서도 랜덤 테스팅을 통한 브라우저 입력 생성을 통해 자바스크립트 API를 테스트할 수 있는 것과 동시에 차등 테스트를 수행함으로써 유저의 도움 없이 자바스크립트 API의 브라우저 핑거프린팅 활용 가능성 여부를 식별할 수 있도록 설계하였다. 그 결과 75개의 클라이언트의 브라우저 종류 또는 버전 관련 정보를 유추할 수 있는 브라우저 핑거프린팅 벡터를 찾았다.

주요어: 브라우저 핑거프린팅, 차등 테스트
학번: 2020-25377

목차

표 목차

그림 목차

제 1 장 서론

웹 트래킹(Web Tracking)은 웹 환경에서 활동하는 유저를 식별하기 위해 해당 유저와 관련된 정보를 수집하는 행위를 의미한다. 웹 트래킹은 일반적으로 각각의 유저들에게 최적화된 정보를 제공하기 위한 수단으로 활용된다. 예를 들어, 웹 서버에 접근한 클라이언트가 사용하고 있는 디바이스의 화면의 크기에 대한 정보를 알 수 있다면 웹 서버는 클라이언트에게 제공할 웹 컨텐츠를 클라이언트의 디바이스의 화면 크기에 최적화되어 있는 형태로 가공하여 제공할 수 있다.

웹 트래킹을 위해 활용할 수 있는 정보 중 가장 대표적인 것은 쿠키(Cookie)이다. 쿠키는 유저가 브라우저 등을 통해 웹 페이지에 접근할 때 생성되는 일련의 데이터로, 방문한 웹 페이지와 관련된 쿠키 정보가 유저의 컴퓨터에 저장되며, 웹 페이지에서는 해당 쿠키 정보를 기반으로 유저를 식별할 수 있다.

그러나, 최근 수집된 정보를 통해 브라우저를 활용하는 유저를 식별하는 행위가 유저의 프라이버시를 침해할 수 있는 잠재적 위협으로 인식됨에 따라, 브라우저 벤더들은 유저들의 프라이버시 보호를 위한 대책이 논의되거나 마련되고 있다. 구글(Google)은 크롬(Chrome) 브라우저에서 써드-파티 쿠키(Third-party Cookie)의 사용을 2024년까지 단계적으로 제거할 예정이라고 발표했으며 [1], 모질라(Mozila)는 파이어폭스(Firefox) 브라우저에서 써드-파티 쿠키를 차단하는 기능을 제공하고 있다 [2].

쿠키 기반의 웹 트래킹에 대한 보호 대책이 마련됨에 따라 브라우저 핑거프린팅 기술이 주목을 받고 있다. 브라우저에서 지원하는 API를 통한 브라우저 핑거프린팅 기술이 소개된 이후 [3], 브라우저 핑거프린팅에 브라우저에서 2D 이미지를 그리기 위해 활용되는 Canvas API와 3D 그래픽 이미지를 그리기 위해 활용되는 WebGL API도 브라우저 핑거프린

팅에 활용될 수 있다는 것이 밝혀졌다 [4] [5]. 한편, 브라우저 핑거프린팅이 프라이버시 관점에서 얼마나 정확하게 유저를 식별할 수 있는지를 확인하기 위한 연구들도 진행되었다 [6] [7] [8] [9] [10] [11] [12]. 또한, 이전 연구들을 통해 밝혀진 브라우저 핑거프린팅에 활용될 수 있는 특정 API를 분석하여 근본적인 원인을 분석하고 그에 대한 해결 방법을 제안하거나 [13], 브라우저 핑거프린팅으로 인한 잠재적인 프라이버시 침해로부터 웹 클라이언트를 보호하기 위한 방법론을 제안한 연구도 수행된 바 있다 [14] [15].

브라우저 핑거프린팅은 브라우저에서 제공하는 API를 통해 수행되기 때문에 브라우저에서 제공하는 모든 자바스크립트 API는 잠재적으로 유저의 프라이버시를 침해할 위협이 될 수 있다. 또한, 브라우저 핑거프린팅은 호출되는 API의 종류나 개수에 비례하여 특정 유저를 유일하게 (uniquely) 식별할 수 있는 가능성이 높아진다. 따라서, 브라우저에서 제공하는 많은 종류의 API들을 대상으로 브라우저를 사용하는 유저의 프라이버시를 침해하는 위협으로써 영향력을 미칠 수 있는지 여부에 대한 이해가 필요하다.

그러나, 브라우저 핑거프린팅 관련 연구들은 이미 알려져 있는 벡터나 이전 연구에서 활용했던 벡터에 의존한다. 이전 연구들이 다루고 있는 브라우저 핑거프린팅 벡터들은 대부분 중복되며, 특정 웹 페이지가 브라우저 핑거프린팅 기능을 보유하고 있는지 여부를 판단하는 과정도 이미 알려져 있는 브라우저 핑거프린팅 벡터 리스트를 기반으로 수행되고 있다. 반면, 이미 알려져 있는 브라우저 핑거프린팅 이외의 다른 브라우저 핑거프린팅 벡터를 찾는 연구는 거의 수행되고 있지 않다.

본 논문에서는 브라우저 핑거프린팅 벡터로 활용될 수 있는 자바스크립트 API를 식별할 수 있는 도구를 제안한다. 구체적으로, 다양한 종류의 자바스크립트 API들을 테스트하기 위해 브라우저에 제공하는 입력이 잘 구성되어(well-structured) 있으면서도 별도의 유저의 도움 없이 브라우저 핑거프린팅에 활용될 수 있는 API를 식별하기 위한 작업이 이루어지는 도구를 제안한다.

문법 기반의 입력 생성과 랜덤 테스팅 기법을 활용하여 브라우저 입력을 생성하며, 차등 테스트를 통해 유저의 도움 없이 브라우저 핑거프린팅 벡터를 효율적으로 식별할 수 있는 도구를 구현하였다. 그 결과 총 75개의 브라우저 종류 또는 버전 정보를 유추할 수 있는 브라우저 핑거프린팅 벡터를 찾았다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 내용과 관련된 배경지식에 대해 소개한다. 3장에서는 본 논문에서 제안한 브라우저 핑거프린팅 식별 도구의 설계 및 구현에 대해 설명한다. 4장에서는 구현한 도구의 성능을 평가한다. 5장에서는 본 논문의 결론을 내린다.

제 2 장 배경

2.1 브라우저 핑거프린팅

브라우저 핑거프린팅(Browser Fingerprinting)이란 브라우저에서 제공하는 기능을 통해 얻을 수 있는, 유저를 식별하는 데 활용될 수 있는 일련의 정보를 수집하는 행위를 의미한다 [16]. 쿠키에 작성되어 있는 정적인 정보를 기반으로 하여 특정 상태를 갖는(stateful) 쿠키 기반의 웹 트래킹과는 달리, 브라우저 핑거프린팅을 통해 얻을 수 있는 정보는 정적인 형태가 아니기 때문에 특정 상태를 갖지는 않는다(stateless). 따라서, 유저를 식별하는 성능의 관점에서 쿠키 기반의 웹 트래킹보다 불안정할 수 있다. 그러나, 최근 주요 브라우저 벤더들을 중심으로 쿠키 기반의 웹 트래킹 행위로부터 유저의 프라이버시를 보호하기 위한 대책을 마련하고 있으며, 이에 대한 대응으로 브라우저 핑거프린팅을 통한 웹 트래킹 기술이 주목을 받고 있다,

브라우저 핑거프린팅은 정보를 얻을 수 있는 방법 또는 경로에 따라 수동적인 핑거프린팅(Passive Fingerprinting)과 적극적인 핑거프린팅(Active Fingerprinting)으로 구분된다. 수동적인 핑거프린팅은 웹 클라이언트와 웹 서버 사이에서 웹 컨텐츠를 주고받는 동작 과정에서 얻을 수 있는 정보들을 수집하는 행위를 가리킨다. 적극적인 핑거프린팅은 수동적인 핑거프린팅과 달리 웹 클라이언트의 실행 환경과 관련되어 있는 임의의 정보들을 수집하는 행위를 가리킨다. 적극적인 핑거프린팅을 통해 얻고자 하는 정보는 브라우저 종류, 브라우저 버전, 브라우저를 실행하고 있는 운영체제에 대한 정보, 또는 브라우저를 실행하고 있는 호스트 머신의 하드웨어와 관련된 특성 등이 있다.

브라우저 핑거프린팅이 유저를 식별하는 작업에 활용될 수 있는 이유는 유저마다 브라우저를 실행하고 있는 환경이 다르고, 서로 다른 실행

환경에서 비롯된 실행 결과의 차이 때문이다. 브라우저를 통해 동일한 웹 서버에 접근하여 브라우저 내부적으로 웹 컨텐츠 생성을 위한 작업을 수행할 때 그 결과가 항상 같지는 않을 수 있다. 크롬 또는 파이어폭스와 같이 브라우저의 종류에 따라 그 결과가 다를 수 있고, 브라우저의 버전에 따라 결과가 달라질 수 있다. 또한, 브라우저가 실행되고 있는 운영체제에 따라서도 달라질 수 있으며 [4], 심지어는 유저가 사용하고 있는 머신 또는 디바이스의 종류에 따라 달라질 수도 있다. 자바스크립트 언어를 위한 표준이 정립되고는 있으나 [20], 실행 환경의 차이에 따른 결과의 차이까지 고려하는 것은 현실적으로 어려운 것이 사실이다.

2.2 차등 테스트

차등 테스트(Differential Testing)는 소프트웨어 테스트에 활용되는 기법 중 하나로, 동일하거나 유사한 기능을 수행하는 여러 개의 소프트웨어에 동일한 입력을 제공한 뒤 실행 결과를 비교함으로써 소프트웨어의 버그를 찾는 기법을 가리킨다. 메모리 버그(Memory Corruption Bug)와 같이 문제 상황을 정의하거나 비정상적인 상태를 탐지하기 쉬운 버그들과 달리, 소프트웨어가 의도한대로 구현되어 있는지와 같이 직관적이지 않고 옳고 그름의 기준을 정의하는 것이 어려운 소프트웨어 버그가 존재한다. 대표적으로는 시맨틱(semantic) 버그가 있으며, 이러한 유형의 버그를 탐지하기 위해 차등 테스트가 활용된다. 차등 테스트를 활용한 연구로는 파일시스템을 대상으로 한 연구 [17]와 CPU 구현을 대상으로 한 연구 [18]가 있다.

차등 테스트는 브라우저에서 제공하는 자바스크립트 API 중에서 브라우저 핑거프린팅에 활용될 수 있는 API들을 식별하기 위한 방법으로 적합하다. 브라우저 핑거프린팅은 자바스크립트 API의 실행 결과의 차이에서 비롯되기 때문에 실행 결과로 얻어진 값 자체만으로는 브라우저 핑거프리팅 활용 여부를 판단하기 어렵다. 즉, 메모리 버그와 같이 비정상적인 상황을 명시적으로 정의하는 것이 어렵다. 반면, 차등 테스트를 통해

여러 브라우저에서 실행된 결과 값들을 서로 비교함으로써 브라우저 핑거프린팅으로 활용될 수 있는지 여부를 판단할 수 있다. 현재 브라우저 시장에 크롬, 사파리(Safari), 엣지(Edge), 파이어폭스 등 다양한 종류의 브라우저가 공급되어 있기 때문에 차등 테스트 수행을 위해 유사한 기능을 수행하는 여러 종류의 브라우저를 확보하는 것도 어렵지 않다.

제 3 장 설계

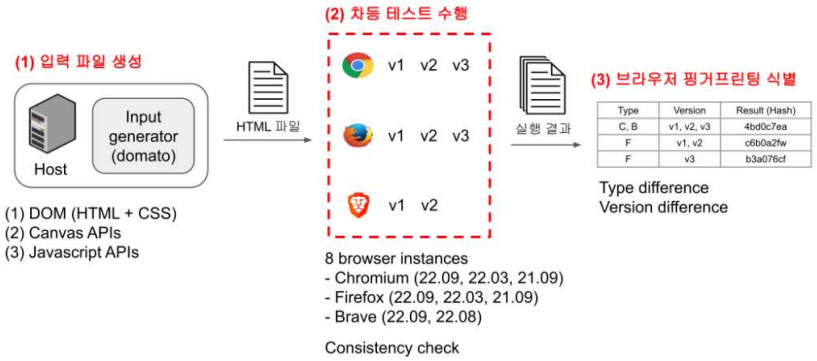
3.1 식별 도구 설계

그림 3.1은 본 논문에서 제안한 브라우저 핑거프린팅 식별 도구의 전체적인 디자인을 나타낸다. 브라우저 핑거프린팅 식별 도구는 입력 파일 생성 단계, 차등 테스트 수행 단계, 브라우저 핑거프린팅 식별 단계로 구성되어 있다.

3.1.1 입력 파일 생성

입력 파일 생성 단계에서는 브라우저에게 제공할 입력 파일을 생성한다. 입력 파일은 HTML 문법을 따르며, 미리 정의되어 있는 grammar 파일을 기반으로 생성된다. Grammar 파일은 테스트하고자 하는 자바스크립트 API에 대한 구체적인 명세를 포함하고 있다. 구체적인 명세란 API 함수 호출에 필요로 하는 파라미터 변수의 타입에 대한 정보를 의미한다. 예를 들어, Math.sin() 함수는 숫자 하나를 받아 해당 숫자의 사인값을 반환하며, 이 함수를 테스트하기 위해서는 파라미터로 하나의 Number 타입의 변수 또는 상수를 사용해야 한다. 이와 같이 grammar 파일은 API 함수의 올바른 실행을 위해 필요로 하는 파라미터에 대한 정보를 포함한다. 따라서, 생성되는 입력 파일은 문법적인 오류를 거의 포함하고 있지 않다. 이와 관련된 구체적인 정보는 자바스크립트 API에 대해 소개하거나 간단한 사용법을 설명한 문서와 자바스크립트 표준을 정의한 문서인 ECMAScript를 참고하였다 [19] [20].

생성되는 입력 파일은 테스트하고자 하는 API의 분류에 따라 HTML+CSS, Canvas, Javascript로 나뉘며, 각 분류에 따른 구체적인 내용은 다음과 같다.



Type	Version	Result (Hash)
C, B	v1, v2, v3	4bd0c7ea
F	v1, v2	c6b0a2fw
F	v3	b3a076cf

그림 3.1: 핑거프린팅 식별 도구 구현 디자인

HTML+CSS 파일은 웹 페이지의 구성과 관련된 요소들이 임의로 생성된 결과를 포함한다. 또한, grammar 파일에 작성되어 있는 자바스크립트 API 중 생성된 구성 요소와 연관된 API가 임의로 선택되어 문법에 맞게 작성된 구문(statement)들을 포함한다.

Canvas 파일은 고정된 하나의 <canvas> 원소(element)를 생성하며, 생성된 요소와 연관된 자바스크립트 API 중 일부가 임의로 선택되어 문법에 맞게 작성된 구문들을 포함한다.

Javascript 파일은 앞의 두 파일에서 포함하고 있지 않은 자바스크립트 API 중 일부가 임의로 선택되어 문법에 맞게 작성된 구문들을 포함한다. 단, 해당 파일이 브라우저에서 제공하는 모든 자바스크립트 API를 테스트할 수 있도록 구현되지는 않았다.

3.1.2 차등 테스트 수행

차등 테스트 수행 단계에서는 생성된 입력 파일을 여러 개의 브라우저에게 전달한 후 해당 파일에 있는 자바스크립트 API들을 실행한다. 현재 시간, 랜덤값 출력 등 실행할 때마다 결과값이 달라지는 경우를 제외하기 위해 동일한 입력에 대하여 차등 테스트를 두 번 수행한 뒤 그 결과가 같은지 여부를 확인하는 작업을 동반한다.

3.1.3 브라우저 핑거프린팅 식별

브라우저 핑거프린팅 식별 단계에서는 실행된 결과로부터 브라우저 핑거프린팅으로 활용될 수 있는 지 여부를 판단한다. 이를 위해 차등 테스트 수행 단계에서 브라우저 별로 실행된 결과들로부터 각각의 해시값을 계산한 뒤, 해시값을 기준으로 브라우저 종류와 버전을 분류한다. 두 개 이상의 해시값이 생성되었다면, 브라우저 종류 또는 버전에 따른 실행 결과에 차이가 발생되었다는 사실을 알 수 있다.

3.2 구현

본 논문에서 제안한 식별 도구는 운영체제 Ubuntu 20.04, 커널 버전 5.15.0-52-generic, Python 3.8.10 환경에서 구현하였다. 로컬 환경에 Apache 서버를 통해 생성한 입력 파일들을 브라우저에게 전달하였으며, 사용한 Apache 버전은 2.4.41이다.

입력 파일 생성을 위해 open-source html generator인 domato를 활용하였다 [21]. Domato는 Google Project Zero에서 제공하고 있는 오픈소스로써, 미리 정의되어 있는 grammar 파일을 기반으로 HTML 파일을 생성하는 도구이다. Domato는 canvas, jscript, mathml3, php, vbscript, webgl 등 많은 종류의 grammar 파일을 제공하고 있으나 누락되거나 더 이상 지원되지 않는 API들을 포함하고 있으며, 특정 타입의 변수를 정의할 때 미리 정해놓은 상수 값들만 활용하며, 제한된 몇 가지 항목에 대해서만 임의의 값을 생성하고 있다. 따라서 API 설명 문서를 참고하거나 [22] [23] [24] [25], 미리 정해놓은 특정 상수 값 이외에도 임의의 값을 생성할 수 있도록 기존 grammar 파일을 개선하였다.

차등 테스트 수행을 위해 크로미움(Chromium), 파이어폭스(Firefox), 브레이브(Brave) 브라우저를 대상으로 하였다. 브라우저 별 버전 정보는 크로미움의 경우 105.0.5195.125, 99.0.4844.51, 94.0.4606.61 버전을 활용하였으며, 파이어폭스는 105.0, 98.0, 92.0 버전을 활용하였고, 브레이브 브

라우저는 v1.43.91, v1.42.97 버전을 활용하였다. 원활한 테스트 수행을 위해 셀레니움(Selnium with Python)을 활용하였다 [26].

브라우저 핑거프린팅 식별을 위한 작업은 파이썬(Python)으로 구현하였으며, 핑거프린팅 식별을 위해 구현한 모듈의 코드 길이는 약 600라인이다.

제 4 장 평가

4.1 평가 방법

본 논문에서 제안한 브라우저 핑거프린팅 식별 도구의 성능을 평가하기 위해, 브라우저 핑거프린팅 관련 프로젝트 및 이전 연구에서 활용한 브라우저 핑거프린팅 벡터 리스트와 식별 도구를 통해 찾은 브라우저 핑거프린팅 벡터를 비교하고자 한다. 먼저 관련 프로젝트 및 이전 연구에서 활용했던 벡터들을 설명한 뒤, 구현한 식별 도구를 통해 찾은 벡터를 설명한다.

4.2 결과

4.2.1 이전 연구 평가

표 4.1은 공개되어 있는 브라우저 핑거프린팅 관련 프로젝트에서 유저 식별을 위해 활용하고 있는 브라우저 핑거프린팅 벡터 리스트를 정리한 결과이다. 공통적으로 활용하고 있는 벡터로는 Canvas API, WebGL API, WebAudio API, navigator 오브젝트의 프로퍼티, screen 오브젝트의 프로퍼티 등이 있다.

표 4.2는 이전 연구들에서 활용한 브라우저 핑거프린팅 벡터 리스트를 정리한 표이다. 대부분의 연구들에서 공개되어 있는 브라우저 핑거프린팅 프로젝트를 참고하였으며, 연구에 활용한 브라우저 핑거프린팅 벡터 리스트 또한 대부분 중복되어 있다는 사실을 확인할 수 있다.

API 또는 Property	구분	[27]	[28]	[29]	[30]	[31]
Pixel data (2D-image)	Canvas	○		○	○	○
Pixel data (3D-image)	WebGL	○	○			○
WebGL vendor/renderer	WebGL	○	○	○		○
WebAudio API	WebAudio	○	○		○	○
plugins	navigator	○	○	○	○	○
timezone	Date	○	○	○	○	○
screen width/height	screen	○	○	○	○	○
screen colorDepth	screen	○	○	○	○	○
fonts	Javascript	○	○	○	○	○
cookieEnabled	Javascript	○	○	○	○	○
pdfViewerEnabled	navigator					○
(local/session)Storage	window	○	○	○	○	
openDatabase	window	○			○	
indexedDB	window	○	○	○	○	
doNotTrack	navigator	○	○	○		○
language(s)	navigator	○	○	○	○	○
platform	navigator	○	○	○	○	○
maxTouchPoints	navigator	○		○	○	○
ontouchstart	Javascript	○			○	
Adblock	Javascript	○	○	○	○	
cpuClass	navigator	○		○	○	
hardwareConcurrency	navigator	○	○	○	○	○
deviceMemory	navigator	○	○		○	○
userAgent	navigator		○	○		○
buildID	navigator		○			○
product/productSub	navigator		○			○
vendor/vendorSub	navigator		○		○	○
javaEnabled	navigator		○			
avail(Top/Left)	screen		○		○	○
avail(Height/Width)	screen		○	○	○	○
permissions	Javascript		○			
devicePixelRatio	window			○		
Math	Javascript			○	○	
colorGamut	matchMedia				○	
invertedColors	matchMedia				○	
forcedColors	matchMedia				○	
monochrome	matchMedia				○	
contrast	matchMedia				○	
reducedMotion	matchMedia				○	
hdr	matchMedia				○	

표 4.1: 핑거프린팅 벡터 (공개 프로젝트)

구분	[6]	[7]	[8]	[9]	[10]	[11]	[12]
Canvas (pixel data)		○	○	○	○	○	○
WebGL (pixel data)			○		○		○
WebAudio				○			○
WebRTC				○			○
Error						○	
navigator	○			○	○	○	
screen	○			○	○	○	
의존성	[27]	[27]	[28]		[27] [28]	[7] [30]	[30]

표 4.2: 핑거프린팅 벡터 (이전 연구)

4.2.2 식별 도구 평가

본 논문에서 제안한 브라우저 핑거프린팅 식별 도구를 통해 총 75개의 브라우저 핑거프린팅 벡터를 식별하였다. 구체적으로, HTML+CSS 항목에서 54개, Canvas 항목에서 11개, Javascript 항목에서 10개를 식별하였다.

표 4.3, 표 4.4는 HTML+CSS 파일에서 찾은 브라우저 핑거프린팅 벡터를 정리한 표이다. 첫 번째 열은 API 또는 프로퍼티의 구체적인 명칭을 의미하며, 두 번째 열은 API 또는 프로퍼티와 관련있는 오브젝트를 가리킨다. 세 번째 열은 해당 브라우저 핑거프린팅 벡터를 통해 얻을 수 있는 정보를 의미하며, C는 크로미움(브레이브 브라우저), F는 파이어폭스를 의미한다. 찾은 54개의 벡터 중 브라우저 종류와 관련된 벡터는 51개, 브라우저 버전과 관련된 벡터는 3개이다.

명칭	구분	핑거프린팅
accessKeyLabel	HTMLElement	종류(C, F)
caretRangeFromPoint	document	종류(C, F)
cloneRange	Range	종류(C, F)
startContainer	Range	종류(C, F)
createNSResolver	document	종류(C, F)
dirName	HTMLTextAreaElement	종류(C, F)
createEvent	document	종류(C, F)
xmlVersion	document	종류(C, F)
getCTM	SVGGraphicsElement	종류(C, F)
inert	Element	종류(C, F)
loading	HTMLImageElement	종류(C, F)
mediaKeys	HTMLMediaElement	종류(C, F)
mozDecodedFrames	HTMLVideoElement	종류(C, F)
mozPresentedFrames	HTMLVideoElement	종류(C, F)
mozFragmentEnd	HTMLMediaElement	종류(C, F)
mozHasAudio	HTMLVideoElement	종류(C, F)
onbeforepaste	Element	종류(C, F)
onwebkitfullscreenchange	Element	종류(C, F)
onwebkitfullscreenerror	Element	종류(C, F)
onbeforecut	Element	종류(C, F)
onbeforecopy	Element	종류(C, F)
onbegin	Element	버전(F)
oncancel	Element	종류(C, F)
onsearch	Element	종류(C, F)
onrepeat	Element	버전(F)
preload	HTMLVideoElement	종류(C, F)
spellcheck	Element	종류(C, F)
translate	HTMLVideoElement	종류(C, F)
webkitFullscreenElement	document	종류(C, F)
webkitVideoDecodedByteCount	HTMLMediaElement	종류(C, F)
webkitDecodedFrameCount	HTMLVideoElement	종류(C, F)

표 4.3: 결과 (HTML+CSS (1))

명칭	구분	핑거프린팅
webkitHidden	document	종류(C, F)
styleMedia	window	종류(C, F)
visualViewport	window	버전(F)
WebKitMutationObserver	window	종류(C, F)
webkitExitFullScreen	HTMLVideoElement	종류(C, F)
execCommand	document	종류(C, F)
defaultstatus	window	종류(C, F)
getCharNumAtPosition	SVGTextContentElement	종류(C, F)
ping	HTMLAnchorElement	종류(C, F)
checkIntersection	SVGSVGElement	종류(C, F)
compareDocumentPosition	Node	종류(C, F)
isPointInStroke	SVGGeometryElement	종류(C, F)
onloadedmetadata	Element	종류(C, F)
style	HTMLUnknownElement	종류(C, F)
onpointermove	Element	종류(C, F)
webkitRequestAnimationFrame	window	종류(C, F)
extentNode	Selection	종류(C, F)
getIntersectionList	SVGSVGElement	종류(C, F)
offscreenBuffering	window	종류(C, F)
item	CSS	종류(C, F)
evaluate	XPathExpression	종류(C, F)
contentEditable	Global attr.	종류(C, F)
transformToDocument	XSLTProcesor	종류(C, F)

표 4.4: 결과 (HTML+CSS (2))

명칭	구분	핑거프린팅
captureStream	HTMLMediaElement	종류(C, F)
createConicGradient	CanvasRenderingContext2D	종류(C, F)
direction	CanvasRenderingContext2D	종류(C, F)
fontKerning	CanvasRenderingContext2D	버전(C) 버전(F)
fontVariantCaps	CanvasRenderingContext2D	종류(C, F) 버전(C)
fontStretch	CanvasRenderingContext2D	종류(C, F) 버전(C)
getTransform	CanvasRenderingContext2D	종류(C, F)
imageSmoothingQuality	CanvasRenderingContext2D	종류(C, F)
letterSpacing	CanvasRenderingContext2D	종류(C, F) 버전(C)
textRendering	CanvasRenderingContext2D	종류(C, F) 버전(C)
wordSpacing	CanvasRenderingContext2D	종류(C, F) 버전(C)

표 4.5: 결과 (Canvas)

표 4.5는 Canvas 파일에서 찾은 브라우저 핑거프린팅 벡터를 정리한 표이다. 찾은 11개의 벡터 중 브라우저 종류와 관련된 벡터는 10개, 브라우저 버전과 관련된 벡터는 1개이다. 기존 연구들에서 활용한 Canvas API에는 toDataURL(), getImageData()와 같이 픽셀 데이터 값과 관련있는 API가 있으며, 이미 잘 알려져 있는 핑거프린팅 벡터들이다. 실험 결과, 이미 잘 알려져 있는 API 이외에도 브라우저 핑거프린팅으로 활용 가능한 API가 있다는 것을 확인할 수 있다.

명칭	구분	핑거프린팅
sin	Math	종류(C, F)
tan	Math	종류(C, F)
hypot	Math	종류(C, F)
log10	Math	종류(C, F)
atan2	Math	종류(C, F)
lastMatch	RegExp	종류(C, F)
test	RegExp	종류(C, F)
exec	RegExp	종류(C, F)
sort.call	Array.prototype	종류(C, F)
Number	Number	종류(C, F)

표 4.6: 결과 (Javascript)

표 4.6은 앞의 API 분류와 중복되지 않는 Javascript API를 포함한 파일에서 찾은 브라우저 핑거프린팅 벡터를 정리한 표이다. 10개의 벡터 모두 브라우저 종류와 관련된 벡터이다.

Math API는 일부 프로젝트에서 핑거프린팅 벡터로 활용하고 있다. 그럼에도 불구하고 실험 결과에 Math API를 포함시킨 이유는, 기존 프로젝트에서 활용한 Math API는 파라미터에 고정된 상수만을 활용했기 때문이다. 실행 결과 값이 하나만 다르더라도 그 결과를 기반으로 핑거프린팅 벡터로 활용할 수 있기 때문에, 동일한 API에 대해서도 가능한 많은 파라미터를 대상으로 테스트를 진행하는 과정이 필요하다. 그러나, Math API를 포함하고 있던 프로젝트들에서는 이 부분을 고려하지 않았다. 예를 들어, 실험을 진행한 세 브라우저 대상으로 Math.sin(-1e300)를 계산한 결과는 0.8178819121159085로 동일하지만, Math.sin(Math.LN10)를 계산한 결과는 크로미움과 브레이브 브라우저는 0.7439803369574931, 파이어폭스는 0.743980336957493로 결과가 다르다. 따라서, Math.sin API는 브라우저 종류를 식별하는 데 활용할 수 있다.

4.2.3 유저 식별 성능 평가

브라우저 핑거프린팅의 유저 식별 성능을 평가하기 위해 섀넌의 정보 엔트로피를 활용하였다 [32].

$$H(X) = -\sum_{i} p_i \log_2 p_i$$

정보 엔트로피는 발생 가능한 사건들의 정보량의 기댓값으로 전체 사건의 확률분포의 불확실성을 나타내며, 위 수식과 같이 계산된다. 식에서 H(X)는 엔트로피, X는 확률변수, p는 사건이 일어날 확률을 의미한다. 예를 들어, 동전을 던졌을 때 앞면과 뒷면이 나올 확률이 각각 0.5라고 가정할 경우 엔트로피는 1이다.

일반적으로 브라우저 핑거프린팅의 유저 식별 성능을 평가하기 위해 브라우저 핑거프린팅 데이터셋을 활용한다. 핑거프린팅 데이터셋 수집은 브라우저 핑거프린팅 벡터를 실행시키고 그 결과를 수집할 수 있는 웹사이트를 구축한 뒤 실험 참여자들을 모집하는 방식으로 이루어진다. 본 논문에서는 참여자들을 모집하는 대신 여러 종류의 기기들로부터 브라우저 핑거프린팅 데이터셋을 수집하였다. 평가에 사용한 기기는 데스크탑 1대, 노트북 2대, 모바일 1대, 태블릿 1대 총 5대이며, 구체적인 사양은 표 4.7과 같다.

분류	CPU	RAM	운영체제
데스크탑	Intel i7-8700	32(GB)	Ubuntu 20.04
노트북 1	Intel i5-5200U	8(GB)	Windows 10 Home
노트북 2	i5-10210U	16(GB)	Windows 10 Home
모바일	Qualcomm SM8450	8(GB)	Android 12
태블릿	Qualcomm SM8250-AB	8(GB)	Android 12

표 4.7: 평가에 사용한 기기들의 사양

본 논문에서 찾은 핑거프린팅 벡터들의 유저 식별 성능을 평가하기 위해 표 4.1에 명시된 브라우저 핑거프린팅 벡터들의 엔트로피와 비교하였다.

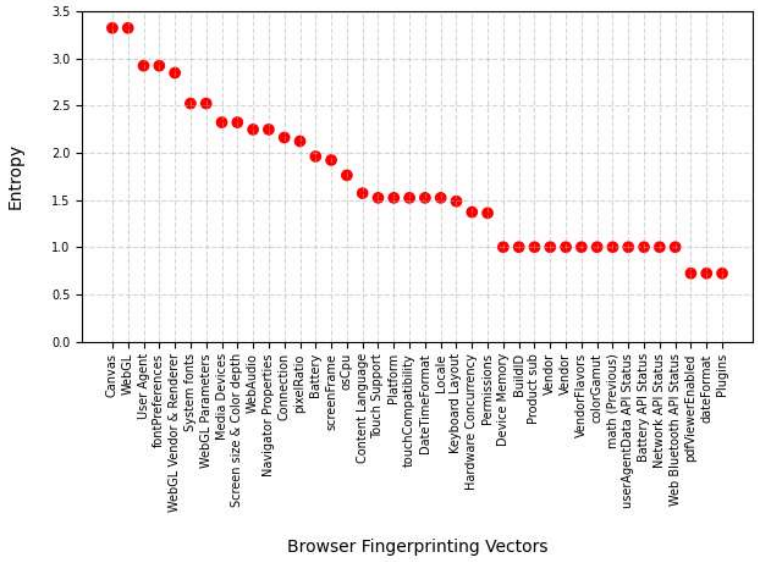


그림 4.1: 기존 핑거프린팅 벡터의 엔트로피

그림 4.1은 실험에 사용한 다섯 종류의 기기를 대상으로 표 4.1에 명시되어 있는 브라우저 핑거프린팅 벡터들의 엔트로피를 계산한 결과이다. 각 기기 별 최신 버전의 크롬과 파이어폭스를 통해 얻어진 값을 기반으로 엔트로피를 계산하였으며, 모든 기기들에 대하여 동일한 결과가 나온 벡터들은 제외하였다. 엔트로피 계산 결과 Canvas와 WebGL API는 모든 기기를 유일하게 식별할 수 있었던 반면, 브라우저 종류에 대한 정보만을 식별할 수 있는 벡터들도 다수 포함되어 있었다.

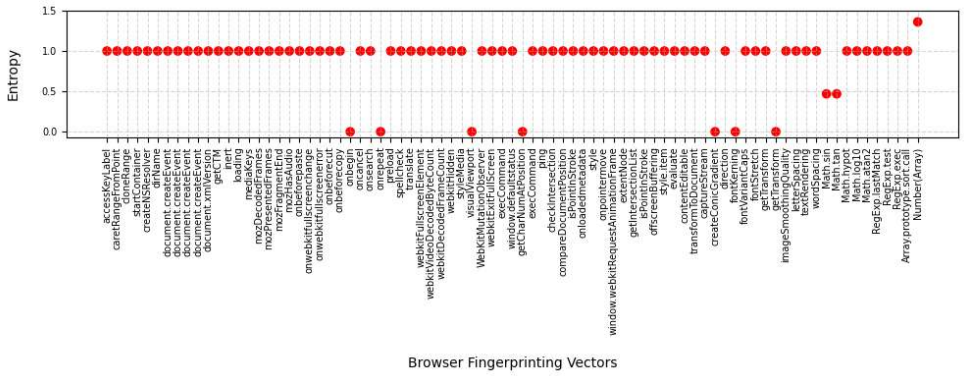


그림 4.2: 새로 찾은 핑거프린팅 벡터의 엔트로피

그림 4.2는 본 논문에서 제안한 브라우저 핑거프린팅 식별 도구를 통해 찾은 브라우저 핑거프린팅 벡터들의 엔트로피를 계산한 결과이다. 대부분 브라우저 종류에 대한 정보를 얻을 수 있는 핑거프린팅 벡터들인 것을 확인할 수 있으며, 브라우저 버전에 대한 정보를 얻을 수 있는 핑거프린팅 벡터의 경우에는 기기들을 구별하지 못하는 경우도 있었다. 이는 핑거프린팅 수집 시 활용한 기기들의 브라우저를 최신 버전으로 업데이트한 상태에서 핑거프린팅을 수집하였기 때문이다.

제 5 장 결론

쿠키 기반의 웹 트래킹에 대한 보호 대책이 논의됨에 따라 이를 대체하기 위한 방법으로 브라우저 핑거프린팅을 통한 웹 트래킹 기법이 주목을 받고 있다. 이에 따라 브라우저 핑거프린팅의 유저 식별 도구로써의 성능을 평가하는 많은 연구들이 수행되었다. 그러나 기존 연구들에서는 Canvas API, WebGL API, navigator 오브젝트의 프로퍼티, screen 오브젝트의 프로퍼티 등 이미 잘 알려져 있는 핑거프린팅 벡터만을 활용하였다. 이에 본 논문에서는 브라우저에서 제공하는 자바스크립트 API를 대상으로 브라우저 핑거프린팅 벡터로 활용될 수 있는 지 여부를 효율적으로 식별할 수 있는 도구를 구현하였다. 브라우저 핑거프린팅 식별 도구를 통해 총 75개의 클라이언트의 브라우저 종류 또는 버전에 대한 정보를 얻을 수 있는 브라우저 핑거프린팅 벡터를 찾았다.

참고문헌

[1] Anthony Chavez. "Expanding testing for the Privacy Sandbox for the Web". https://blog.google/products/chrome/update-testing-privacy-sandbox-web/ (visited on November 22, 2022).

[2] Marissa Wood. "Today's Firefox Blocks Third-Party Tracking Cookies and Cryptomining by Default". https://blog.mozilla.org/blog/2019/09/03/todays-firefox-blocks-third-party-tracking-cookies-and-cryptomining-by-default/ (visited on November 22, 2022).

[3] Eckersley, Peter. "How unique is your web browser?." *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, Berlin, Heidelberg, 2010.

[4] Mowery, Keaton, and Hovav Shacham. "Pixel perfect: Fingerprinting canvas in HTML5." *Proceedings of W2SP* 2012 (2012).

[5] Cao, Yinzhi, Song Li, and Erik Wijmans. "(Cross-) Browser Fingerprinting via OS and Hardware Level Features." *NDSS*. 2017.

[6] Nikiforakis, Nick, et al. "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting." *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013.

[7] Acar, Gunes, et al. "The web never forgets: Persistent tracking mechanisms in the wild." *Proceedings of the 2014 ACM SIGSAC Conf*

erence on Computer and Communications Security. 2014.

[8] Laperdrix, Pierre, Walter Rudametkin, and Benoit Baudry. "Beauty and the beast: Diverting modern web browsers to build unique brows er fingerprints." 2016 IEEE Symposium on Security and Privacy (SP). IEEE, 2016.

[9] Englehardt, Steven, and Arvind Narayanan. "Online tracking: A 1-million-site measurement and analysis." *Proceedings of the 2016 AC M SIGSAC conference on computer and communications security*. 20 16.

[10] Gómez-Boix, Alejandro, Pierre Laperdrix, and Benoit Baudry. "Hi ding in the crowd: an analysis of the effectiveness of browser fingerp rinting at large scale." *Proceedings of the 2018 world wide web confe rence*. 2018.

[11] Vastel, Antoine, et al. "{Fp-Scanner}: The Privacy Implications o f Browser Fingerprint Inconsistencies." 27th USENIX Security Sympo sium (USENIX Security 18). 2018.

[12] Iqbal, Umar, Steven Englehardt, and Zubair Shafiq. "Fingerprintin g the fingerprinters: Learning to detect browser fingerprinting behavio rs." *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 202 1.

[13] Wu, Shujiang, et al. "Rendered Private: Making {GLSL} Executio n Uniform to Prevent {WebGL-based} Browser Fingerprinting." *28th USENIX Security Symposium (USENIX Security 19)*. 2019.

[14] Trickel, Erik, et al. "Everyone is different: client-side diversificati on for defending against extension fingerprinting." *28th USENIX Sec urity Symposium (USENIX Security 19)*. 2019.

[15] Pierre Laperdrix. "Browser Fingerprinting: An Introduction and th e Challenges Ahead". https://blog.torproject.org/browser-fingerprinting -introduction-and-challenges-ahead/ (visited on November 22, 2022).

[16] W3C. "Mitigating Browser Fingerprinting in Web Specifications". https://w3c.github.io/fingerprinting-guidance/ (visited on November 21, 2022).

[17] Min, Changwoo, et al. "Cross-checking semantic correctness: The case of finding file system bugs." Proceedings of the 25th Symposium on Operating Systems Principles. 2015.

[18] Hur, Jaewon, et al. "Difuzzrtl: Differential fuzz testing to find cpu bugs." 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 20 21.

[19] Mozila. "Web APIs". https://developer.mozilla.org/en-US/docs/We b/API (visited on November 29, 2022).

[20] ECMA International. "ECMA-262". https://www.ecma-internationa l.org/publications-and-standards/standards/ecma-262/ (visited on Nove mber 14, 2022).

[21] Ivan Fratric. Domato: A DOM Fuzzer. https://github.com/googlepr

ojectzero/domato (visited on November 21, 2022).

[22] WHATWG. "HTML Standard". https://html.spec.whatwg.org/multipage/ (visited on November 4, 2022).

[23] Mozila. "HTMLCanvasElement". https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement (visited on October 29, 2022).

[24] Mozila. "CanvasRenderingContext2D". https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D (visited on October 29, 2022).

[25] Mozila. "JavaScript". https://developer.mozilla.org/en-US/docs/Web/JavaScript (visited on October 31, 2022).

[26] Baiju Muthukadan. "Selenium with Python". https://selenium-python.readthedocs.io/ (visited on November 21, 2022).

[27] Electoric Frontier Foundation. "Cover Your Tracks". https://coveryourtracks.eff.org/ (visited on November 22, 2022).

[28] "AmIUnique". https://amiunique.org/ (visited on November 22, 2022).

[29] Peter Hraska. "Browser Fingerprinting". http://fp.virpo.sk/ (visited on November 28, 2022).

[30] "FingerprintJS". https://github.com/fingerprintjs/fingerprintjs (visited on November 20, 2022).

[31] "BrowserLeaks". https://browserleaks.com/ (visited on October 31, 2022).

[32] Shannon, Claude Elwood. "A Mathematical theory of communicati on." The Bell system technical journal 27.3 (1948): 379-423

Abstract

Finding Browser Fingerprinting Vector Using Differential Testing

Jinhan Kwak
Department of Electrical and Computer Engineering
The Graduate School
Seoul National University

Browser fingerprinting refers to the act of collecting and managing a series of information to identify web clients by utilizing functionaliti es provided by browsers. A browser supports many kinds of APIs th rough javascript engine for executing javascript code and rendering e ngine for displaying web contents to clients. Because the browser fin gerprinting can be performed by executing these APIs, we have to an alyze APIs' functionalities provided by browser in terms of a user pri vacy. In this paper, we proposed a tool that can identify Javascript A PIs that can be used as browser fingerprinting vectors. It can test Ja vascript APIs by generating syntactically correct HTML inputs. Also, it can identify browser fingerprinting automatically by performing diff erential testing. As a result, we found 75 browser fingerprinting vecto rs that can infer browser type or version-related information.

Keywords: Browser Fingerprinting, Differential Testing
Student Number: 2020-25377