SoK:
On the Analysis of Web Browser Security

스마트보안전공 202334843 이동하

SoK 논문이란?

SoK(State of the Knowledge) 논문은 특정 연구 분야나 주제에 대한 현재까지의 연구 동향, 기여를 포괄적으로 조사, 정리한 논문이라고 한다.

Web Browser background

현대에 필수적인 요소
은행 일, 의료 기록 조회 등 매우 민감한 작업이 일어나는 곳

Web Browser background

현대에 필수적인 요소
은행 일, 의료 기록 조회 등 매우 민감한 작업이 일어나는 곳

<보안 관점>

코드 스케일이 매우 크다(크롬:2500만 라인) & 복잡하다

remote 환경을 직접적으로 액세스하여 매력적인 공격 포인트

매우 빠르게 변화한다

Web Browser background

논문 목표 :

- 4대 major browser를 분석하고 보안 설계를 정리한다
- 10년간 발견된 보안 취약점을 정리하여 트랜드를 분석한다
- 웹 브라우저에 도입된 mitigation을 분석한다
- full-chain exploit을 분석한다

Web Browser background

2021년 조사 결과 0-day 공격의 50%가 browser를 target으로 삼는다는 결과가 제시되었다.

공격자 : browser를 대상으로하는 exploit을 연구하고 있다
브라우저 벤더 : 새로운 mitigation을 구현, 도입하고 있다

bug bounty, open source, fuzzing 등을 통해 버그를 악용 이전에 찾으려는 노력이 진행되고 있다

Web Browser background

web browser 연구는 매우 빠르게 변화하고, 스케일이 크다는 이유로 체계화 되어있지 않다

본 연구를 통해 4대 web browser를 하나의 아키텍처로 모델링하고 보안 설계를 비교하겠다

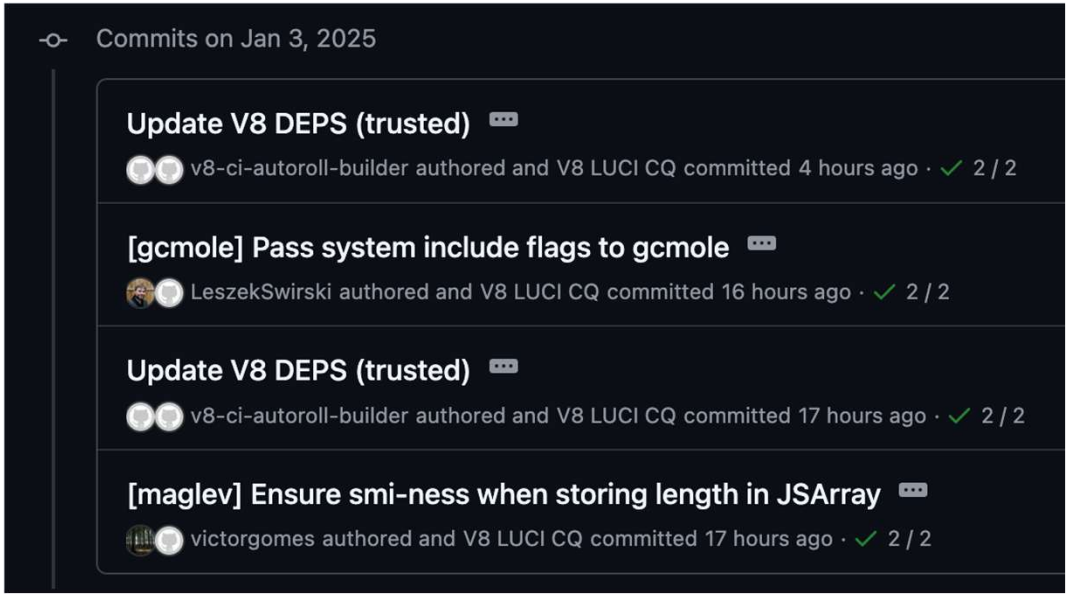
Web Browser 분석 아키텍처 challenge

A Moving Target

브라우저는 일반적인 업데이트 주기보다 빠른 업데이트 주기를 가진다

weekly update가 있을 정도

따라서 근본적인 설계에 집중한다



Web Browser 분석 아키텍처 challenge

Overwhelming size

source code 규모가 매우 크다

0-day, vulnerability, mitigation 정보가 분산되어 있다

->Chrome, Firefox, Safari, Edge를 4대 브라우저로 정의

->이슈트래커, 벤더사의 공개 자료, NVD 등을 바탕으로 정보 수집

Web Browser 분석 아키텍처 challenge

Unique design

브라우저마다 요구사항, 제약사항이 다르다

따라서 모든 브라우저를 설명하기 위한 아키텍처를 구현하는 동시에,

세부 내용이 누락되지 않도록 해야한다

Web Browser 분석 아키텍처 challenge

server의 문제는 본 논문에서 다루지 않는다

ex) XSS, SQL Injection

그러나 UXSS는 브라우저의 문제이므로 논문에서 다룬다

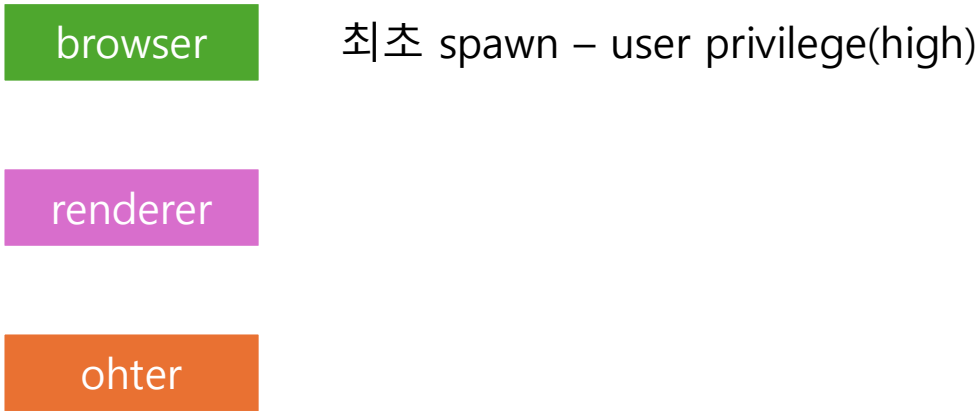
Web Browser 분석 아키텍처 challenge

브라우저의 프로세스는 Principle of Least Privilege를 기반하여 권한이 정의된다.
권한의 정의는 운영체제의 기능을 이용한다.

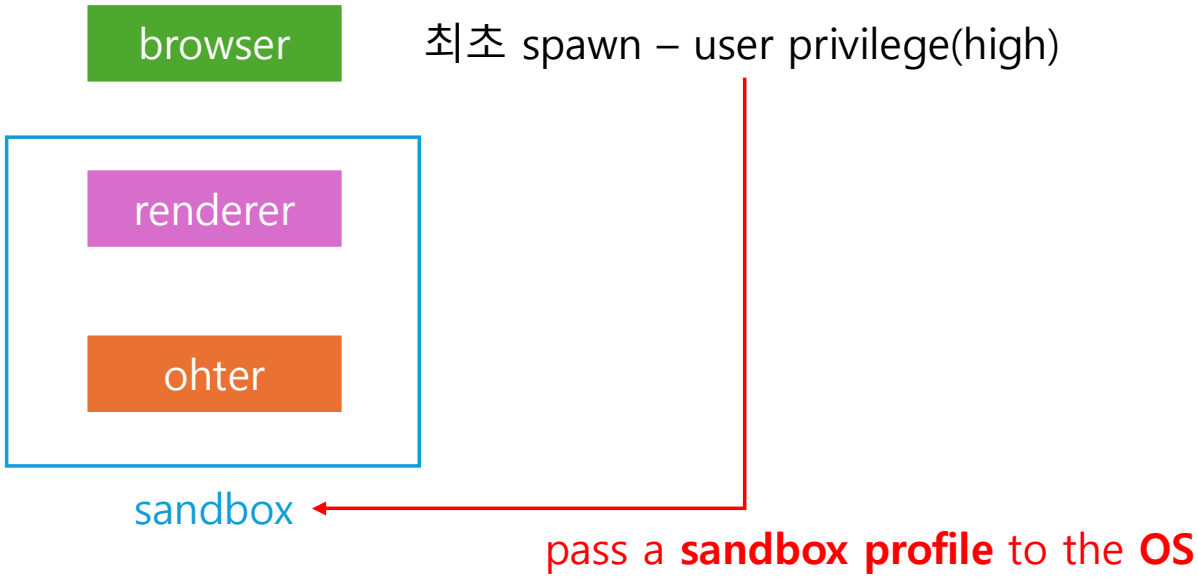
Web Browser 분석 아키텍처 challenge



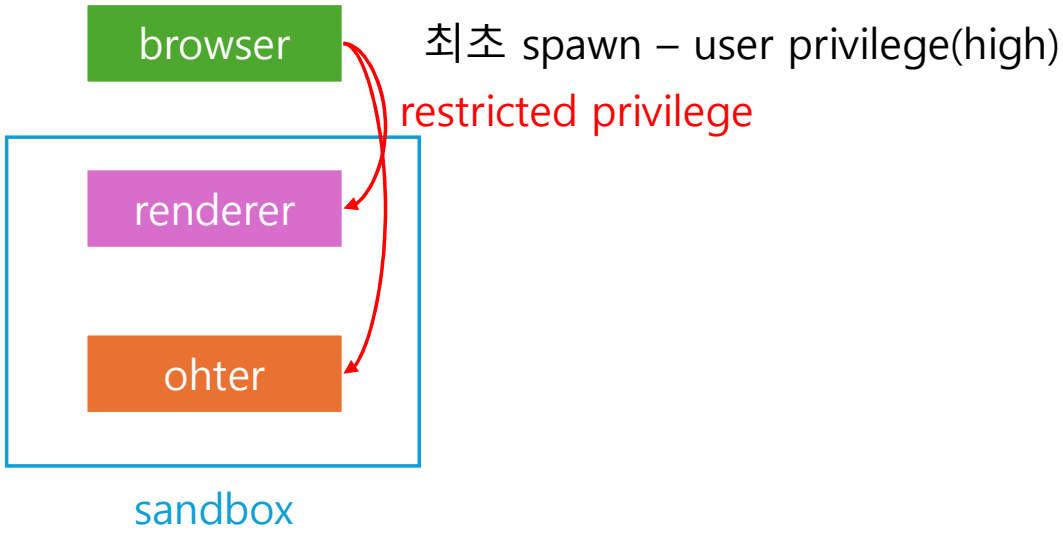
Web Browser 분석 아키텍처 challenge



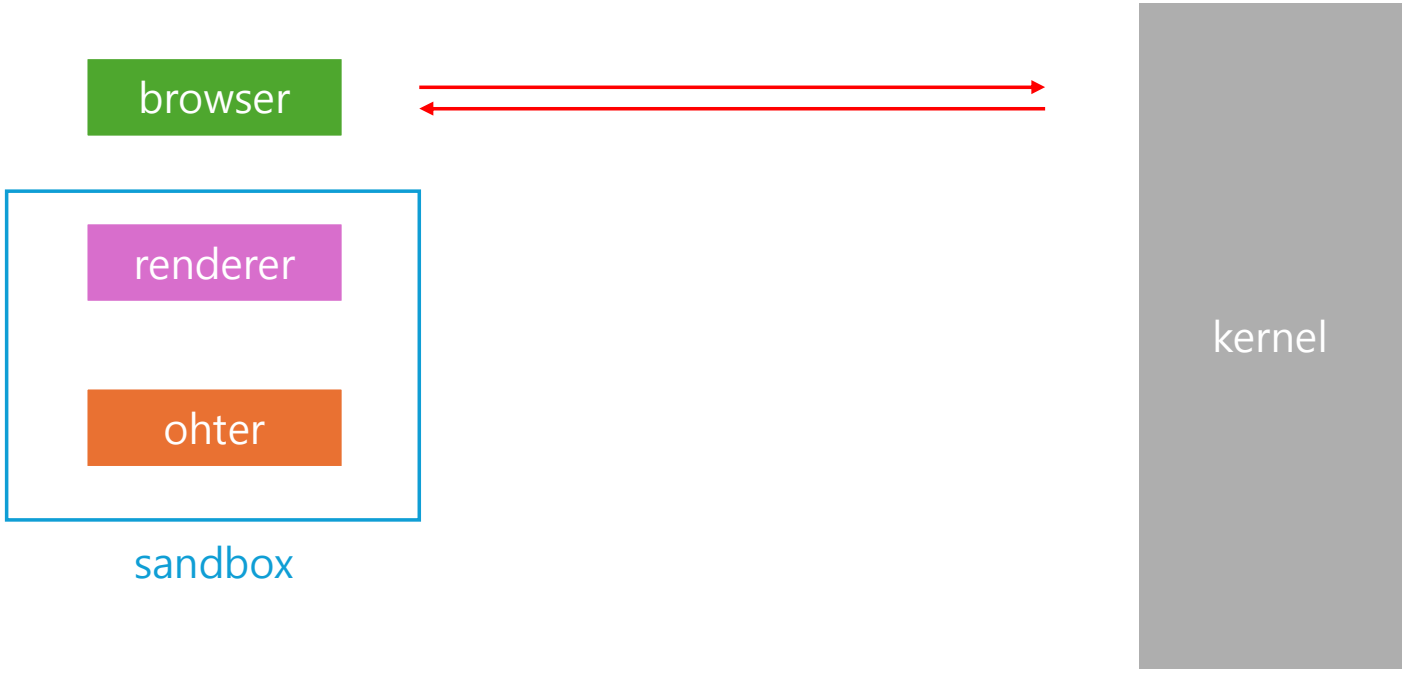
Web Browser 분석 아키텍처 challenge



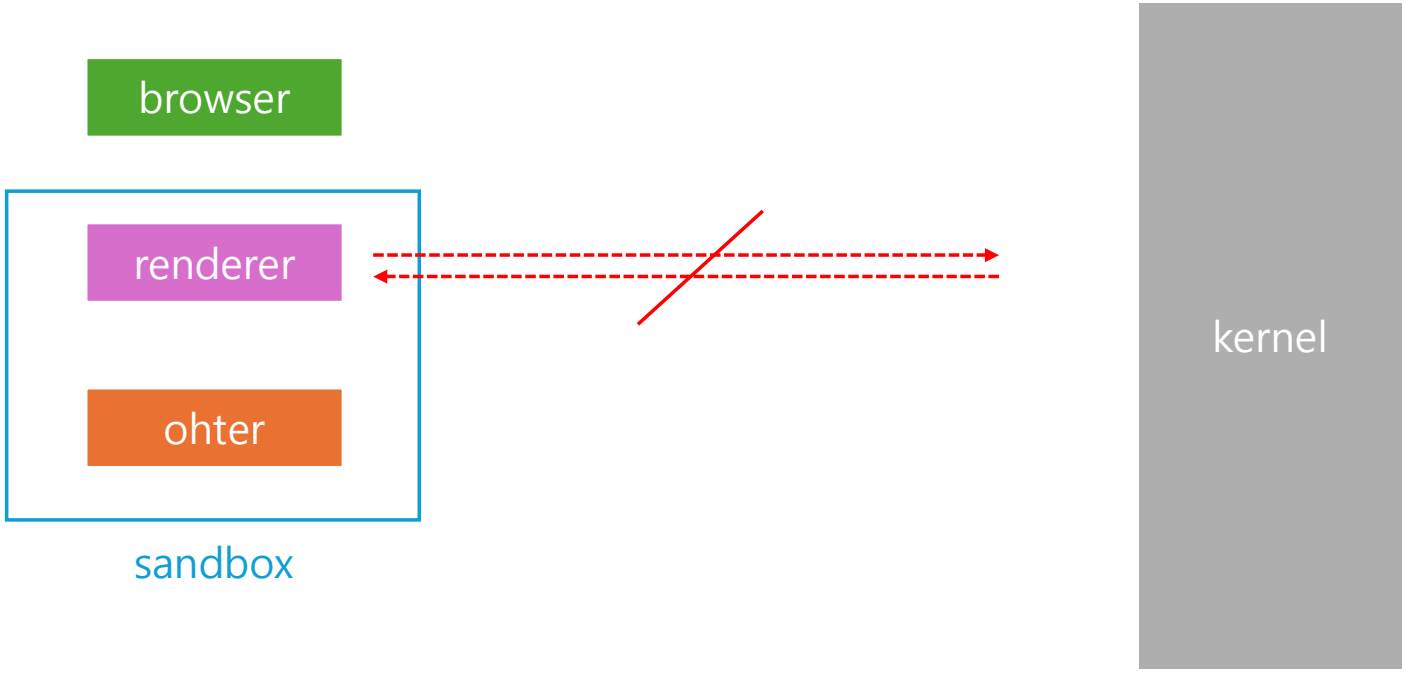
Web Browser 분석 아키텍처 challenge



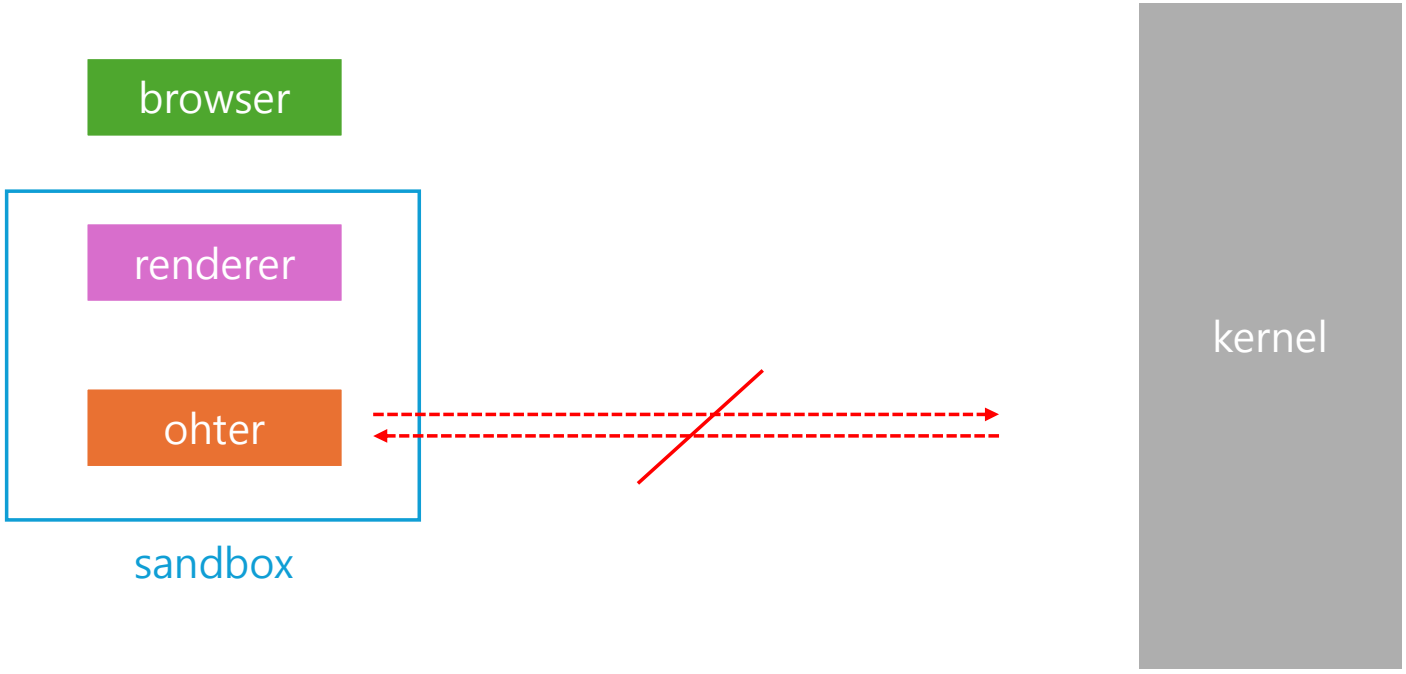
Web Browser 분석 아키텍처 challenge



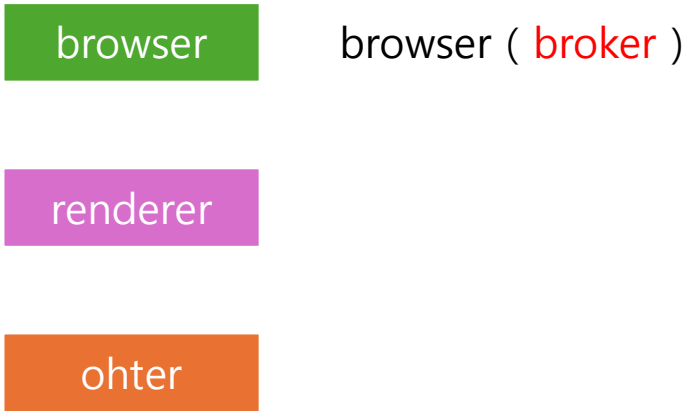
Web Browser 분석 아키텍처 challenge



Web Browser 분석 아키텍처 challenge

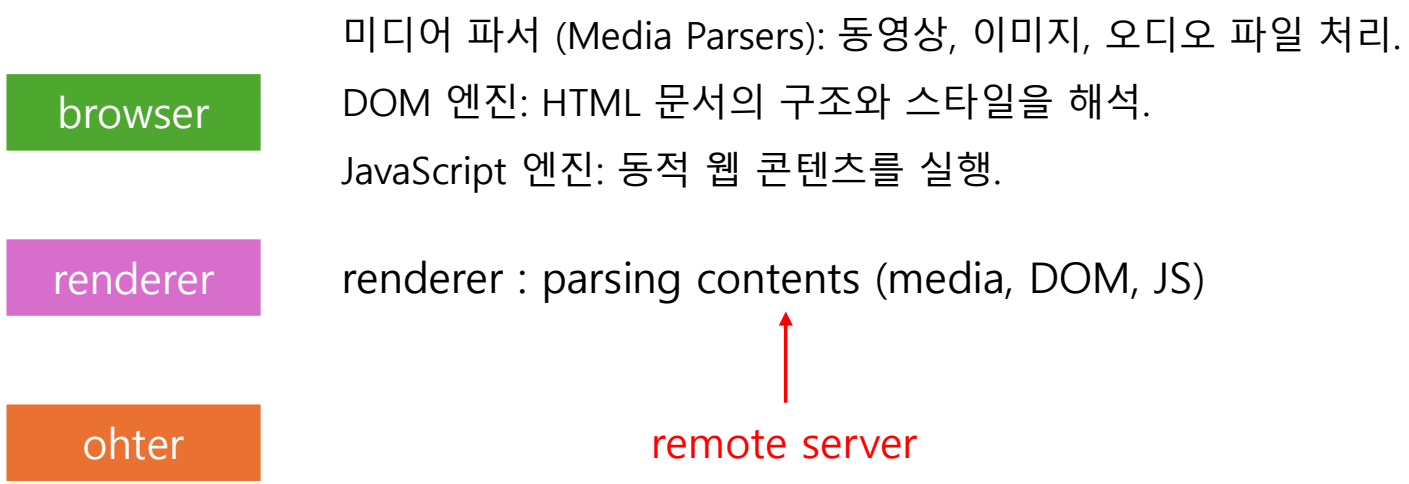


Web Browser 분석 아키텍처 challenge

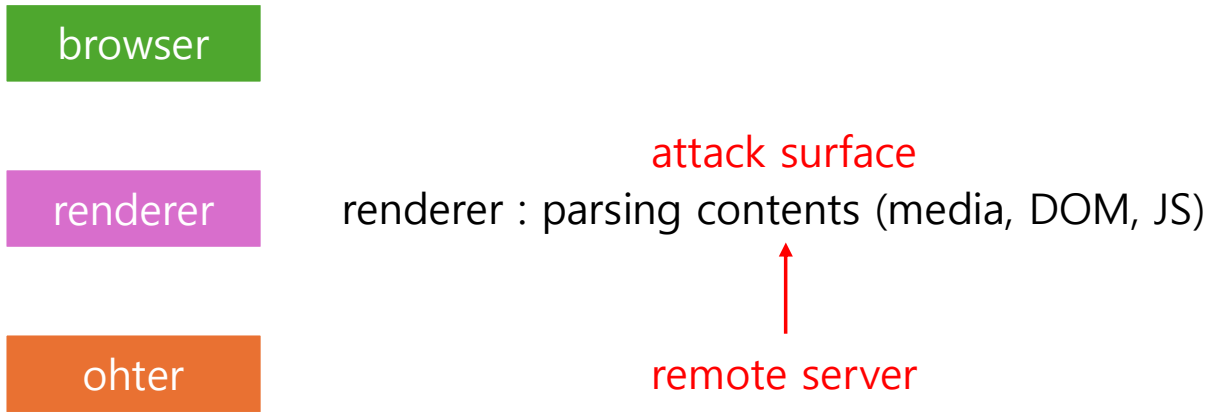


browser (broker)

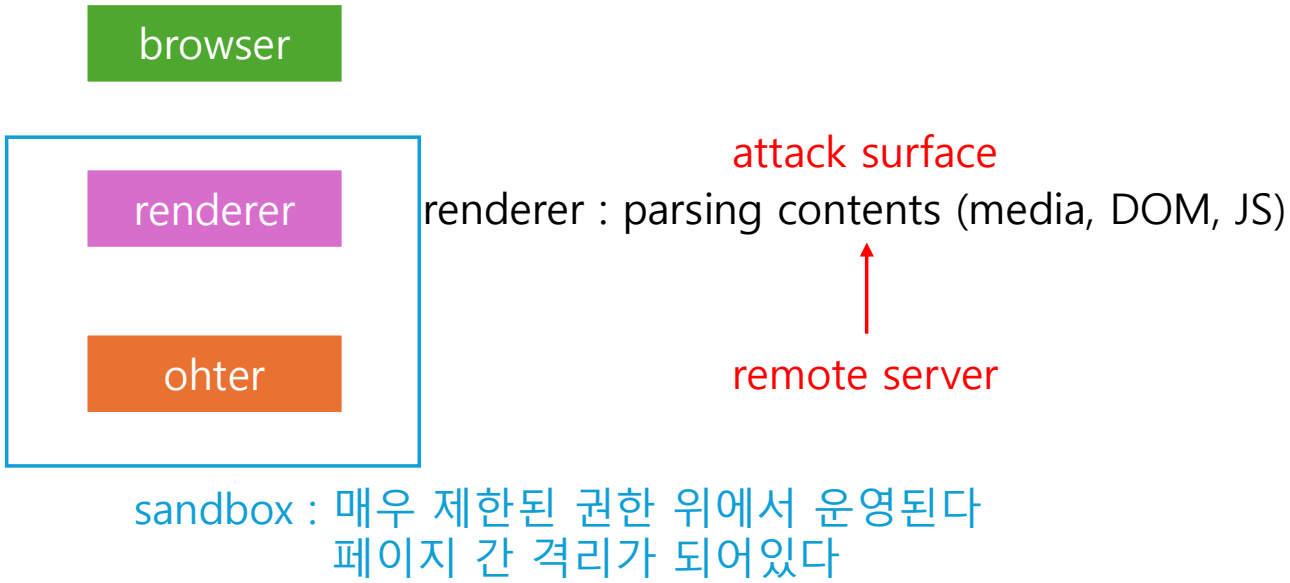
Web Browser 분석 아키텍처 challenge



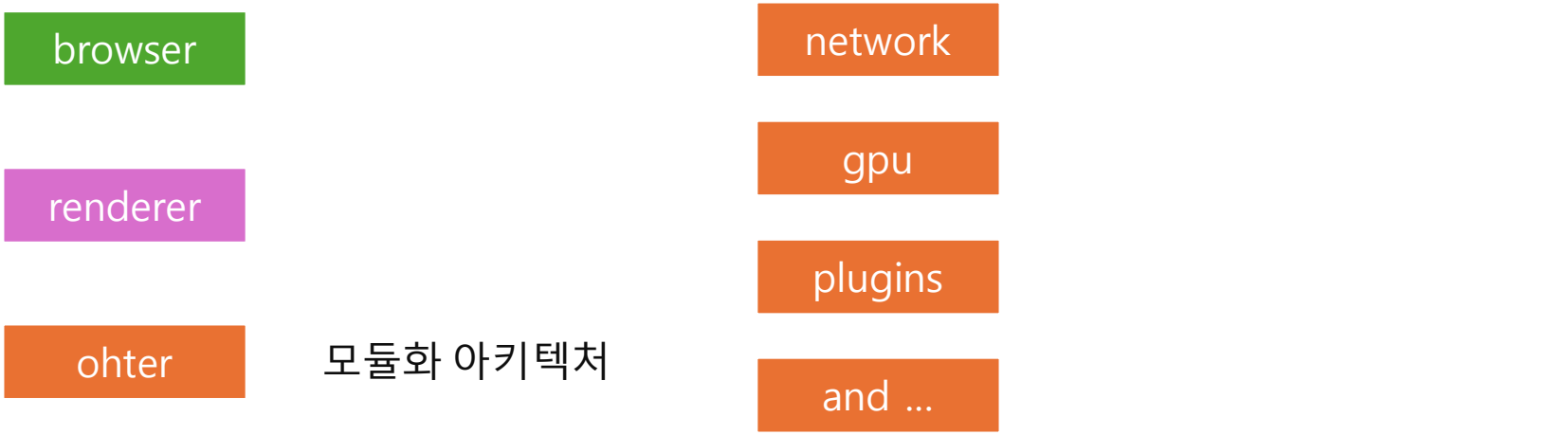
Web Browser 분석 아키텍처 challenge



Web Browser 분석 아키텍처 challenge

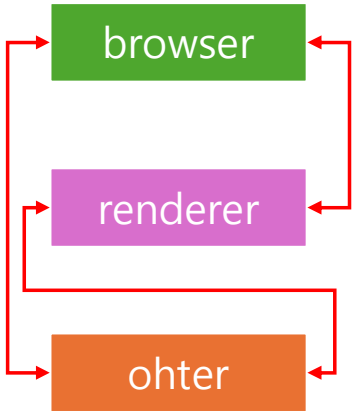


Web Browser 분석 아키텍처 challenge



기능에 따른 최소 권한만 부여

Web Browser 분석 아키텍처 challenge



IPC : process간 통신

browser process가 reference monitor로 작동하며 일부 작동을 제한한다

- 민감 데이터 접근
- system call등의 높은 권한이 필요한 동작

버그 발생 시 IPC filtering을 통해 브라우저 전체에 영향이 가지 않도록 한다

Web Browser 분석 아키텍처 challenge

SOP(Same-Origin Policy) : 동일 출처 정책

웹페이지가 상호작용할 수 있는 리소스를 제한

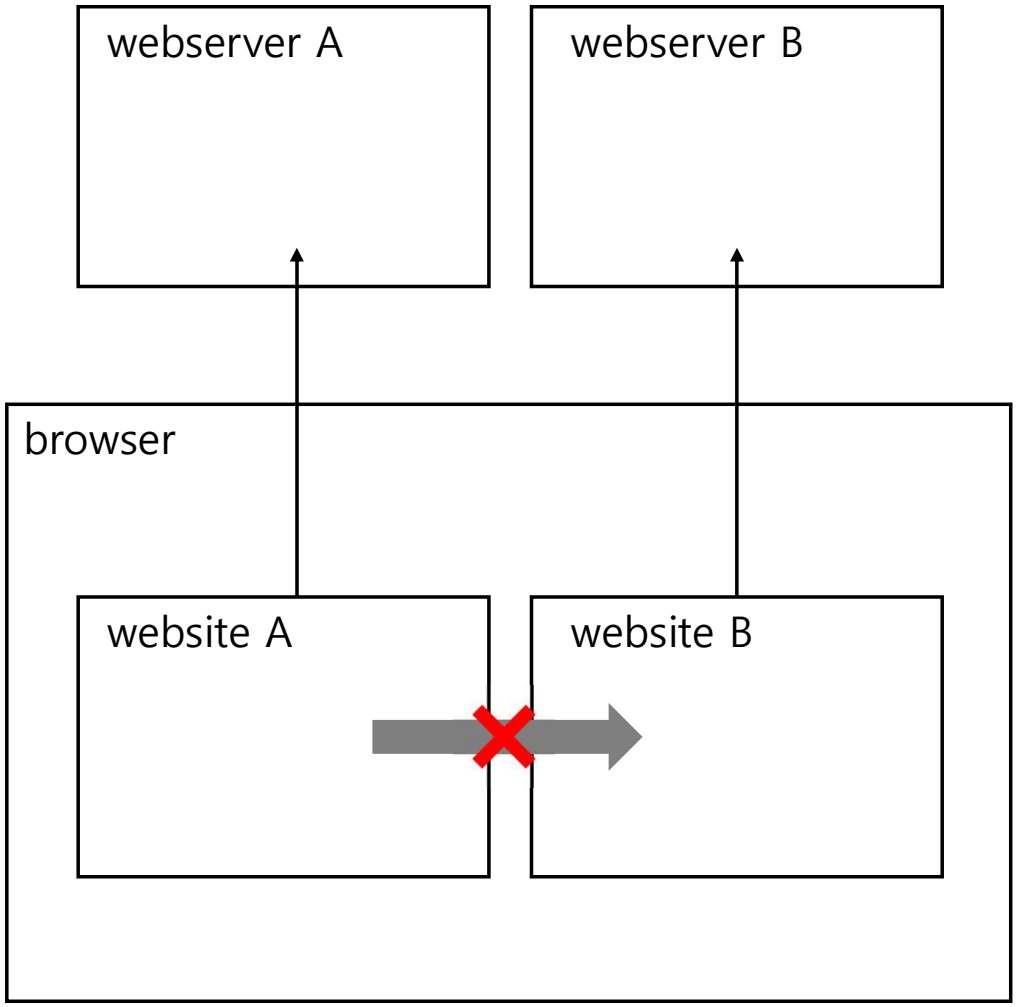
Origin : 프로토콜, 도메인, 포트 번호를 기준으로 정의

쿠키, DOM, AJAX 요청 등 특정 리소스와 상호작용할 때, 해당 리소스가 동일한 출처에서 온 경우에만 접근

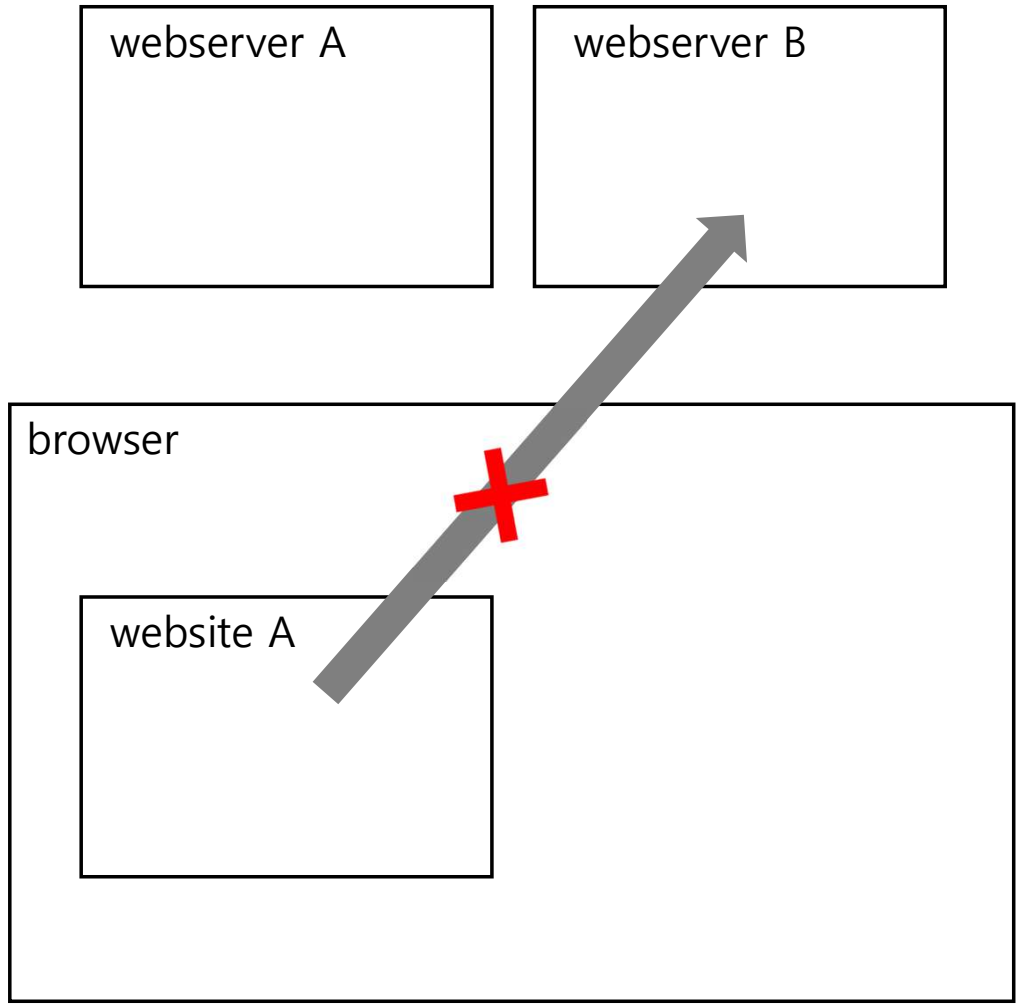
CDN에서 asset, js 등을 load, iframe 사용 시 타 origin 컨텐츠가 load된다

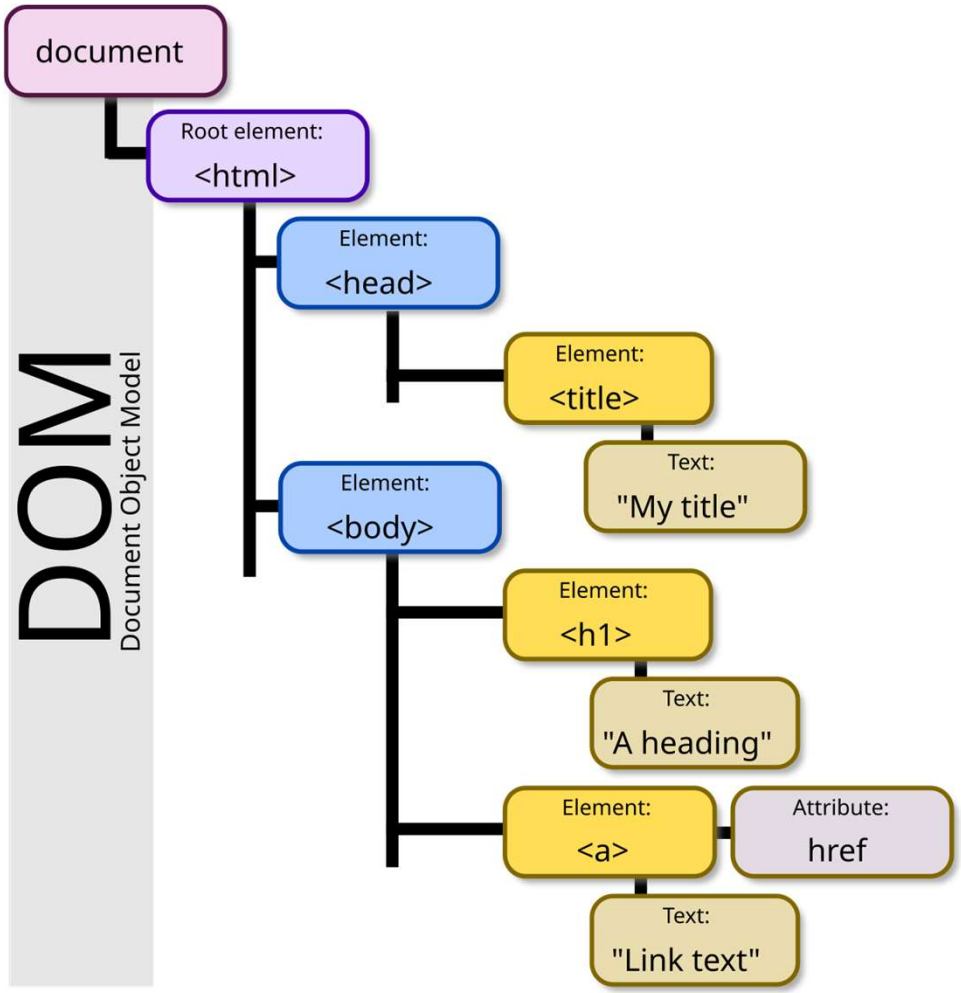

http://www.example.com(:80)
프로토콜 or
스키마
호스트
포트 번호

Web Browser 분석 아키텍처 challenge

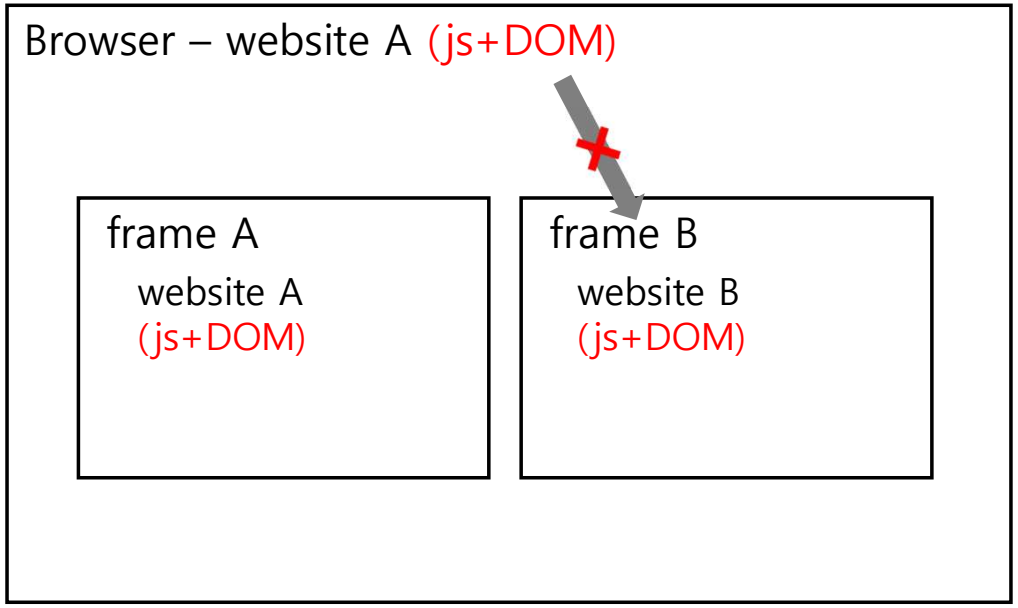


Web Browser 분석 아키텍처 challenge




DOM
Document Object Model
document
Root element:
<html>
Element:
<head>
Element:
<title>
Text:
"My title"
Element:
<body>
Element:
<h1>
Text:
"A heading"
Element:
<a>
Attribute:
href
Text:
"Link text"

Web Browser 분석 아키텍처 challenge



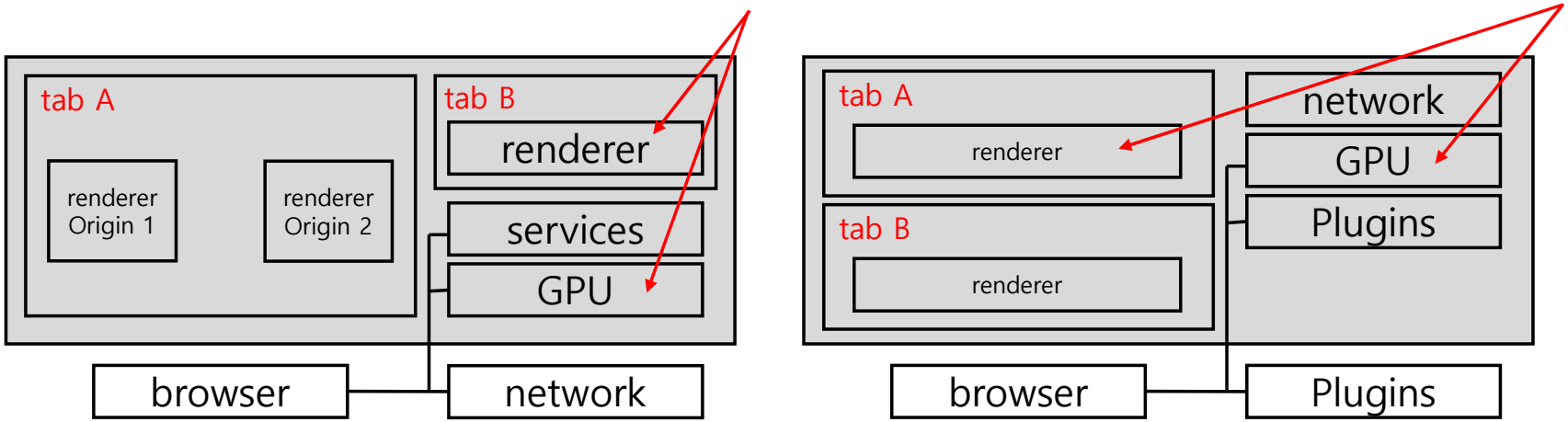
Web Browser 분석 아키텍처 challenge

CORS: Cross-Origin Resource Sharing

SOP 정책을 완화 하기 위해 설계된 표준

서버 헤더에 추가
Access-Control-Allow-Origin: * 🤭🤭🤭

Web Browser 분석 아키텍처 challenge – browser 간 차이

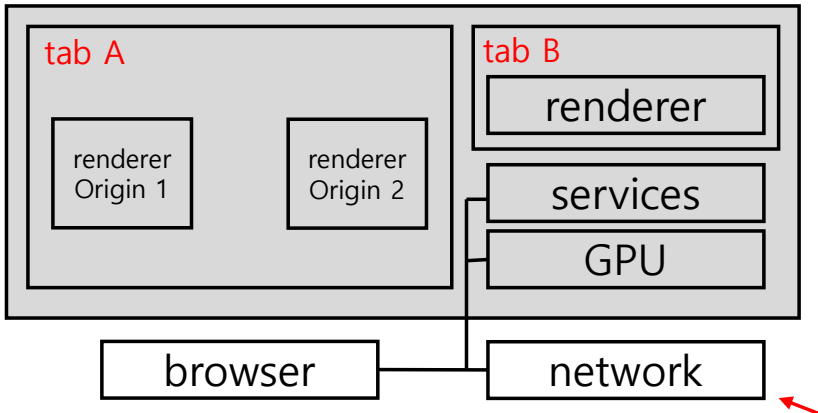


<chrome>

<safari>

Chrome & Safari 는 renderer와 GPU 가 분리되어 있다

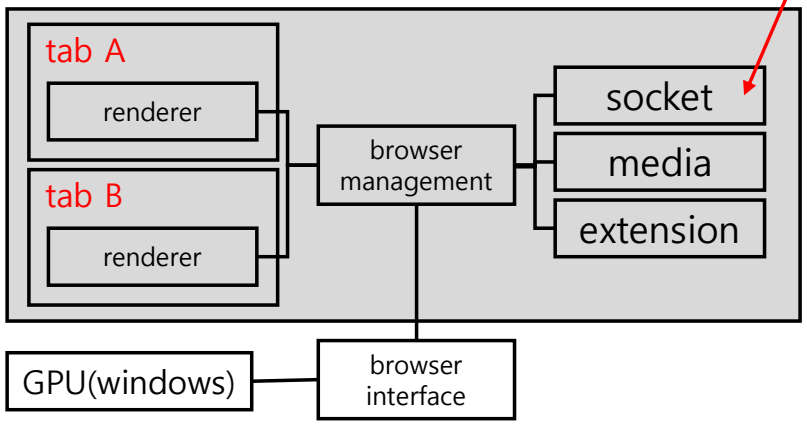
Web Browser 분석 아키텍처 challenge – browser 간 차이



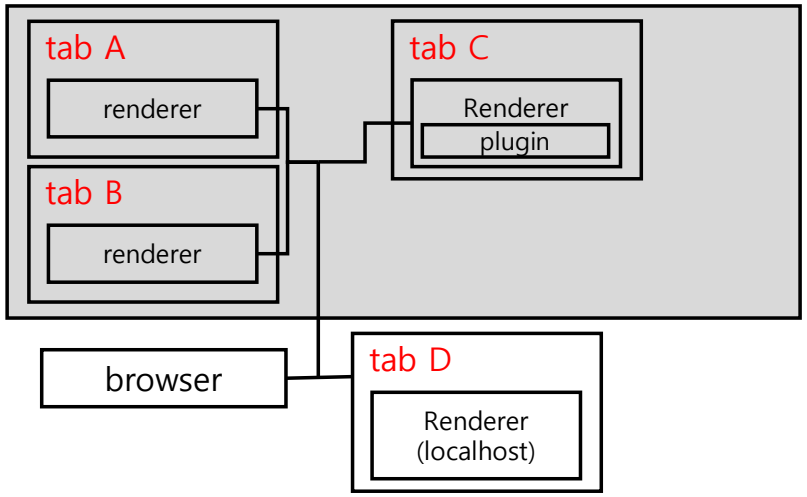
<chrome>

Chrome 은 network sandbox가 구현 중이라고 한다
(현 시점에서는 잘 모르겠다-서치 결과 못 찾음)

Chrome, Safari, FireFox는 network 프로세스를 별도로 관리

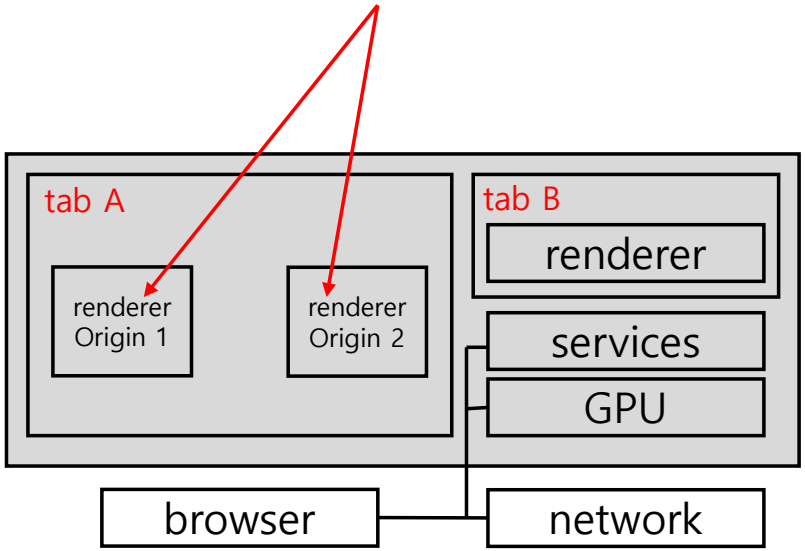


<firefox>

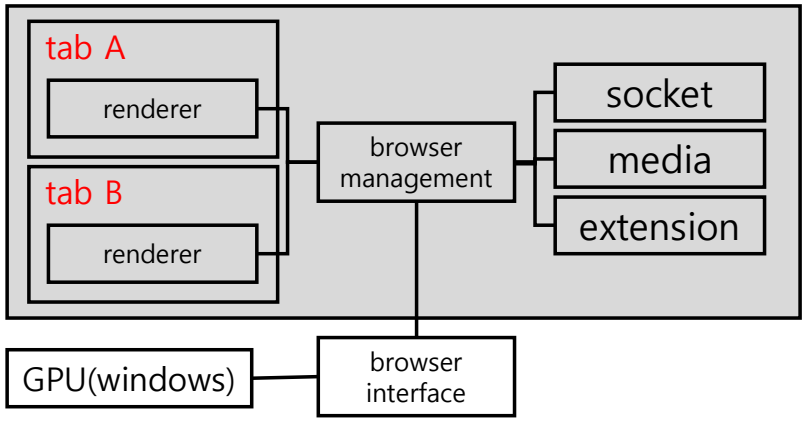


<Internet Explorer>

각각의 origin간 데이터 공유 차단 -> UXSS 차단



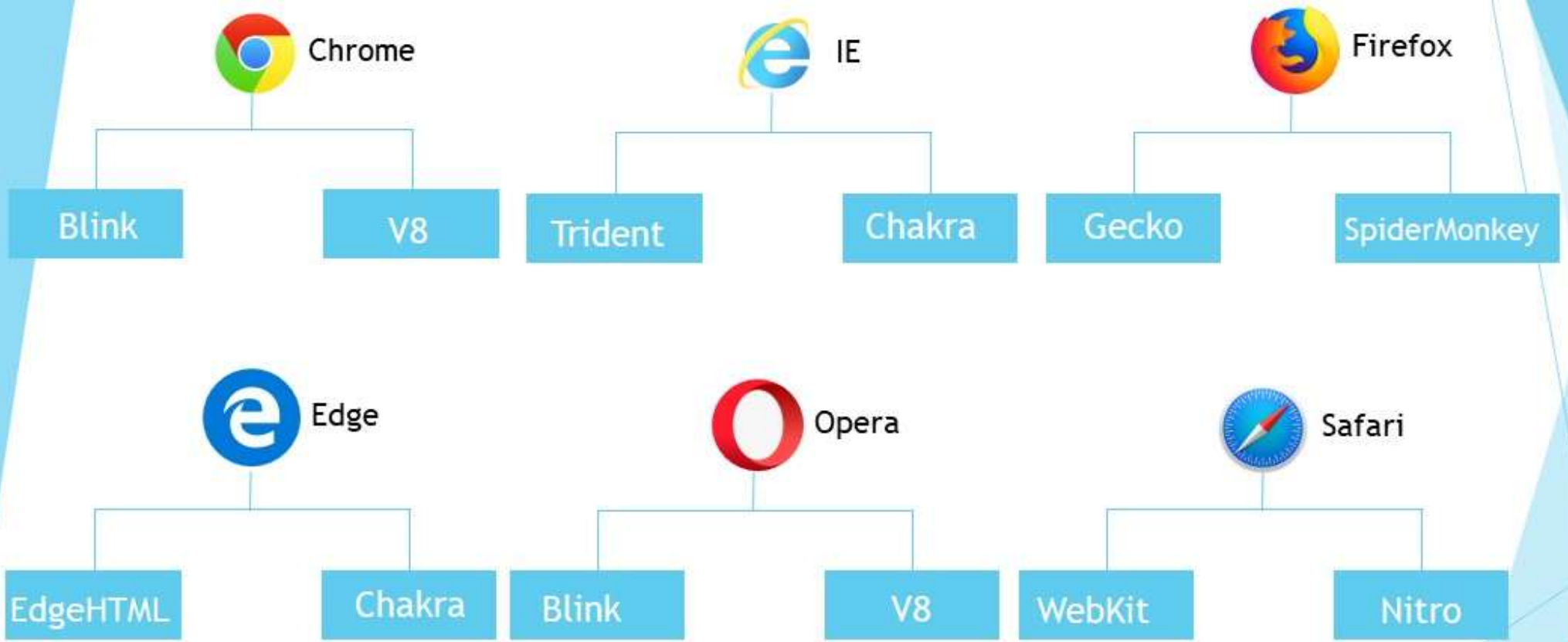
<chrome>



<firefox>

Fission : Firefox 88 베타에 포함되어 출시
-> Chrome site 격리와 같은 기능

Rendering Engines & JavaScript Engines



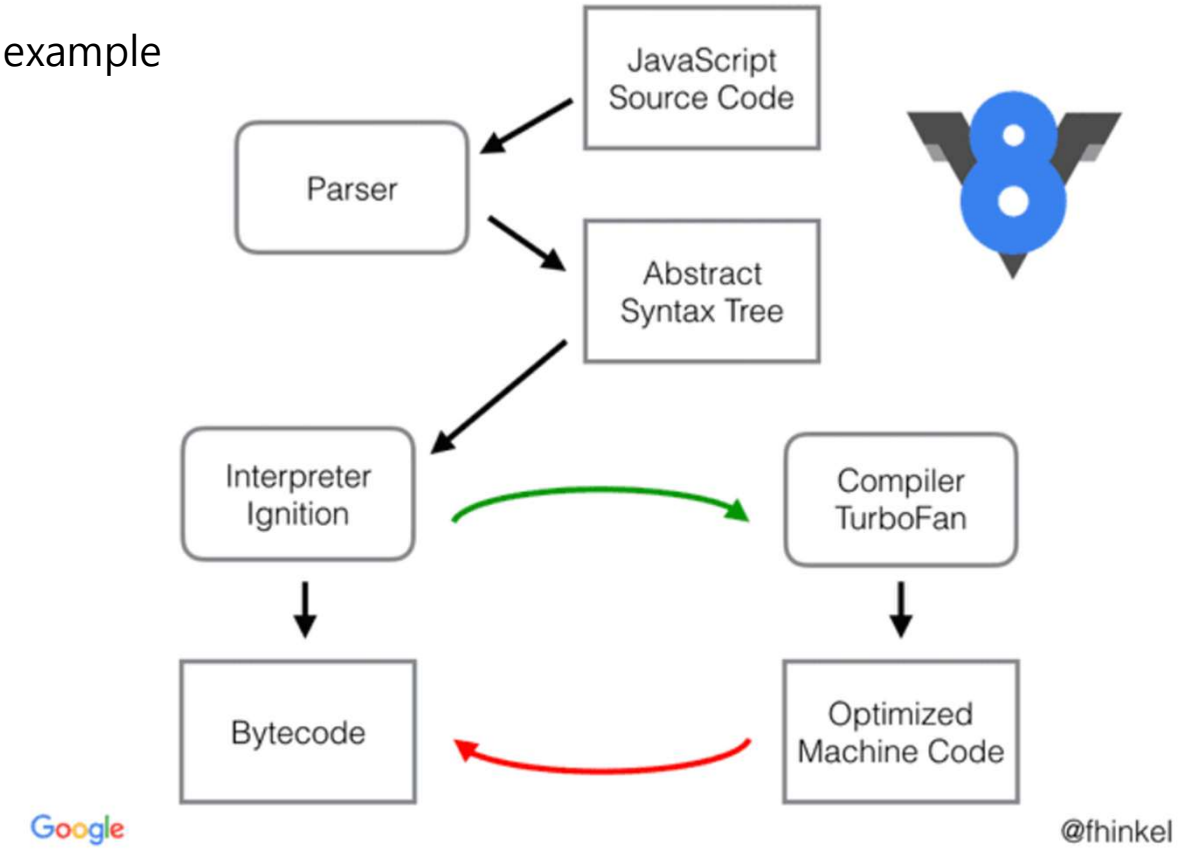
Javascript 처리를 위해 js engine이 사용된다.

Javascript -> 기계어 흐름으로 처리
JIT(just in time) 컴파일을 통해 실행 속도를 빠르게 한다.
JIT 컴파일러는 컴파일 타임에 최적화를 도입한다.

최적화 시점에 메모리 커럽션이 발생하는 기계어를 생성하는 경우가 있다.

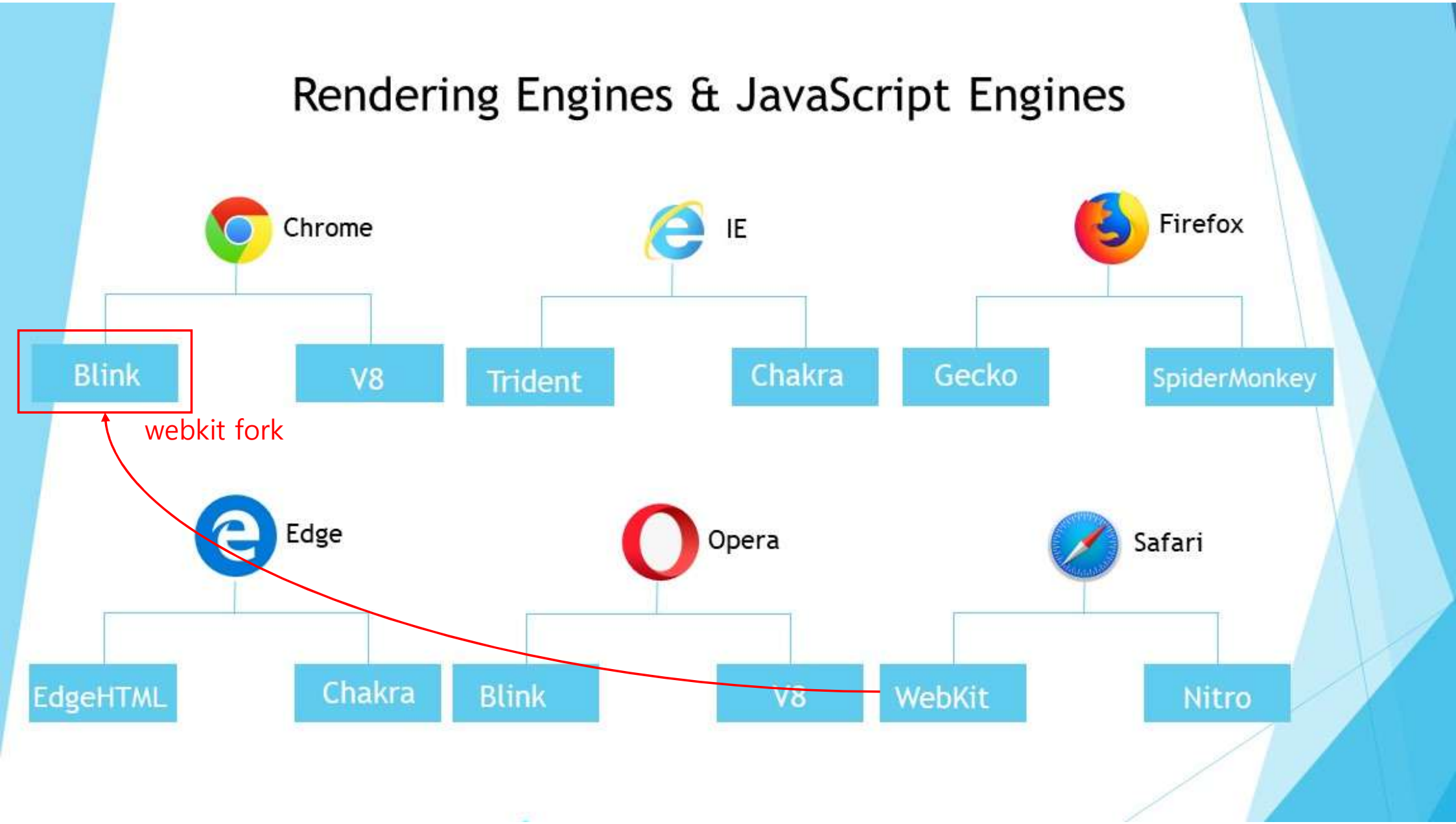
Js engine이 다양하지만, 표준이 같기에 어택 벡터도 유사한 경우가 많다.
심지어 webserver backend engine 중에도 node js 와 같은 js engine을 공유하는 경우가 있기에 파급력이 높다

V8 example

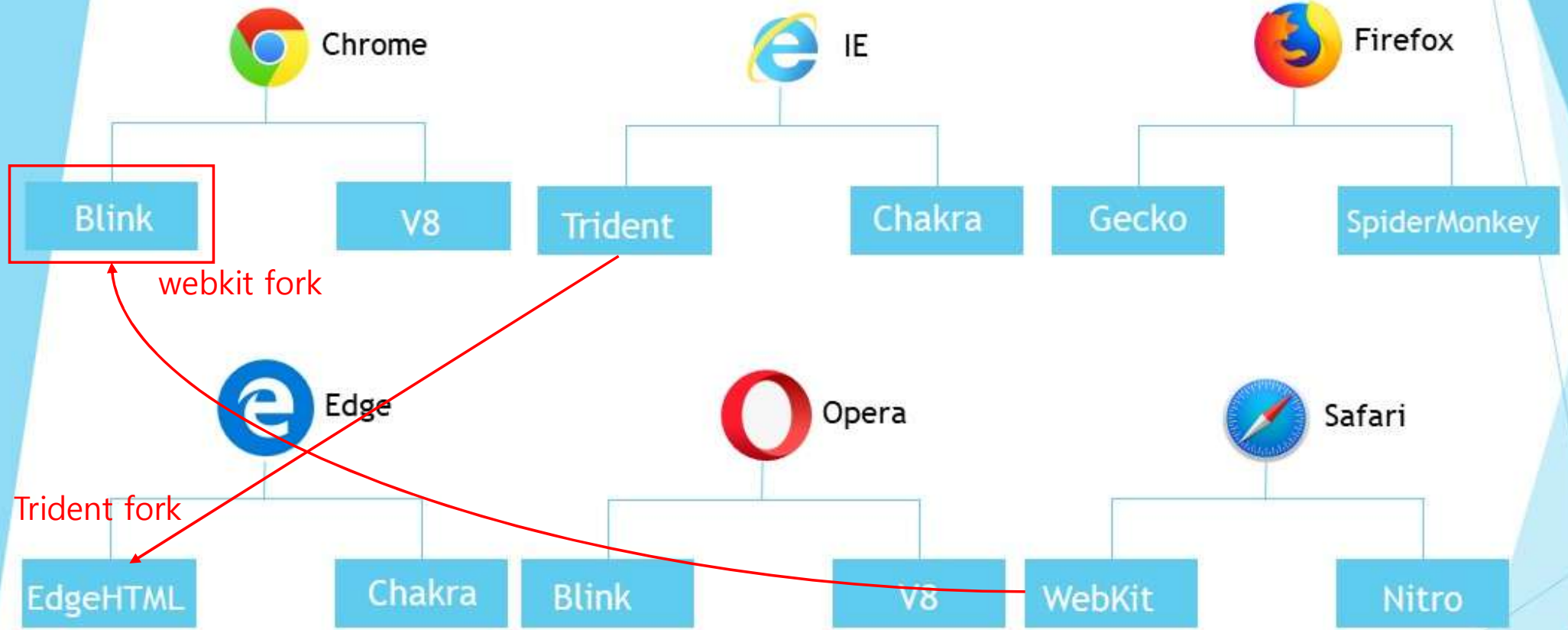


JSConf EU 2017에서 발표한 Franziska Hinkelmann님의 자료

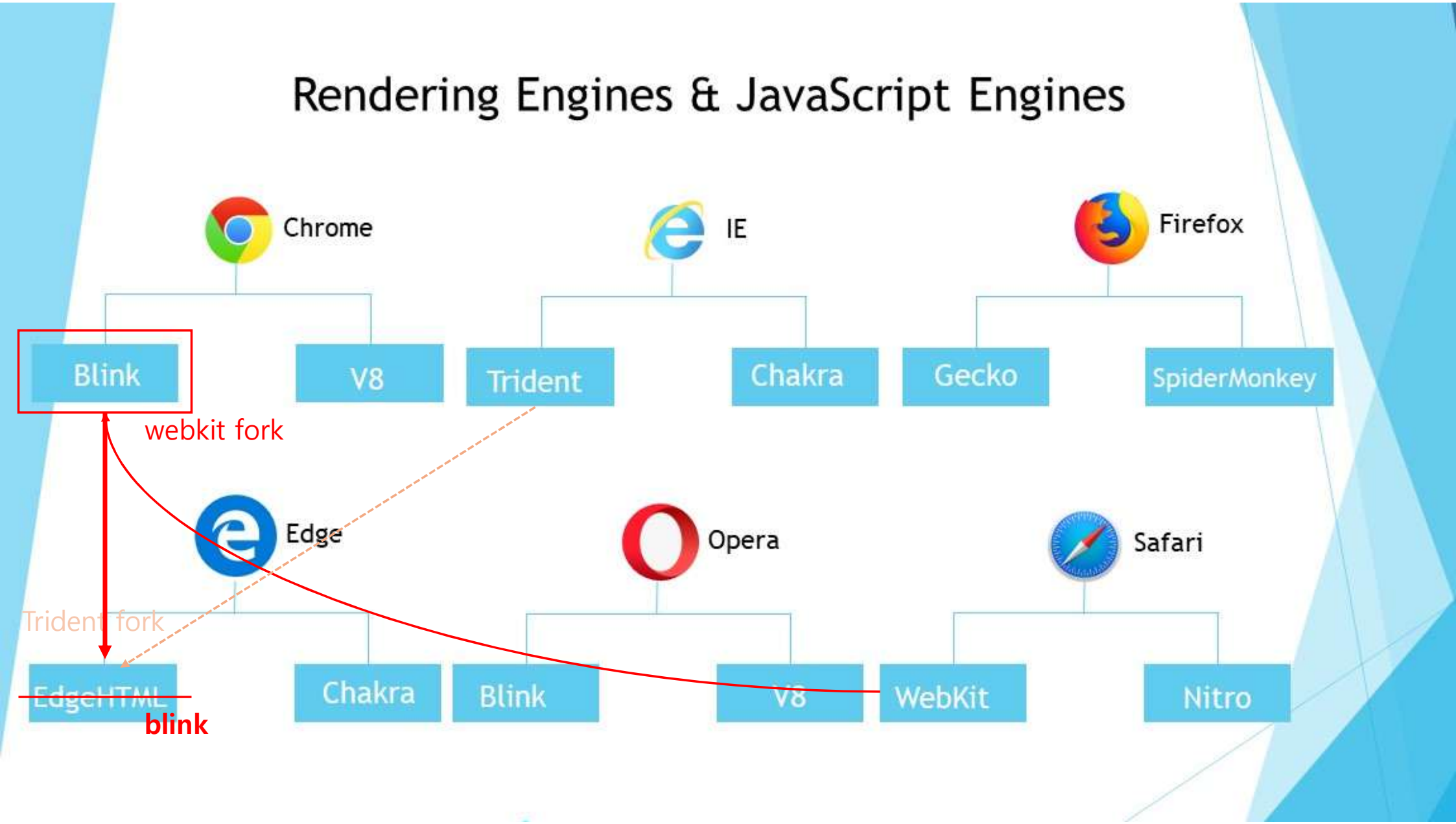
Rendering Engines & JavaScript Engines



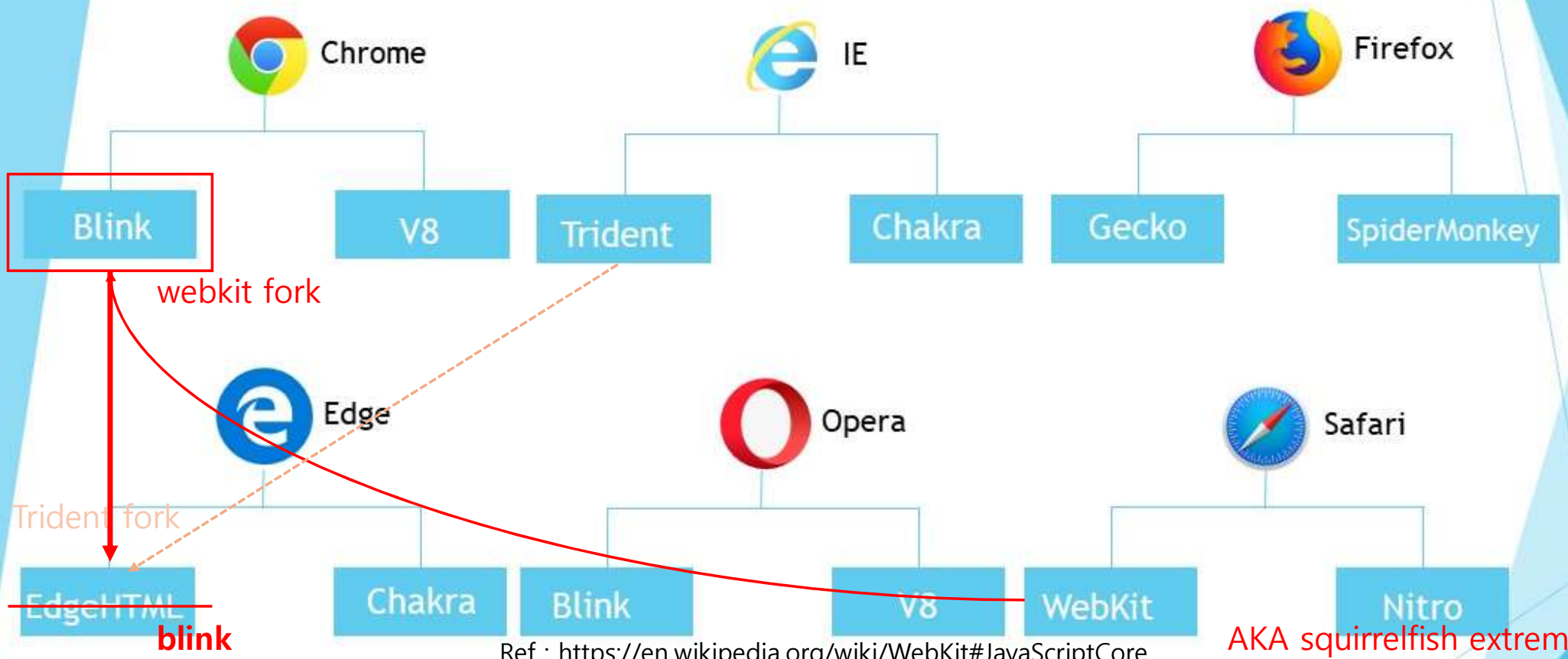
Rendering Engines & JavaScript Engines



Rendering Engines & JavaScript Engines



Rendering Engines & JavaScript Engines



Ref : https://en.wikipedia.org/wiki/WebKit#JavaScriptCore

Webkit의 js engine은 JSC(javascript core)

그러나 safari는 JSC -> Nitro 로 전환하게 된다(비슷하지만 opensource가 아님)

https://www.appsierra.com/blog/microsoft-edge-browser

논문에 등장하는 내용은 아니다

Edge HTML 이 시장 점유율 확보에 실패하였다
시장이 Chrome 표준에 맞추어 호환성 이슈가 빈번히 발생하였다

Web Browser 분석 아키텍처 challenge

Rendering 엔진이 다양해짐에 있어 기능 구현 전략도 다양해졌다

성능과 최신 기술을 중시하기도 하며, 안정성을 중시하는 브라우저도 등장하였다

표준이 업데이트 되는 속도에 맞추는 브라우저로 인한 호환성 이슈가 발생하였고,
브라우저마다 지원하는 기능이 달라지면서 공격 벡터도 브라우저 마다 달라졌다.

Web Browser 분석 아키텍처 challenge – OS 특이성

Sandbox 구현 방식이 OS와 아키텍처의 의존성을 띈다

Sandbox 구현 기능

- 권한 감소 : 플랫폼의 권한 시스템 활용
- 도메인 분리 : sanbox process가 접근 가능한 리소스 공간
- 공격 표면 축소 : 커널 서비스, 드라이버에 접근 제한 (불필요한 api 호출 권한 제거)

Web Browser 분석 아키텍처 challenge – OS 특이성

Windows 경우, security token 을 사용한다

Renderer : low integrity token으로 실행

Broker(browser) : Medium integrity token으로 실행

Renderer->Broker write 권한 없음

Windows는 세분화된 접근 제어를 제공하지 않음

chrome, firefox는 자체 IPC프로토콜로 세분화된 규칙을 적용한다

Edge는 windows8에서 도입된 AppContainer를 활용하여 세분화된 규칙을 적용한다

DACL을 통해, 손상된 sandbox가 다른 process를 건들지 못하게 한다([장표]DACL이 뭔지 알아보기)

Web Browser 분석 아키텍처 challenge – OS 특이성

Linux의 경우, 3가지 방법을 사용한다.

- Seccomp : eBPF 언어를 기반으로 한 syscall filter (broker가 사용자 정의 규칙을 적용한다)
- Chroot : 파일 접근을 제한하기 위해 chroot jail을 사용한다.
  Firefox의 경우 broker가 fd를 얻어와서 renderer에게 준다.
- Namespace : 사용자 네임스페이스를 생성하여 프로세스가 별도의 UID,GID를 가진다
  다른 프로세스가 이를 접근할 수 없다. 리소스 전용공간을 할당하는 셈

Web Browser 분석 아키텍처 challenge – OS 특이성

macOS의 경우, .sb파일을 이용한다

.sb file -> sandbox profile 파일

- allowlist : 특정 파일 경로만 접근이 허용된다.(그 외 차단)
- 리소스 접근 정의 : 네트워크, 공유 메모리 등 리소스 접근 권한을 정의한다.
- syscall filter : seccomp와 유사한 필터링이 존재. Safari에만 적용된다.

Web Browser 분석 아키텍처 challenge – OS 특이성

mobile의 경우 sandbox 이용에 제약이 크다

Sandbox는 메모리 리소스를 많이 소모

하드웨어 자원의 제한으로 sandbox가 비활성화되거나 정책 조정을 통해 완화될 수 있다

Web Browser 분석 아키텍처 challenge – OS 특이성

mobile(android)

Origin Isolation 은 메모리 1.9GB가 넘을 때 활성화 된다
또는, 이용자가 비밀번호를 입력하는 경우에 활성화 된다

mobile(ios)

macOS와 다름
모바일 특화된 서비스와 드라이버가 노출되어 차별화 되어있다(잘 모르겠지만 뭔가 다르다고 함)

Web Browser Exploit

목표

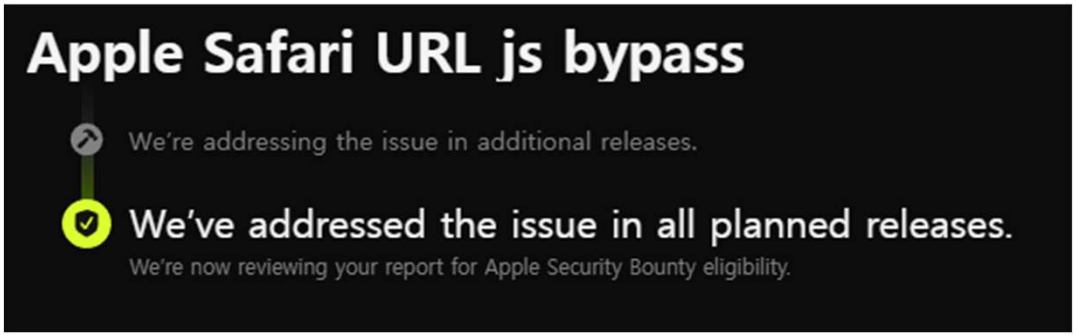
- UXSS(Universal Cross Site Scripting)
- ACE(Arbitrary code execution)

Web Browser Exploit

UXSS(Universal Cross Site Scripting) 란?

XSS공격은 웹사이트 관리자가 아닌 이가 웹 페이지에 악성 스크립트를 삽입할 수 있는 취약점이다. 일반적인 XSS 공격과 달리 UXSS는 XSS 조건을 생성하고 악성 코드를 실행하기 위해 브라우저 또는 브라우저 확장에서 클라이언트 측 취약점을 악용하는 공격 유형이다.

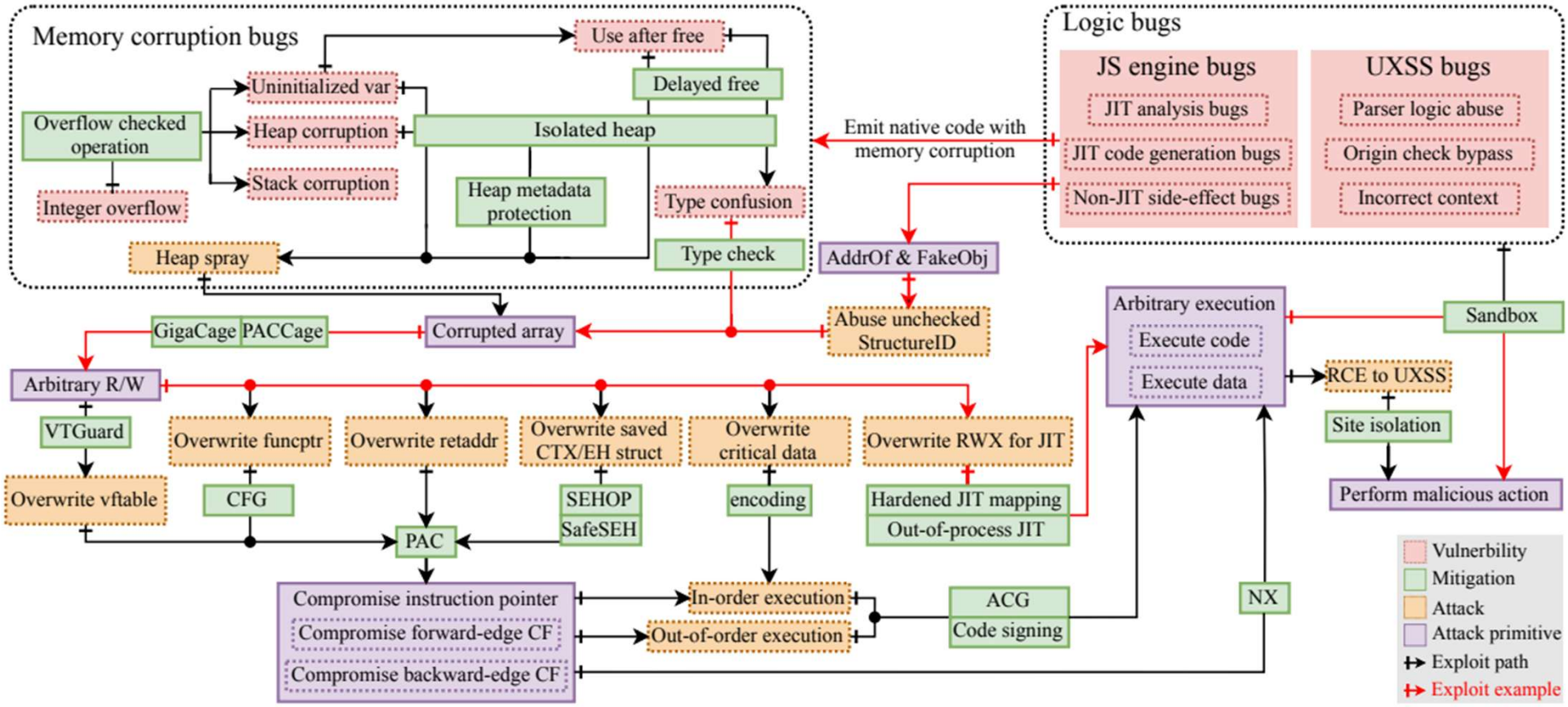
Safari에서 webcam 권한 얻기

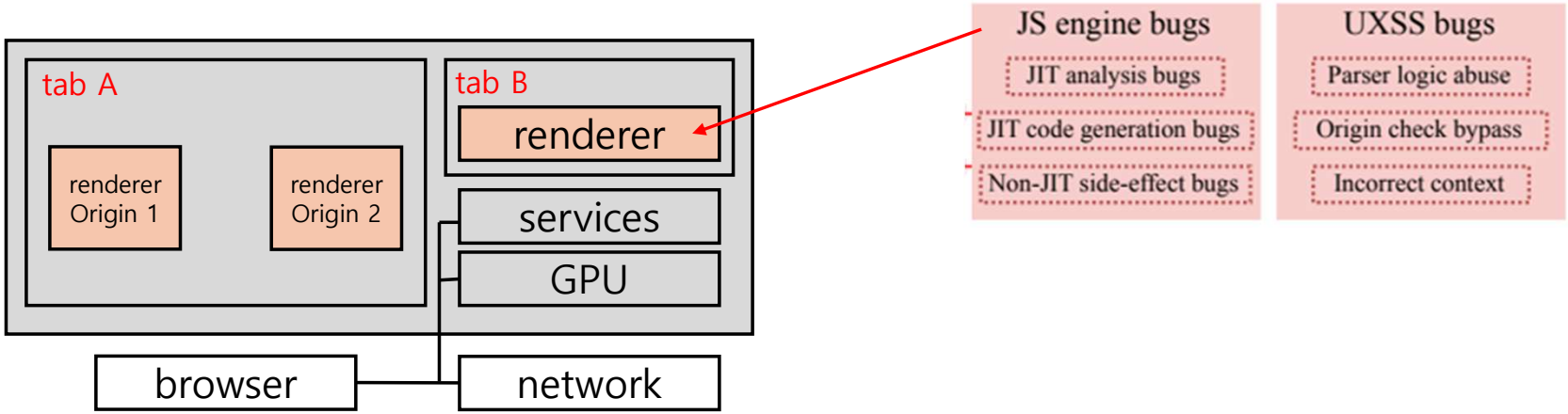


Web Browser Exploit

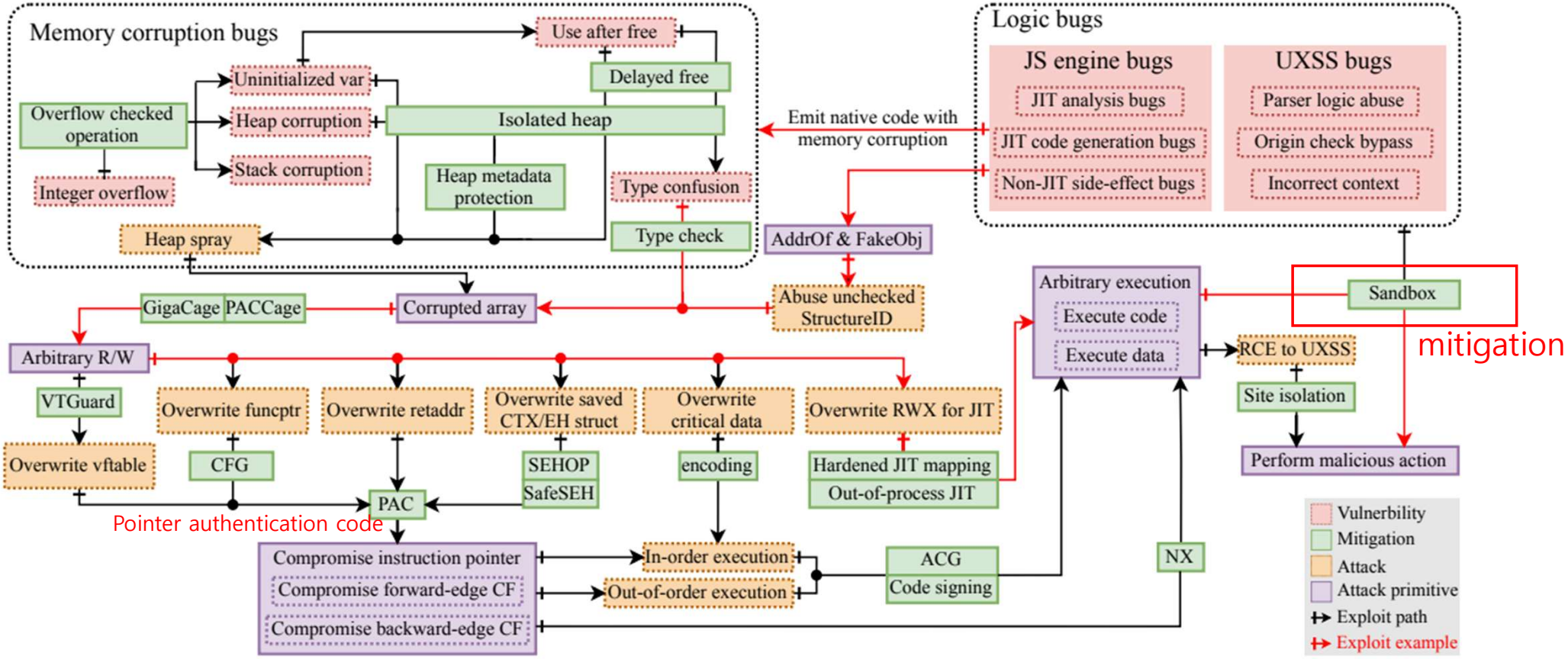
ACE(Arbitrary code execution)란?

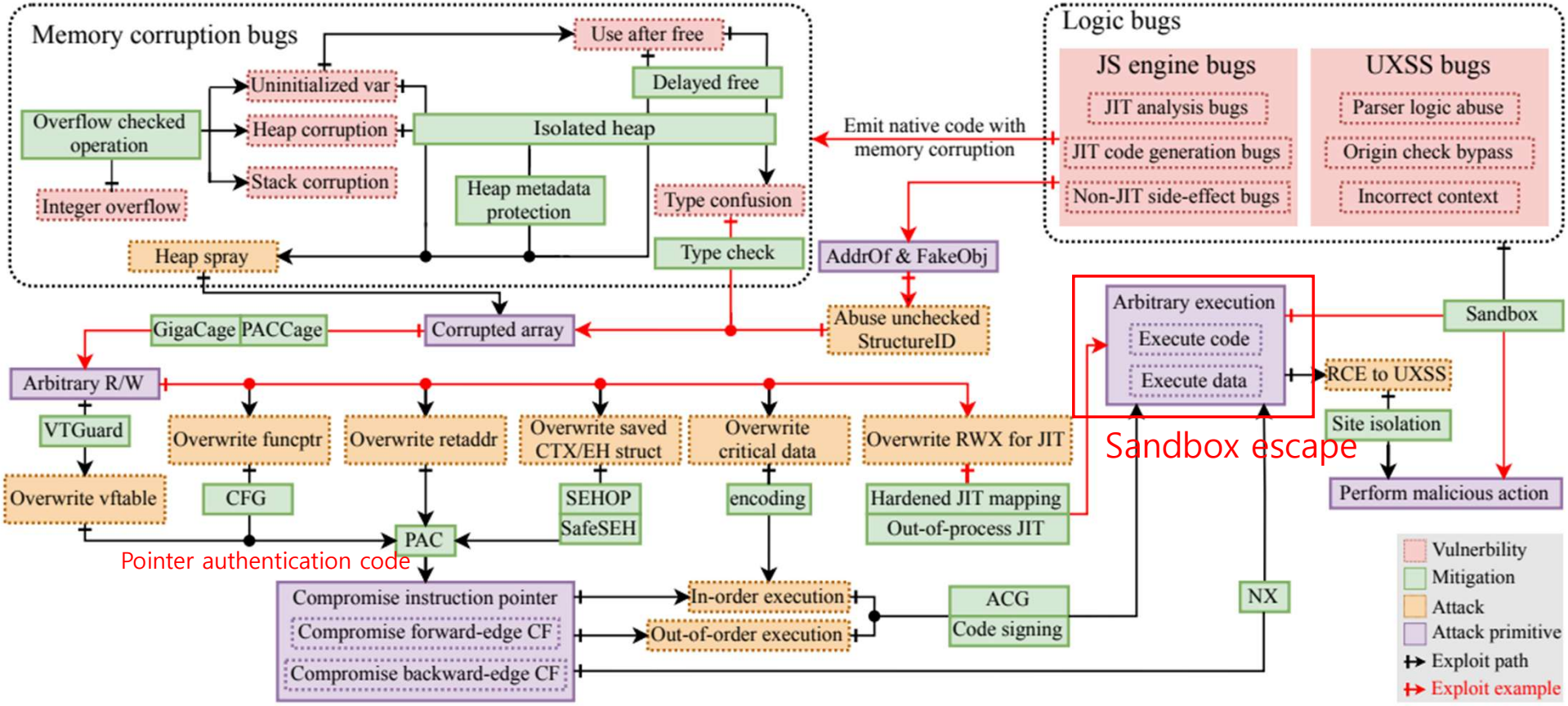
공격자가 시스템의 보안 취약점을 악용하여 임의의 코드를 실행할 수 있는 보안 위협을 의미
이를 통해 권한 상승, 데이터 유출, 서비스 장애 등을 유발할 수 있다


Memory corruption bugs
Use after free
Uninitialized var
Delayed free
Overflow checked operation
Heap corruption
Isolated heap
Stack corruption
Heap metadata protection
Type confusion
Integer overflow
Heap spray
Type check
Logic bugs
JS engine bugs
JIT analysis bugs
JIT code generation bugs
Non-JIT side-effect bugs
UXSS bugs
Parser logic abuse
Origin check bypass
Incorrect context
Emit native code with memory corruption
AddrOf & FakeObj
Abuse unchecked StructureID
GigaCage
PACCage
Corrupted array
Arbitrary execution
Execute code
Execute data
Sandbox
RCE to UXSS
Site isolation
Perform malicious action
Arbitrary R/W
VTGuard
Overwrite vftable
Overwrite funcptr
CFG
Overwrite retaddr
Overwrite saved CTX/EH struct
SEHOP
SafeSEH
Overwrite critical data
encoding
Overwrite RWX for JIT
Hardened JIT mapping
Out-of-process JIT
PAC
Compromise instruction pointer
Compromise forward-edge CF
Compromise backward-edge CF
In-order execution
Out-of-order execution
ACG
Code signing
NX
Vulnerbility
Mitigation
Attack
Attack primitive
Exploit path
Exploit example

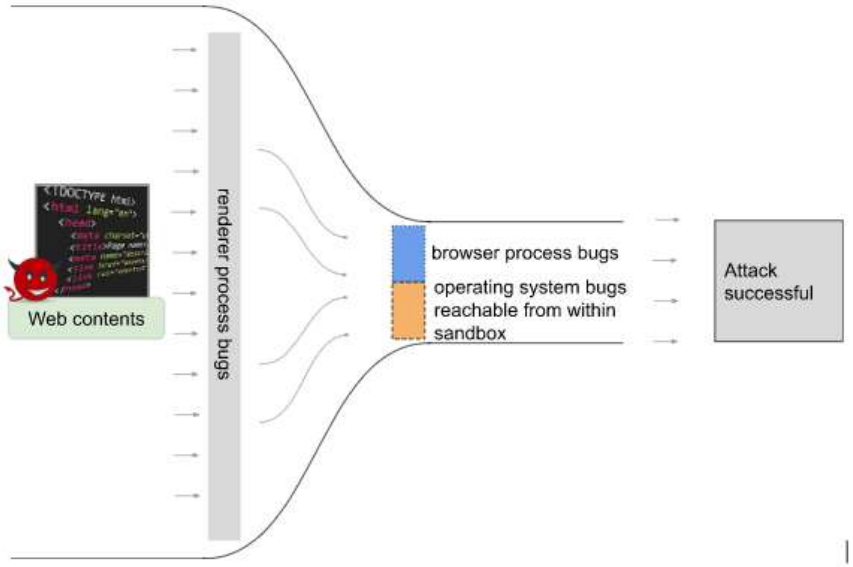

tab A
renderer
Origin 1
renderer
Origin 2
tab B
renderer
services
GPU
browser
network
JS engine bugs
JIT analysis bugs
JIT code generation bugs
Non-JIT side-effect bugs
UXSS bugs
Parser logic abuse
Origin check bypass
Incorrect context

<chrome>


Memory corruption bugs
Use after free
Uninitialized var
Delayed free
Overflow checked operation
Heap corruption
Isolated heap
Stack corruption
Heap metadata protection
Integer overflow
Type confusion
Heap spray
Type check
Logic bugs
JS engine bugs
JIT analysis bugs
JIT code generation bugs
Non-JIT side-effect bugs
UXSS bugs
Parser logic abuse
Origin check bypass
Incorrect context
Emit native code with memory corruption
AddrOf & FakeObj
Abuse unchecked StructureID
GigaCage
PACCage
Corrupted array
Arbitrary R/W
VTGuard
Overwrite vftable
Overwrite funcptr
CFG
Overwrite retaddr
Overwrite saved CTX/EH struct
SEHOP
SafeSEH
Overwrite critical data
encoding
Overwrite RWX for JIT
Hardened JIT mapping
Out-of-process JIT
Pointer authentication code
PAC
Compromise instruction pointer
Compromise forward-edge CF
Compromise backward-edge CF
In-order execution
Out-of-order execution
ACG
Code signing
NX
Arbitrary execution
Execute code
Execute data
Sandbox
RCE to UXSS
Site isolation
Perform malicious action
mitigation
Vulnerbility
Mitigation
Attack
Attack primitive
Exploit path
Exploit example


Memory corruption bugs
Use after free
Uninitialized var
Delayed free
Overflow checked operation
Heap corruption
Isolated heap
Stack corruption
Heap metadata protection
Integer overflow
Type confusion
Heap spray
Type check
Logic bugs
JS engine bugs
JIT analysis bugs
JIT code generation bugs
Non-JIT side-effect bugs
UXSS bugs
Parser logic abuse
Origin check bypass
Incorrect context
Emit native code with memory corruption
AddrOf & FakeObj
Abuse unchecked StructureID
GigaCage
PACCage
Corrupted array
Arbitrary execution
Execute code
Execute data
Sandbox
RCE to UXSS
Site isolation
Perform malicious action
Sandbox escape
Arbitrary R/W
VTGuard
Overwrite vftable
Overwrite funcptr
CFG
Overwrite retaddr
Overwrite saved CTX/EH struct
SEHOP
SafeSEH
Overwrite critical data
encoding
Overwrite RWX for JIT
Hardened JIT mapping
Out-of-process JIT
Pointer authentication code
PAC
Compromise instruction pointer
Compromise forward-edge CF
Compromise backward-edge CF
In-order execution
Out-of-order execution
ACG
Code signing
NX
Vulnerbility
Mitigation
Attack
Attack primitive
Exploit path
Exploit example

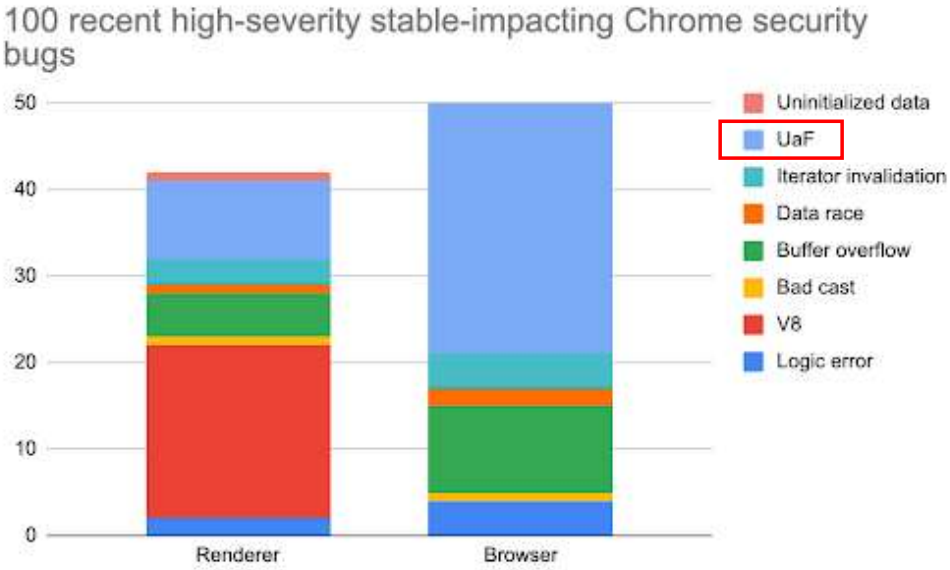
MiraclePtr – UAF mitigation [공부 중 알아낸 논문에서 빠진 내용]

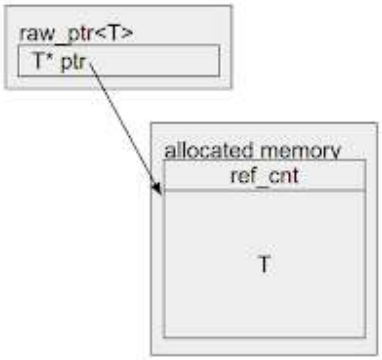


Blue space를 줄이기 위해 미라클포인터 도입
Orange는 os interface를 앞으로 줄여 갈 것이라 밝힘

Ref : https://security.googleblog.com/2022/09/use-after-freedom-miracleptr.html

MiraclePtr – UAF mitigation [공부 중 알아낸 논문에서 빠진 내용]

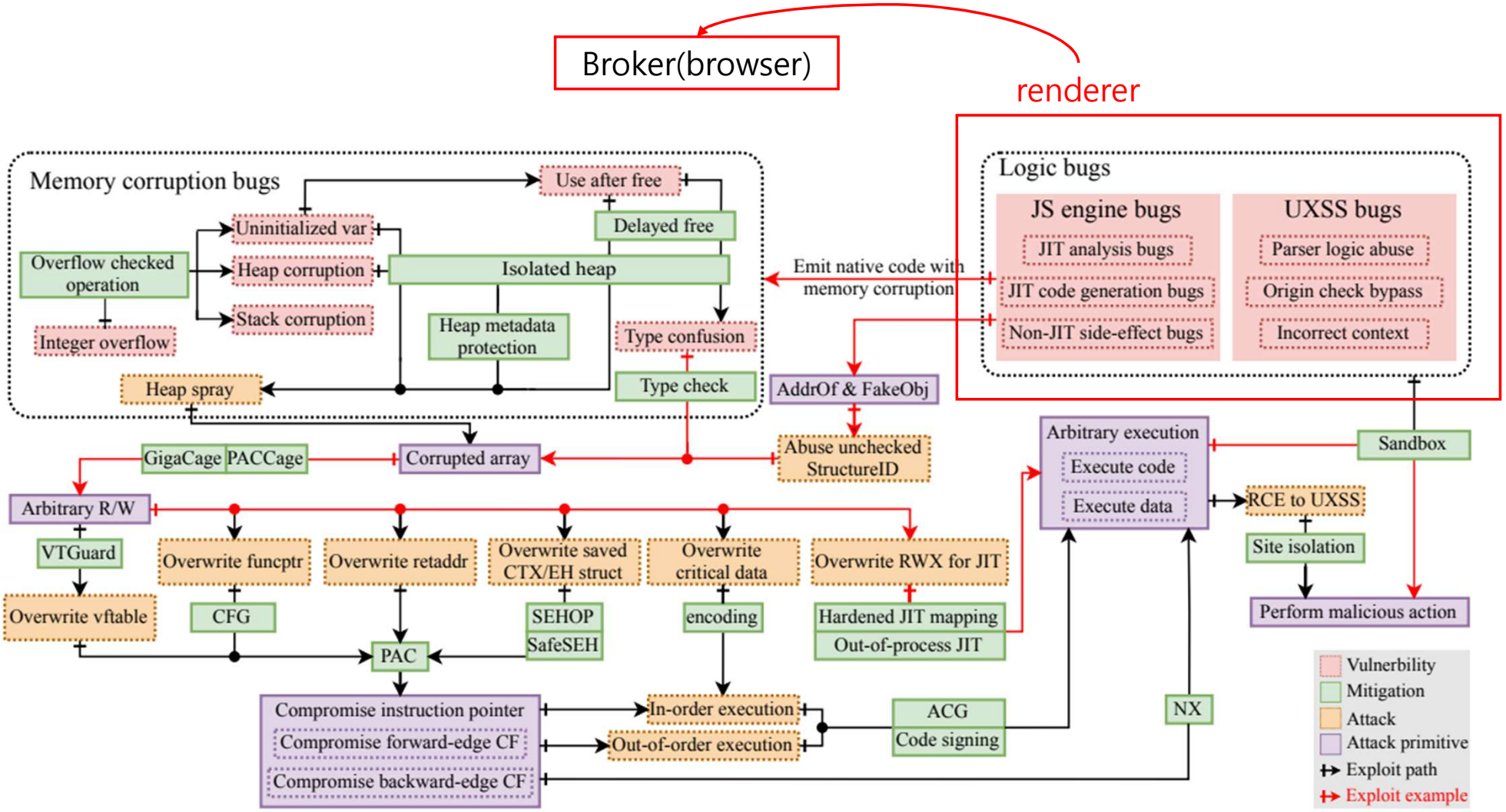




BackupRefPtr을 chrome 힙 할당자에 추가

free,delete 시 reference counter가 있는 경우 해제안하고 격리

Ref : https://security.googleblog.com/2022/09/use-after-freedom-miracleptr.html

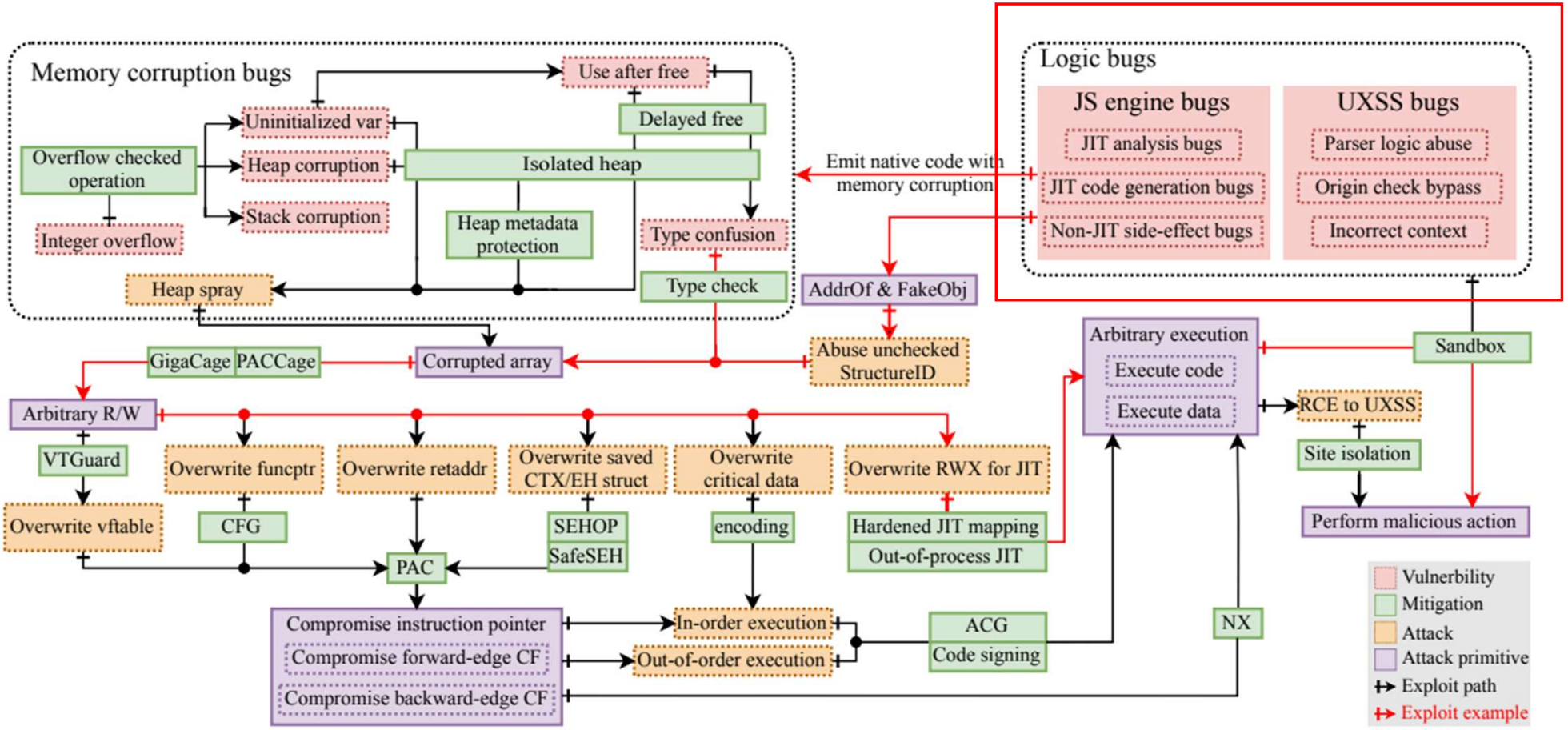

Broker(browser)
renderer
Memory corruption bugs
Use after free
Uninitialized var
Delayed free
Overflow checked operation
Heap corruption
Isolated heap
Stack corruption
Heap metadata protection
Integer overflow
Type confusion
Heap spray
Type check
Logic bugs
JS engine bugs
JIT analysis bugs
JIT code generation bugs
Non-JIT side-effect bugs
UXSS bugs
Parser logic abuse
Origin check bypass
Incorrect context
Emit native code with memory corruption
AddrOf & FakeObj
Abuse unchecked StructureID
GigaCage
PACCage
Corrupted array
Arbitrary R/W
VTGuard
Overwrite vftable
Overwrite funcptr
CFG
Overwrite retaddr
Overwrite saved CTX/EH struct
SEHOP
SafeSEH
Overwrite critical data
encoding
Overwrite RWX for JIT
Hardened JIT mapping
Out-of-process JIT
PAC
Compromise instruction pointer
Compromise forward-edge CF
Compromise backward-edge CF
In-order execution
Out-of-order execution
ACG
Code signing
NX
Arbitrary execution
Execute code
Execute data
Sandbox
RCE to UXSS
Site isolation
Perform malicious action
Vulnerbility
Mitigation
Attack
Attack primitive
Exploit path
Exploit example

Renderer가 쓸 수 있는 드라이버,
커널 인터페이스만으로

kernel

Privilege escalation

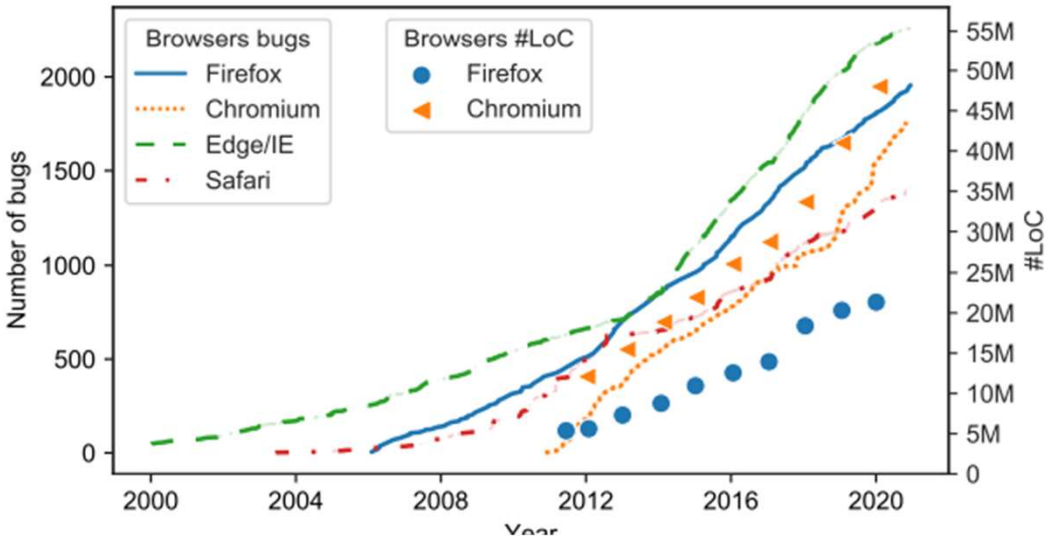
renderer



Web Browser Bug trands

	Firefox†	Chromium†	Safari/WebKit‡	Edge/IE‡
(1) Total CVEs	2190	2582	1436	2278
(2) Collected CVEs	2066	1912	1436	2278
(3) Pwn2Own [61]	14	9	37	31
(4) Google P0 [58]	7	22	8	22

CVE는 chromium 이 가장 많다
Pwn2Own exploit은 safari가 가장 많다



Web Browser Bug trends

Js engine : memory corruption, 부적절한 JIT 컴파일러 최적화
Renderer : contents processing(parsing으로 추정) -> sandbox escape

Web Browser Bug trends

Js engine : memory corruption, 부적절한 JIT 컴파일러 최적화
Renderer : contents processing(parsing으로 추정) -> sandbox escape

mitigation (bug trends에 따른 mitigation) :

JIT Hardening (JIT bug)
Pointer Authentication Codes (memory corruption)
Site Isolation (sandbox 개선)

Overwrite protection : 실행 or 쓰기 권한으로 설정 – 뒤에서 자세히 다룸
XOM : iOS 10 on ARMv8에서 jit 코드 주소를 숨기는 방법 – 실행 전용 메모리 공간

Web Browser Bug trends

Exploitable bug의 34%가 js engine 취약점을 활용한다고 한다.
그 중에서도 JIT bug가 상당 수

addrof : js객체의 주소를 릭
fakeobj : 임의 주소를 객체로 변환하여 접근

Web Browser Bug trends

DOM bug : UAF가 대부분

type confusion 등을 type check와 같은 mitigation으로 대응
(한 때 성행하였으나, 최근에는 선호되는 방식이 아님)

Object Shape Authentication : StructureID 또는 ShapeID에 랜덤 엔트로피 비트를 추가

Address Space Isolation :

GigaCage: 4GB 가상 메모리 영역을 사용해 객체를 다른 힙으로 분리

PACCage: GigaCage와 PAC(Pointer Authentication Codes)을 결합해 보안 강화

Web Browser Bug trends

시사점 1

memory safety language : RUST

Firefox에서 rust를 사용한다.
Rust는 memory bug 줄이는데에 효과적

MS등 다양한 벤더가 rust 도입을 적극적으로 노력하고 있음

Web Browser Bug trends

시사점 2

Bug Bounty

Google VRP 등 확실한 성과가 있었다.
(자세한 설명은 생략)

Web Browser Bug trends

시사점 3

Bug 평가 시스템

Firefox와 Chromium은 서로 다른 점수를 부여하는 경우가 많음

Firefox는 Chromium의 버그에 대해 58%의 일치율만 보임

CVSS-V3 점수: NVD에서 제공하는 통합된 버그 심각도 평가 기준.

벤더 점수: Firefox와 Chromium이 각각 자체적으로 평가한 버그 심각도 점수.

이 둘도 일치율이 낮음

Web Browser Bug trends

Overwrite Protection

Fast Permission Switch :

- APRR(ARMv8 Permission Register) : ios 11이후 ARMv8에서 도입
  페이지 권한 rwx를 8개의 전용 레지스터에 매핑하여 스레드별 권한을 설정할 수 있다
  JIT 영역은 항상 r-x를 유지하여 필요한(+검증된) 경우에만 unlock func를 통해 w를 얻는다

- Intel MPK(Memory Protection Keys)
  4bit 정수를 각 페이지에 추가하여 액세스 비활성화와 쓰기 비활성화를 제공한다
  JIT영역은 r-x로 유지되며 스레드 단뒤로 쓰기 권한이 제한된다

Web Browser Bug trends

JIT-based code-reuse mitigation

JIT Spray :

공격자가 대량의 JIT code를 메모리에 올려 ASLR/DEP를 bypas한다

Mitigation :

JIT code 크기 제한을 건다

64bit platform 활용으로 ASLR 엔트로피를 높힌다

Web Browser Bug trends

JIT-based code-reuse mitigation

JCRA :

JIT code의 gadget을 재활용하여 exploit 코드를 작성한다

Mitigation :

controlled bytes elimination – 난독화와 랜덤화를 통해 무작위성 부여

internal randomization – 명령어 간 상대적 오프셋 랜덤화, 빈 공간 무작위 할당

Web Browser Bug trends

JIT-based code-reuse mitigation

SDSM(Super Duper Secure Mode) : JIT을 비활성화

Web Browser Bug trends

시사점 4

JIT compiler bug의 파급력과 대응

JIT자체의 복잡성과 비활성화 시 성능 저하 이슈로 대응이 어렵다

성능 최적화와 공격표면 감소의 선택에 놓여있다

Web Browser Bug trends

SOP bypass, UXSS문제 또한 악용 시 브라우저의 파급력이 크다

시사점 5

SOP bypass, UXSS 문제를 site isolation으로 대응

프로세스 수준에서 공격을 무력화한다

제 3자 확장 프로그램의 취약성은 여전히 문제로 남아있다

site isolation은 chrome과 firefox에서만 배포되었다

Web Browser Bug trends

1-day : vendor가 적절히 패치

0-day : report 이후 패치가 진행되지만 공격 연구가 앞선다

Vendor의 방어적 조치 한계 : 기존 공격 벡터를 해결하는 사이에 새로운 공격 벡터가 등장한다

[논문 주장]

능동적 조치가 필요하다 : 새로운 기능이 배포되기 전 잠재적 공격 벡터를 연구하고 보완해야 한다

(이미 하지 않을까 싶긴 하다)

Web Browser mitigation 발전

Sandbox 의 발전 :

공격자는 renderer에서 kernel, system service, browser process(broker) 대상으로 escape해야한다

0-day exploit을 저지하기 위한 좋은 수단

Win32k Lockdown :

Win32k 에서 윈도우 커널 취약점의 상당수가 나왔다. 2012년 ms는 system call disable policy라는 것을 도입하였다. 이를 win32k lockdown이라고 부른다.

edge, chrome, firefox가 채택하였다.

Web Browser mitigation 발전

Hypervisor-based sandboxing

WDAG(windows defender application guard)

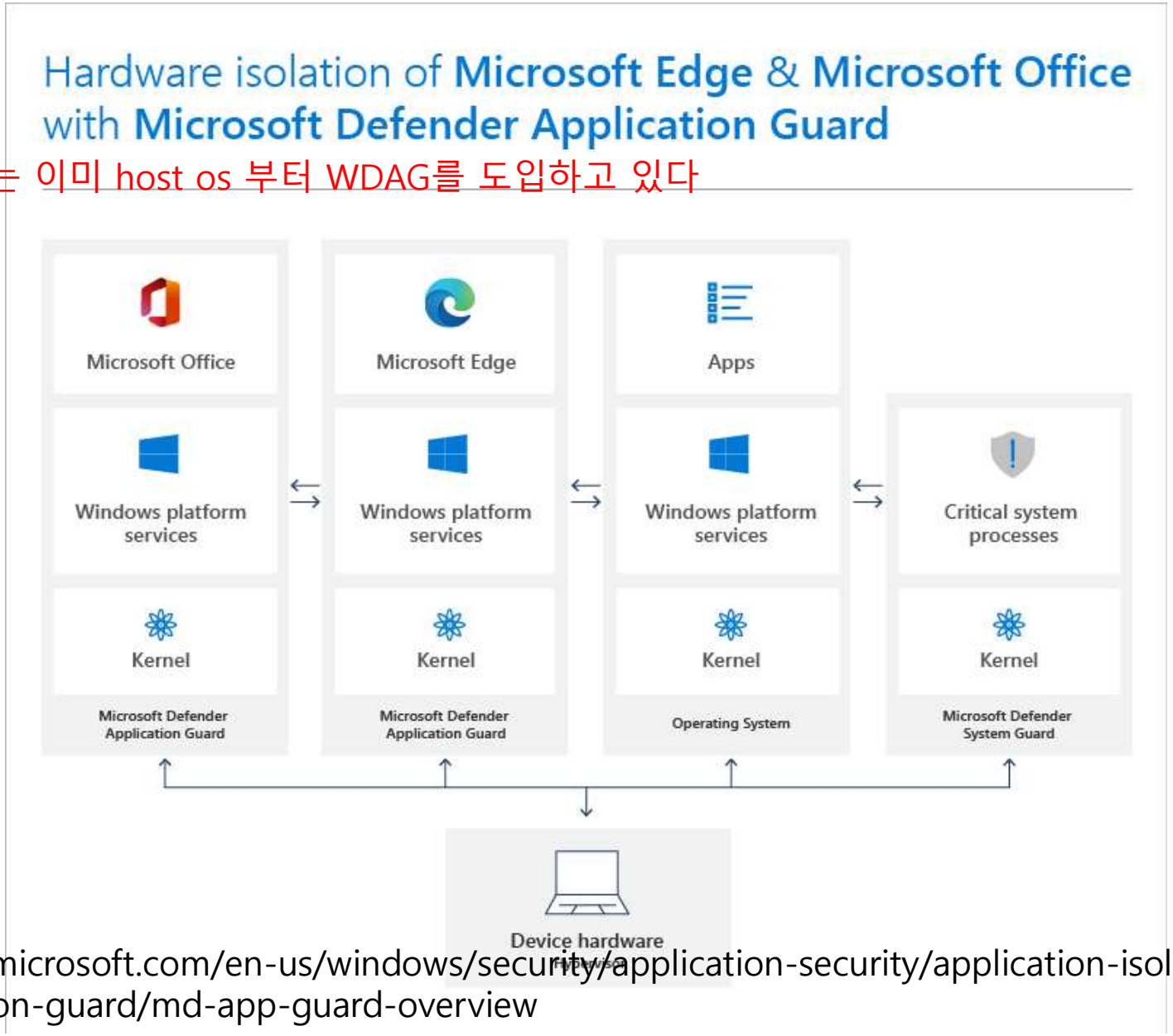
Hyper-v를 사용하여 hardware layer에서 windows instance를 생성

가상머신 escape -> 어렵다

Edge가 도입

*논문 내용은 아님

최신 윈도우(11~)는 이미 host os 부터 WDAG를 도입하고 있다



Ref : https://learn.microsoft.com/en-us/windows/security/application-security/application-isolation/microsoft-defender-application-guard/md-app-guard-overview

Web Browser mitigation 발전

Hardened Allocator

Browser는 성능과 보안 이유로 자체 heap allocator를 쓴다

Isolated heap : 접근 가능한 객체와 다른 객체를 힙에서 분리해둔다
해제된 객체가 다른 레이아웃 객체로 재사용되는 것을 방지(UAF-> type confusion)

Delayed Free : GC(Garbage Collection) 알고리즘을 사용하여 참조가 없는(rc=0) 힙 객체를 제거
Scanning -> 스택 및 힙에서 참조(메모리 포인터 같은 값)를 찾음
한계 : Dangling pointer 가 존재할 수도 있음

Web Browser mitigation 발전

Heap Metadata Protection : 힙 청크에 메타데이터(카나리 같은 랜덤 값)삽입하여 익스플로잇 탐지

Chrome은 partitionalloc에 guard page 추가하여 overflow 시 메타데이터 overwrite 방지

Web Browser mitigation 발전

Control-Flow Integrity

임의로 control flow(실행 흐름)이 넘어가는 것을 방지하는 기술

PC 변조 방지

1. Virtual Function Table 을 보호하는 canary
2. Indirect Branch : 분기되는 대상 주소를 검사하여 allow list에 있는지 확인
3. PAC, MPK(앞에 나온)방법으로 임의 메모리 쓰기 방지

Web Browser mitigation 발전

Side Channel

Meltdown, spectre 같은 캐시, 분기 예측 등의 공격 벡터가 공격에 사용된다
GPU 메모리 액세스 패턴, floating-point timing channel(부동소수 연산 시간 차이),
브라우저 API 악용 등의 공격이 있다

Timer resolution을 낮추는 방법
- 타이머 해상도 낮추기
- 랜덤한 jitter 발생

성능 저하에 영향을 준다

Web Browser mitigation 발전

Resource Sharing Prevention : site isolation, COOP(cross-origin-opener-policy), COEP(cross-origin-embedder-policy)

SharedArrayBuffer의 재활성화

Spectre 및 Meltdown 공격 이후, SharedArrayBuffer는 고해상도 타이머 생성에 악용 가능하다는 이유로 모든 주요 브라우저에서 비활성화됨.

Web Browser mitigation 발전

UAF mitigation : MiraclePtr(V8) (안나오는 줄 알고 앞에서 설명해버림)
논문에는 곧 프로덕션에 적용 될 예정이라고 소개, 현재 적용되어 있음

Torque 언어 도입(V8)
C++를 대체하는 언어로, Domain-Specific Language이다.
LoC가 크게 줄어들면서 error를 감소시켰다.

Web Browser Bug 탐지 Tools

Browser Engine Homogeneity 문제

한 엔진의 취약점이 여러 브라우저 및 애플리케이션에 영향을 미칠 가능성이 높음
Chrome V8 엔진은 Chrome, Edge, Opera, Electron, Node.js 등 다양한 플랫폼에서 사용
WebKit은 iOS에서 강제로 사용되며, WebKit 취약점은 iOS 및 다른 브라우저에 광범위한 영향을 미침

해결 방법 :
브라우저 벤더가 JavaScript 엔진을 공유 라이브러리 형태로 제공.
OTA(Over-the-Air) 업데이트를 통해 패치를 자동으로 배포하여 통합 작업 간소화.

Web Browser Bug 탐지 Tools

Fuzzing과 Static Analysis

(가볍게만 정리)

Mutational Fuzzing : 기존 입력 데이터를 변형하여 테스트

Generational Fuzzing : 새로운 입력 데이터를 생성

+ coverage feedback을 도입 , 대부분 fuzzing으로 버그를 찾음

Static Analysis : 실행없이 분석

Web Browser Bug 탐지 Tools

Fuzzing과 Static Analysis

퍼저와 정적 분석 도구는 많은 버그를 발견했지만, 복잡한 논리적 취약점 탐지는 여전히 수동 분석

필요한 개선:
더 정교한 자동화 도구 개발.
학계와 산업계의 협력을 통한 새로운 탐지 기술 연구.