SecHack365

WebAssemblyを使ったSpectre攻撃の有効性

研究駆動コース　吉澤龍一

なんでこのテーマ？

- Spectre攻撃に興味を持った
  ⇒何かに適用してみたい
- WebAssemblyの名前が挙がる
  ⇒ソフトウェアでサンドボックス化している
- 正規の実行フローを使わない
  Spectreでは破れる可能性があるかも？

SpectreとWebAssemblyについて調べる所からスタート：目標にする

投機実行攻撃 = Spectre 攻撃とは

CPUの機能

キャッシュ：CPUキャッシュは物理メモリへのアクセスよりを高速にアクセスできる。CPUがメモリアクセス命令を実行する際、アクセス先周辺のアドレスをキャッシュにコピーし、一時的にメモリの代わりをすることで効率化している。

投機実行：高速化の仕組み。分岐命令実行時、分岐先が確定するまでにはクロックレベルでは少し時間がかかる。よって、分岐先がわかるまで、分岐しやすい方の命令の実行を先読みで始める。これを投機実行という。

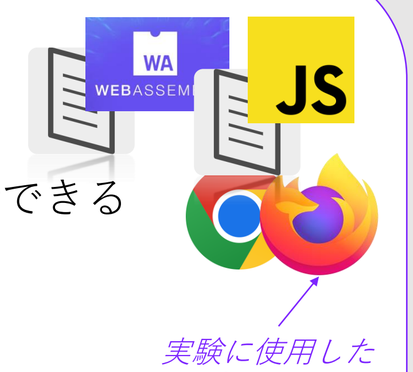
Spectreによる不正な読み取り

攻撃者は任意のコードを書けるとする。任意のアドレスにメモリアクセス可能な命令を分岐先の一つに持つ分岐命令を用意する。分岐条件は「アドレスが確保された領域内であるか」とし、何度か条件が真となるように実行する。その後、通常は不正アクセスとなるアドレスを入力すると、メモリアクセス命令側が「分岐しやすい」と判断され、投機実行される。投機実行によって行われたメモリアクセスによって、不正なアドレスの値がキャッシュされる。これをアクセス時間の差で値を特定する**キャッシュタイミング攻撃**によって特定する。

以上の操作を攻撃者が任意のアドレスに対して行うことで、メモリ空間内の任意の値を読み取ることができる。

WebAssembly(WASM)とは

- 実体はバイナリ形式のファイル
- C/C++, Rustからコンパイルするのが一般的
- ブラウザなどのWASMランタイム上で実行できる
  - ブラウザではJavaScriptから呼び出して使う
  - JavaScriptエンジンにWASMランタイムがある
- 単体で実行できる専用ランタイムもある
  - 専用ランタイムでは普通のバイナリのように実行できる

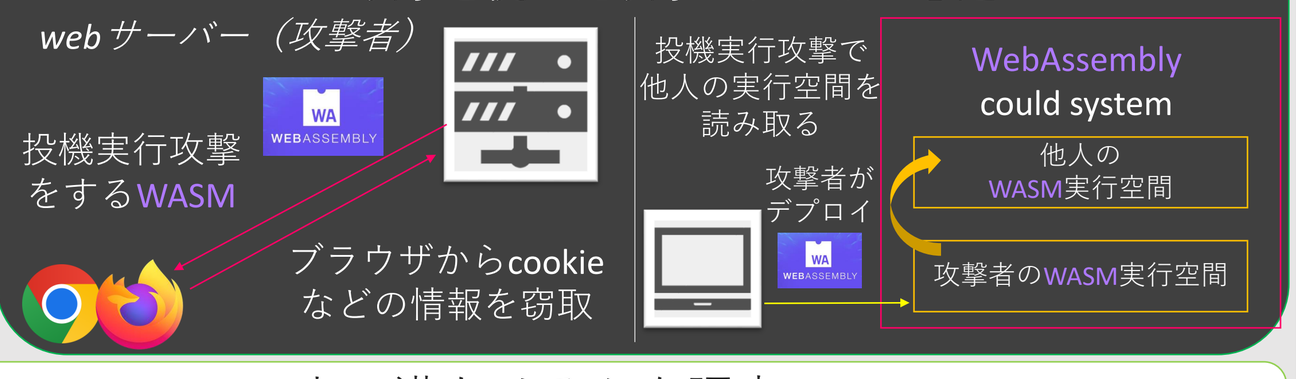


実験に使用したアプリケーション

WasmEdgeRuntime　Wasmer

- 32bitのメモリ空間内で動作（サンドボックス化）

この攻撃を使った攻撃シナリオの想定



攻撃に必要な以下の要件をWebAssembly上で満たせるかを調査

要件1. **キャッシュタイミング攻撃**が可能か　　**要件2**. 投機実行で任意のアドレスがキャッシュできるか

ブラウザ・専用ランタイムそれぞれの実行環境で評価した結果

説明の詳細・ソースコードや失敗理由などの考察はこちら！

ブラウザ

評価はFirefoxで行った。タイマーの精度は設定で上昇させても10μs単位である。キャッシュミス/ヒットのアクセス時間の差は100ns単位といわれているので、アクセス時間の差を倍化させる工夫が必要であった。そこで1000個のキャッシュラインに同じデータを格納し、1000個のキャッシュラインに対して測定をするアプローチをとった。これにより1bitの特定で、本当の値が「0」の場合は93.2%、「1」の場合は81.2%で特定できた。

要件1

専用ランタイム

WASIに準拠した専用ランタイムでは高精度なタイマーが使用できる。評価に利用したWasmerでは、WASIで提供されている関数clock_time_get()の内部でRDTSCP命令を使用していたため、クロックレベルのタイマーが使用できる。WasmerではバックエンドコンパイラとしてCranelift、LLVM、SinglePassが使用できるが、いずれのコンパイラでも97%以上で1byteの特定ができた。

要件2

ブラウザ

投機実行によってキャッシュした値を上記の手法で特定できたら要件2を満たしたとする。しかし、残念ながら実装はできなかった。上記の実装方法はキャッシュラインを1000個使うが、一般的なCPUのL1, L2のキャッシュライン数は合計で4000～5000程度なため、投機実行でキャッシュさせる手順が多く、測定時には読み取り対象がキャッシュに乗っていなかった可能性がある。また、ブラウザにおいてはWebAssemblyから生成された機械語を見ることが難しく、攻撃者にとってはなぜ失敗しているのかが非常にわかりづらいなど、成功させることはほぼ不可能だといえる。

専用ランタイム

通常、WASMバイナリから生成された機械語はメモリアクセス命令に使うアドレスを32bitに変換する（サンドボックス内に収まるように）。しかし攻撃に使うWASMバイナリから手動で無駄な処理を省くと、32bitに変換しない場合が発生した。これはアドレスの値が32bitであることが条件分岐で保障されている場合に、LLVMの最適化によって発生する。WebAssemblyのメモリ空間はランタイムのメモリ空間内の一部を使用しているため、ランタイムのメモリ全体を読み取り対象とすることができる。実際に投機実行でキャッシュさせることは成功しなかったが、SpectreのPoCから生成された機械語と見比べると、成功する可能性は高い。

得られた知見

ブラウザはSpectre対策としてタイマーの精度を低くした経緯があるので、今回はそのうえで攻撃を試みた。結果として、ブラウザではキャッシュタイミング攻撃を行うことは可能であった。しかし、Spectre攻撃を行うには大変すぎるため、世界の攻撃者は割に合わないと感じるだろう。一方専用ランタイムにおいては成功する可能性が高いと考えられ、成功した場合はランタイムのメモリ空間全体が攻撃対象となる。よって、WASI準拠のランタイムをクラウドサービスに採用する場合は、タイマーの精度だけ低くする必要がある。

本活動を通して、WebAssemblyが使える機能は提供されているAPI、WASMバイナリから機械語に変換するバックエンドコンパイラに大きく依存するということが分かった。つまり、使用するWASMランタイムによって、別々の実装になっていることが分かった。

今後の研究方針

Web3でのWebAssembly

ブロックチェーン上では仮想通貨やNFTのやり取りをプログラムのように定義して実行できる、スマートコントラクトという技術がある。一般的にはSolidityという言語で記述され、EVMという実行基盤上で実行される。スマートコントラクトの満たすべき性質はWebAssemblyもカバーしているということで、一部のサービスではEVMの代わり（あるいは並行して）WASM VMを使用している。知見で述べたように、スマートコントラクトに使う機能はAPIで提供されているため、プラットフォームごとに実装が異なっているという現状がある。そこで、WASM VMを採用するプラットフォームに対し、セキュリティ的な問題がないのか調査したい。

NICT National Institute of Information and Communications Technology