추측 최적화 중 발생하는 Type confusion 탐지 도구 설계

이창헌[1], 전유석[2]
[1]성공회대학교 정보통신공학전공 학부생
[2]울산과학기술원 컴퓨터공학과 교수
darklight2357@gmail.com, ysjeon@unist.ac.kr

Designing a methodology for detection of type confusion during speculative optimization

Changheon Lee[1], Yuseok Jeon[2]
[1]Dept. of telecommunication engineering, Sungkonghoe University
[2]Dept. of Computer Science Engineering, UNIST

요 약

추측 최적화의 적절하지 못한 구현으로 인해 발생하는 취약점에 대한 연구는 Spectre, Meltdown 등 주로 CPU에서 발생하는 취약점에 초점을 맞추어 활발히 이루어지고 있으나, 브라우저 엔진 및 프로그래밍 언어 구현체 등의 소프트웨어 수준에서 추측 최적화의 적절하지 않은 구현으로 인해 발생하는 Type confusion을 검출하려는 연구는 없었다. 본 연구에서는 소프트웨어 내 추측 최적화의 적절하지 않은 구현으로 인해 발생할 수 있는 취약점으로 Type confusion을 제시하고, 이를 Type check 미흡에 의한 Type confusion, 적합하지 못한 추측에 의한 Type confusion 2개로 분류해 정의하였다. 이 중 적절하지 못한 추측에 의한 Type confusion을 검출할 수 있는 방법론을 설계하였다.

1. 서론

추측 최적화 (Speculative Optimization)는 실행될 가능성이 높은 코드 경로를 예측하여 실제 실행 전 사전에 최적화를 수행하는 기법으로, CPU 및 자바스크립트 엔진과 프로그래밍 언어 구현체에 이르기까지 하드웨어와 소프트웨어에 걸쳐 널리 사용되는 최적화 기법이다. 허나 적절하게 구현되지 않은 추측 최적화는 Spectre [1], Meltdown [2] 등 심각한 보안 취약점의 원인이 될 수 있으며, 기존 연구 또한 SPECTECTOR [3] 와 같이 하드웨어 수준에서의 추측 최적화 과정에서 발생하는 취약점을 효과적으로 검출하고 완화하는 것에 초점을 두고 있다.

추측 최적화는 하드웨어 뿐만 아니라 Google V8, Mozilla SpiderMonkey, Apple JavaScriptCore 등 자바스크립트 엔진과 같은 주요 소프트웨어에서 실행 성능 향상을 위해 이용되며, 적절하게 구현되지 않았을 경우 CVE-2023-3420 [4] 등 Type confusion으로 대표되는 심각한 보안 취약점을 유발할 수 있으나 이를 효과적으로 검출하기 위한 연구는 없었다. 따라서 본 연구는 추측 최적화로 인해 발생하는 Type confusion이 무엇인지 정의하고 이를 검출하는 방법론을 설계하고자 한다.

2. 이론적 배경

추측 최적화는 CPU에서 주로 사용하는 최적화 기법이지만, 자바스크립트 엔진 또한 실행 중 객체의 유형이 변경될 수 있는 자바스크립트의 동적 특성으로 인해 감소하는 성능을 상쇄하기 위하여 인라인 캐시와 히든 클래스, 유형 추론 체계등의 다양한 추측 최적화 기법을 통해 성능을 향상시킨다. 허나 적절하지 못한 구현은 CVE-2023-3420 등 심각한 보안 취약점을 발생시킬 수 있다.

추측 최적화 중 발생하는 Type confusion은 Type check 미흡에 의한 Type confusion과 적합하지 못한 추측에 의한 Type confusion의 두 가지 주요 유형로 분류할 수 있다. 첫 번째는 Type check가 적절하지 못하게 구현된 경우 발생하며, 두 번째는 최적화 과정에서 발생한 잘못된 예측이 올바른 Type check를 우회하면서 발생하는 문제이다. 특히 두 번째 유형의 경우, 자바스크립트 엔진이 잘못된 타입을 예측하고 이에 기반하여 최적화된 코드를 실행함으로써 공격자가 이를 악용할 수 있다.

3. 기존 연구

기존 연구는 BinTyper [5], HexType [6] 등

C/C++ 기반 프로그램에서 발생하는 Type confusion에 대한 검출 방법론 연구와 SPECTECTOR 등 하드웨어 수준의 추측 최적화로 인해 발생하는 Type confusion의 검출과 완화에 대한 연구의 두 가지가 있다.

허나 자바스크립트 엔진에서 발생하는 추측 최적화에 의한 Type confusion의 경우 C/C++ 언어 명세의 Type casting을 위반하는 것이 아닌 자바스트립트 엔진의 잘못된 예측으로 인하여 실행되는 문제라는 점에서 C/C++ 기반 연구는 한계를 지닌다. 또한 SPECTECTOR의 경우 하드웨어 수준에서 수행되는 추측 최적화 취약점 검출 및 완화를 목적한다는 점에서 한계점을 지닌다.

4. 디자인

자바스크립트 엔진 내 추측 최적화의 경우 최초 입력 유형과 추측 조건 및 명세 상 예측 유형이 소스코드 수준에서 정의되어있다.

따라서 추측 최적화를 수행하는 각 함수의 입력 유형 α 및 명세 상 허용되는 예측 유형 β를 1개 쌍으로 하고, 실제 프로그램의 실행 중 추측 최적화 과정을 거치는 객체 A를 추적하여 A가 지니는 α-β 쌍이 소스 코드 상 명세와 상이한 경우, 추측 최적화 과정에서 잘못된 예측으로 인해 Type confusion이 발생했다고 정의할 수 있다.

이에 기반하여 그림 1와 같이 Type을 지니며 추측 최적화의 대상이 되는 객체를 추적하여 그 실행 정보 ɣ를 기록하고, 실행 종료 후 해당 객체가 지니는 α-β과 ɣ 상 에서 허용되는 값 및 추측 최적화를 수행하는 함수 ε에의 입력값과 허용되는 추측 범위 간의 비교를 수행함으로써 적화 과정에서의 잘못된 예측으로 인해 발생하는 Type confusion을 검출할 수 있다.

5. 결론

본 연구는 추측 최적화 과정에서 발생하는 Type confusion을 Type check 미흡에 의한 Type Confusion confusion과 적절하지 못한 추측에 의한 Type confusion의 두 가지 주요 유형으로 분류하고, 이 중 적절하지 못한 추측에 의한 Type confusion에 대한 검출 방법론을 제시하였다.

추후 연구를 통하여 각 함수 별 입력 유형과 추측 조건 및 명세상 허용 유형을 식별하고 검사 구문을 자동으로 삽입하는 과정을 구현할 계획에 있다.

Algorithm 1 Type Confusion Detection

Definition of Object A:
Object A holds the following variables:
$\alpha \leftarrow$ Stated type of input
$\beta \leftarrow$ Stated type of output
$\gamma \leftarrow$ Encoded log including $\alpha_{\text{expected}}, \beta_{\text{expected}}, \delta$
Additional Definitions:
$\alpha_{\text{expected}} \leftarrow$ Expected type of input
$\beta_{\text{expected}} \leftarrow$ Expected type of output
$\delta \leftarrow$ Executed code path including ϵ
$\epsilon \leftarrow$ Speculative optimized code path
$\epsilon_{\text{results}} \leftarrow$ Actual result type for A
$\epsilon_{\text{expected}} \leftarrow$ Stated result type for A
Detect type confusion during speculative optimization:
function A'(Object A)
    Step 1: Decode Information
    Get α, β
    Decode $\gamma \rightarrow \{\alpha_{\text{expected}}, \beta_{\text{expected}}, \Pi\}$
    $\Pi \rightarrow \{\Pi_{\text{results}}, \Pi_{\text{expected}}\}$
    Step 2: Compare Types and Results
    if $\alpha \neq \alpha_{\text{expected}} \vee \beta \neq \beta_{\text{expected}} \vee \Pi_{\text{results}} \neq \Pi_{\text{expected}}$ **then**
        Trigger type confusion detection for Object A
    else
        Log Object A as safe from type confusion
    end if
    Store all type information, results, and execution path γ for Object A
end function

(그림 1) Type confusion 검출 도식

참고문헌

[1] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y. (2018). “Spectre Attacks: Exploiting Speculative Execution”

[2] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M. (2018). “Meltdown: Reading Kernel Memory from User Space”

[3] M. Guarnieri, B. Köpf, J. F. Morales, J. Reineke, and A. Sánchez, "SPECTECTOR: Principled Detection of Speculative Information Flows,"

[4] National Institute of Standards and Technology. (2023). CVE-2023-3420: Vulnerability Summary for CVE-2023-3420. NVD

[5] Kim, D. (2020). BinTyper: Type Confusion Detection for C++ Binaries.

[6] Jeon, Y., Biswas, P., Carr, S. A., Lee, B., & Payer, M. (2017). HexType: Efficient Detection of Type Confusion Errors for C++.

Acknowledgement

이 논문은 2024년도 정부(과학기술정보통신부)의 지원으로 정보통신기획평가원의 지원을 받아 수행된 연구 결과임 (RS-2024-00437306, 메모리 안전 언어의 적용 확대 및 안전 적용을 위한 통합 플랫폼 기술 개발)과 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구 결과임 (RS-2022-II220745, 랜섬웨어 공격 근원지 식별 및 분석 기술 개발).