THE LINUX FOUNDATION | Research

法律専門家のための WebAssembly

ライセンス・コンプライアンスにおける現在のパラメータを探る

2022年12月

By Armijn Hemel, MSc, Tjaldur Software Governance Solutions

目次

このドキュメントの目的

現在、かなりの人気を集めている技術であるWebAssembly ですが、インターネット上のWebAssemblyドキュメントは、主に開発者を対象としており、その使用方法や開発方法に焦点を当てています。しかし、オープンソース ライセンス コンプライアンスの観点からWebAssemblyを見たドキュメントがないため、その空白を埋めるためにこのドキュメントがあります。

WebAssemblyは、（ほとんどの技術は過去に何らかの形で使われてきているので）技術的レベルでは珍しいものではありませんが、ガバナンスレベルでは、Webのスピードを根本的に変えることができる技術です。ただし、従来とは異なり、主要なブラウザエンジンの開発者が共同で取り組んでいる技術であり、その開発はW3Cの一部です。

技術的には既存の技術と類似しているため、他のオープンソースコンテキストが使用しているライセンス コンプライアンス プロセスをWebAssemblyで変更する必要はありません。しかし、WebAssemblyコードの配布メカニズムは主にネットワーク経由であるため、コンプライアンス情報を探す場所が変わる可能性があります。

このドキュメントは技術文書ではありません。プログラミングのチュートリアルでもありません。多くの技術的詳細を簡略化して説明することで、このドキュメントの目的が複雑になることを避けています。必要に応じて、より詳細な情報を含む（技術）文書にリンクしています。

また、このドキュメントは法律文書ではないため、読者はこの内容から法的結論を導き出すべきではありません。このドキュメントは、WebAssemblyのオープンソース ライセンス コンプライアンスがどのように見え、またいくつかのシナリオでの実施の可能性について議論の出発点として機能することを目的としています。目標は、潜在的なライセンスコンプライアンスの落とし穴を強調し、潜在的な法的問題を提示する可能性のある事実の共通理解を確立するのを助けることです。

WebAssembly はまだ発展途上であるため、このドキュメントはおそらくいつかは古くなります。このドキュメントに特定のシナリオについての記載がないからといって、安全に使用が可能であるとか、ライセンス コンプライアンスの問題が潜在的にないことを保証しているわけでありません。

WebAssembly とは？

WebAssemblyは新しい技術で、主にJavaScriptでは実現できないような高パフォーマンスなWebアプリケーションの作成・展開を想定していますが、それだけにとどまらず、その他の使用事例[1]もみられます。WebAssemblyプログラムはサーバーからダウンロードされ、通常Webブラウザ内で動作する仮想マシンによって実行されます。セキュリティ向上のため、仮想マシンはサンドボックス化されており、サンドボックス内で動作するコードは、明確なアクセス許可がない限り、サンドボックス外のコード、データ、リソースにアクセスすることができません。

JavaScriptとの違いは、WebAssemblyのプログラムがコンパイル済みプログラムとしてダウンロードされ、解析や解釈を必要とせずに、ブラウザ内の仮想マシンで直接実行されることです。これは、JavaScriptプログラムが（通常は最小化された[2]）ソースコード形式でダウンロードされ、ブラウザによって解釈されるのとは対照的です。

WebAssemblyの基本設計では、Webブラウザをクライアントとして想定していますが、WAMR[3]やWasmer[4]などのスタンドアロンのWebAssembly 仮想マシンもあり、Webブラウザのコンテキストの外でWebAssemblyを利用することができます。

WebAssemblyのウェブサイト[5]では、WebAssemblyについて次のように説明されています。

> 「WebAssembly（略称：WASM）は、スタックベースの仮想マシンのためのバイナリ命令形式です。Wasmはプログラミング言語用のポータブルなコンパイルターゲットとして設計されており、クライアントおよびサーバーアプリケーションのWeb上での展開を可能にします。」

Mozilla WebAssemblyのウェブサイトには、[6]次のように書かれています。

> 「WebAssembly は、最新の Web ブラウザで実行できる新しいタイプのコードです。コンパクトなバイナリ形式の低水準のアセムブリ言語的な言語で、ネイティブに近いパフォーマンスで実行でき、C/C++、C#、Rust などの言語にコンパイルターゲットを提供して、Web 上で実行できるようにするものです。また、JavaScriptと並行して動作するように設計されており、両者の連携が可能です。」

ダウンロードしたコンパイル済みコードをブラウザ内のサンドボックス環境で実行するコンセプトは新しいものではなく、過去にも例がいくつかあり、特に Java アプレットや ActiveX が有名です。WebAssembly がこのような過去の例と異なるのは、すべての主要ブラウザエンジンのチームが取り組んでいるオープン標準という点です。つまり、単一のベンダーによって押し付けられた技術であったり、単一のブラウザでしか動作しない技術、あるいはブラウザが特定のクライアントOS上で動作する場合にのみうまく動作する技術（例えば、ActiveXの場合）ではないのです。WebAssemblyは、一つのプログラミング言語だけに限定されず、複数の言語で書かれたプログラムをWebAssemblyバイナリに結合することが可能です。

WebAssembly の元々の設計目的は、JavaScriptでは十分対応できない処理の高速化でしたが、WebAssembly の設計者が思いつかなかったまったく新しいクラスのアプリケーションや使い方を開発者が考え出すこと可能性があります。また、WebAssemblyの仮想マシンを共有ライブラリとして組み込み、あらゆるプログラムを WebAssembly で拡張できるようにする取り組みも行われています。

WebAssembly の形式

WebAssemblyのドキュメントでは、いくつかの形式について説明されています：

1. ソースコード（おそらくいくつかあるプログラミング言語のいずれかで書かれたもの）
2. コンパイラーによって生成され、仮想マシンに読み込まれて実行されるバイナリ形式
3. バイナリ形式のテキスト表現。アセンブリ言語の命令に似ている。

ソースコードとバイナリ形式

ソースコードは、通常プログラマーによって書かれます。プログラマーはEmscripten[7]などのコンパイラーを使い、ソースコードをバイナリ コードにコンパイルします。このコンパイラーは、LLVM 対応の言語のどれでもWebAssemblybinary にコンパイルすることができます。現在、コンパイラーは、C/C++、C#、Rustに対応しており、Python[8]などの他の言語への対応は現在開発中です。

プログラムのバイナリ形式は、Webブラウザ内の仮想マシンが実行するオブジェクトコード命令で構成されており、プログラマーが書いたソースコードからコンパイラーが生成するのが一般的です。

テキスト表現

バイナリ形式のテキスト表現は、通常、ソースコードではありませんが、バイナリ命令のテキスト表現はサンドボックスが実行可能です。これは、バイナリ形式と同じものであり、バイナリ命令をテキスト表現に変換するツールやその逆のツールもあります。

テキスト表現は、スタックを処理する「S式」と呼ばれる命令によるアセンブラ言語のようなものです[9]。例えば、本ドキュメントで後ほど紹介されている例にあるバイナリコードのテキスト表現のごく一部は、次のようになります。

```
(func (;6;) (type 2) (result i32)
      (local i32 i32 i32)
      i32.const 1078
      local.set 0
      i32.const 0
      local.set 1
      local.get 0
      local.get 1
      call 51
      drop
      i32.const 0
      local.set 2
      local.get 2
      return)
```

これらの命令によって、仮想マシンの状態が操作されます。

構文は、例えば ARM プロセッサや MIPS プロセッサのアセンブラコードに似ています。データはスタックに置かれ、スタックからポップされ、操作されます。ちょっと想像力を働かせれば、WebAssembly の命令を、WebAssembly 仮想マシンを CPU とした命令セットと見ることもできます。[10]。

このようなアセンブラのような命令を使って小さなプログラムを直接書くこともできますが、プログラマーが（効率、表現力、再利用性などを考え、）Rust、C、C++などの高水準言語でコードを書く可能性のほうが高くなります。プログラマーはコンパイラーを使い、より高レベルのソースコードをバイナリーコードやそのテキスト表現に変換します。

ほとんどの場合、仮想マシンはバイナリコードを実行しますが、たいていの WebAssembly エンジンは、テキスト表現の命令を 「ジャスト・イン・タイム」でロードし、解釈できます。

WebAssembly の例

コンプライアンスの観点から、コンパイルプロセスがどのように動作するかについて少し理解しておかなければなりません。具体的には、コンパイル時にライセンス情報を含むコメントはどうなるのか、また、ソースコードファイルの情報は配布ファイルに含まれ、必要なコンプライアンス情報がもしあれば、それはコンパイル後のバイナリ形式に含まれるのかといったようなことです。

Mozilla Developer Network[11]の例を見てみましょう。Emscripten (WebAssembly 用コンパイラ) や wabt (コンパイル済みコードのバイナリ表現とテキスト表現を変換するツール) など、必要なコンポーネントやツールがインストールされており、コンパイルは Linux システムで行われると仮定します。この例は非常に単純な C ファイルからなり、hello.c というファイルに保存されています。

```
#include <stdio.h>
int main() {
        printf("Hello World\n");
}
```

これをEmscriptenコンパイラーでコンパイルすると、次のようになります。

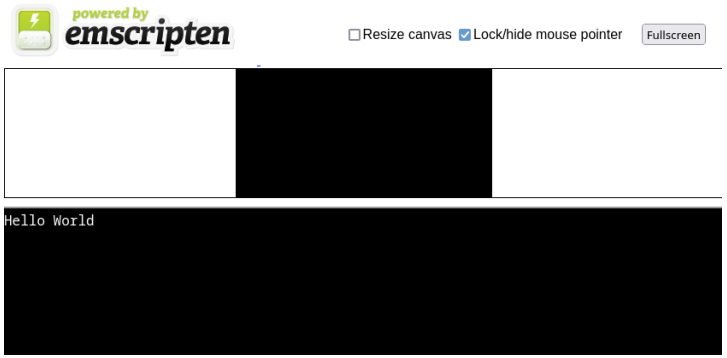
```
$ emcc hello.c -o hello.html
```

すると、以下のようなファイルが生成されます。

- hello.html
- hello.js
- hello.wasm

Emscripten コンパイラーが生成するファイル hello.html と hello.js は、WebAssembly バイナリ ファイル名以外のC ファイルの情報を一切使用しません。ファイル hello.wasm は、ソース コード ファイル "hello.c" からコンパイルされた WebAssembly バイナリ ファイルとなります。

ユーザーが Web サーバーにリソースを要求する Web セッションの間、Web サーバーはバイナリ ファイルをクライアント (通常は Web ブラウザだが、ヘッドレスクライアント[12]でもよい) に送信し、そのファイルは Web ブラウザで実行されているサンドボックス環境で実行されます。このように機能するためには、WebAssembly がウェブブラウザによってサポートされていなければなりません。JavaScriptのコードを使用すれば、実行中のWebAssemblyバイナリと.htmlウェブページとやりとりが可能になります。



WebAssemblyのおもしろい特徴は、命令セットの設計によって、サーバーからクライアントへのストリーミング命令が可能になることです。クライアントは、サーバーが命令を含むすべてのバイナリコードの送信終了を待つ必要がなく、残りのコードを受信しながら処理を開始できます[13]。したがって、WebAssembly のコードを、連続した終わりのない命令の流れとしてクライアントに送り続けるプログラムを作成することができます。このコードには、オープンソースコードからコンパイルされた命令が含む可能性があるため、ライセンス準拠の事例に独自工夫を加えることが可能です。

「wabt」パッケージ[14]のツールを使用すると、バイナリファイルをテキスト形式のアセンブリに似た表現に変換することが可能です。例えば、wasm2wat というツールは、バイナリファイルの内容を WebAssembly 命令のテキスト表現にダンプすることができます。次のコマンド

```
$ wasm2wat hello.asm
```

を実行すると、この単純な Hello World の例に関して、約 5,500 行のアセンブリ言語のような命令が出力されます。

WebAssembly オープンソース ライセンス コンプライアンス

WebAssemblyのオープンソース ライセンス コンプライアンスは、他の状況でのライセンス コンプライアンスとあまり変わりません。実際ほとんど同じです。既存のライセンスは、現在の一般的パターンとわずかに異なるだけの新しい技術的状況に適用されることになります。WebAssemblyは、多くの法律専門家にとって珍しいかもしれない、新しいプログラミングパターンではありますが、類似点も多いので、利用できる専門知識も多く、一からやり直す必要はありません。その代わり、例えば、OpenChainプロジェクト[15]などの最良事例を利用することができます。

ほとんどのオープンソースライセンスには、配布のたびに有効化になる条項があります。コード配布時にやらなければいけないことは、ライセンスによって異なります。ライセンスによっては、(ライセンス テキストを含む) 対応する完全なソースコードを利用可能にすることが要求される場合もあります。例えば、GNUパブリック・ライセンスの場合、ソースコードとバイナリを一緒に配布すること、あるいはソースコードのオファーを書面でユーザーに申し出ることが必要になります。他のライセンスでは、ライセンス テキスト (と時には他のいくつかの情報) をソフトウェア (例: MIT、2条項BSD、Apache 2) に含むことが求められます。上記は単純に分類したものにすぎませんが、要求事項はライセンスによってさまざまです。ですから、どのソフトウェアのライセンスを使用していて、そのライセンスが何を要求しているかを知ることが重要です。

最初の質問は、配布が実施されるどうかです。標準的な WebAssembly の使用例では、その答えは "実施される "でしょう。一般的は配布形態は、WebAssembly のバイナリコードを Web サーバからクライアントに送信する形です。もし配布によって有効になる条項が使用しているオープンソースコードにある場合、ライセンス義務を果たすことが必要です。一見シンプルなライセンスであっても、配布に関連しない他のライセンス義務もあるかもしれません。

例えば、ウェブサイト開発者の間で人気のあるライセンス、MITライセンス[16]の例を見てみましょう (多くのJavaScriptコードがこのライセンス下でライセンスされています) 。これは、オープンソースライセンスの中でも短いものですが、バイナリソフトウェア配布の際に (ライセンスの条件として) ライセンス準拠を要求する内容が含まれます。SPDXのWebサイトには、以下のようなMITライセンステンプレートが掲載されています (本ドキュメント用に強調してあります) 。

> MITライセンス
>
> Copyright (c) <year> <copyright holders>
>
> 以下に定める条件に従い、本ソフトウェアおよび関連文書のファイル (以下「ソフトウェア」) の複製を取得するすべての人に対し、ソフトウェアを無制限に扱うことを無償で許可します。これには、ソフトウェアの複製を使用、複写、変更、結合、掲載、頒布、サブライセンス、および/または販売する権利、およびソフトウェアを提供する相手に同じことを許可する権利も無制限に含まれます。
>
> 上記の著作権表示および本許諾表示を、ソフトウェアのすべての複製または重要な部分に記載するものとします。
>
> **ソフトウェアは「現状のまま」で、明示であるか暗黙であるかを問わず、何らの保証もなく提供されます。ここでいう保証とは、商品性、特定の目的への適合性、および権利非侵害についての保証も含みますが、それに限定されるものではありません。 作者または著作権者は、契約行為、不法行為、またはそれ以外であろうと、ソフトウェアに起因または関連し、あるいはソフトウェアの使用またはその他の扱いによって生じる一切の請求、損害、その他の義務について何らの責任も負わないものとします。**

ライセンスには基本的に、ソフトウェアの複製が何らかの形 (ソースコードあるいはバイナリ) で配布されるたびに、ライセンステキストと著作権表示をソフトウェアと一緒に配布しなければならないと書かれています。MITライセンス下でライセンスされた WebAssembly のコードベースの文脈において、これは、MITライセンステキストと著作権表示をバイナリ.wasmファイルかそのテキスト表現のどちらかと一緒に配布することに

なります。

では、たいていの.wasmファイルが水面下でウェブブラウザに配信され、ユーザーに目に触れないようにしている場合はどのように行うのでしょうか?バイナリコードにライセンス情報が含まれていて、ユーザーがそのライセンス情報に簡単にアクセスできるのであれば、ライセンスの条件を満たしたことになります。その次の質問は、コードをコンパイルするとき、ライセンス情報をバイナリに含むかどうかということです。

試しに、先のファイル "hello.c" を "hello_license.c" にコピーし、上記のMIT ライセンステキストを "hello_license.c" の先頭にコメントとして追加してみました。ライセンステキストが追加されたファイルは、元のファイルよりかなり大きくなっています。通常のソースコード・ファイルがわずか65バイトであるのに対し、ライセンス情報を含むソースコード・ファイルは1217バイトです。

```
$ du -b *.c
65    hello.c
1217    hello_license.c
```

2つ目のファイルの先頭にあるライセンステキストのコンパイラーによる処理を検証するために、2つのCファイルをwasmファイルにコンパイルしました。

```
$ emcc hello.c -o hello.html
$ emcc hello_license.c -o hello_license.html
```

そして、サイズとMD5チェックサムを比較すると:

```
$ du -b *.wasm
12394    hello_license.wasm
12394    hello.wasm
$ md5sum *.wasm
74118fea55ff6490d8fdd9abb201c3b1  hello_license.wasm
74118fea55ff6490d8fdd9abb201c3b1  hello.wasm
```

上記からわかるように、2つのコンパイルコマンドの出力が同一であるため、コンパイラーはコメント (この場合はライセンステキスト) を破棄します。つまり、wasm バイナリにはライセンス情報が含まれず、バイナリのみを提供されたユーザーは、ライセンステキストを抽出したりアクセスしたりすることができません。そのため、ユーザーは別の方法でライセンス情報を取得しなければなりません。

これにはいくつかの方法があります。

ブラウザベースのライセンス情報配布

ユーザーがページ上に読み込まれた WebAssembly コードとやりとり可能な場合、ライセンス条件を満たすための選択肢として、次のようなものが考えられます。

- ユーザーが閲覧している Web ページにライセンス情報を表示する。
- Web ページの可読 HTML ソース コードにライセンス情報を表示する (ユーザーがソースを表示するときに表示できるようにする) 。
- WebAssembly バイナリ ファイルと一緒に送信される JavaScript コードにコメントとしてライセンス情報を追加する。
- ライセンス情報を別のファイルまたは URL に保存し、それを Web ページのソース コードまたは JavaScript ファイルにリンクする。
- SPDX® ISO/IEC 5962:2021 など、ユーザーがライセンステキスト識別のために守る業界標準のライセンス識別子を含める (著作権表示も含める必要がある場合があることに注意してください) 。

テキスト表現コードの配布

別の方法としては、バイナリコードのテキスト表現を生成、配信し、このファイルの先頭にコメント (セミコロン2つで始まる行) で関連情報を追加するやり方もあります。

```
;;  MIT License
;;
;; Copyright (c) <year> <copyright holders>
;;
;; 以下に定める条件に従い、
[...]
```

または、SPDXライセンスIDを使用する[17]:

```
;; Copyright (c) <year> <copyright holders>
;; SPDX-License-Identifier: MIT
```

[...]

テキスト形式が、通常、WebAssemblyプログラムの配布用の既定モードではないことに注意してください。バイナリ形式ではなく、テキスト形式を使用する場合、最初にテキスト形式をバイナリ形式に変換するか、テキスト形式の解釈が必要になり（開発者はあまりやりたくありません）、パフォーマンスが低下する可能性があります。

上記の方法は、ライセンステキストやその他の情報をオリジナルのソースコードから最初に抽出しなければなりません。もちろん、WebAssemblyのバイナリの隣に、すべての法的文章を含む、対応する完全なソースコードを提供するという選択肢もあります。これはライセンステキストの抽出が不要なので、おそらくいくつかのライセンスについては、最初にライセンステキストと著作権表示を抽出して別々に提供するよりも情報を提供しやすいでしょう。たとえば、GNU General Public Licenseバージョン2.0でライセンスされ、ネットワーク上でコピーにアクセスすることで配布されるプログラムでは、GPL-2.0 (第3節)に基づいてソースへの「同等のアクセス」を提供すれば、通常ライセンスの基本的な義務十分に果たします。

まとめると、次の5つの方法になります。

1. ライセンス情報を(インタラクティブな)ウェブサイトに表示する。
2. ウェブページのソースコードや JavaScript ファイルにライセンス情報を含めるか、ウェブページのソースコードや JavaScript ファイル内にリンクされた別のファイルを追加する。
3. テキスト表現を使用し、対応するライセンス情報を追加する。
4. 完全なソースコードを提供し、バイナリコードの隣に置いて利用できるようにする。
5. 上記のいずれかを行うが、特定のライセンステキストを参照する業界標準のライセンス識別子を使用する。

コンプライアンスに必要なステップ

WebAssembly バイナリ配布用のライセンス コンプライアンスについていくつかの解決策をみてみましょう。

第1のシナリオでのコンプライアンスに対する解決策は、あらゆる通知と他の法的文章（例えば、書面によるオファーなど）をWebAssembly バイナリファイルの隣に保存される別ファイルに保存し、以下の説明のように利用できるようにすることです。

第2のシナリオは、すべてのライセンステキストを含む対応するソースコードを提供することです。この2つのシナリオは互いに矛盾するものではありません。すべてのライセンス情報と（該当する場合）書面によるオファーを含むソースコードアーカイブと個別のファイルを提供することができるからです。

両者に共通するのは、まず何かを提供すべきかどうか、そして提供するとしたら何を提供すべきか (ライセンス文のみなのか、あるいは著作権表示や著者表示、対応する完全なソースコード、あるいは書面によるオファーなどのより多くの情報なのか) についてを理解するために、ライセンス義務の確認が必要なことです。

シナリオ 1:
ライセンス通知とその他の法的情報を含む別のファイル

第1のシナリオで提供されるのは、すべてのライセンス通知とその他の法的情報を含む別のファイルです。そのためには次の手順に従ってください：

1. ソースコードからライセンス表示、著作権表示、および作者表示を抽出する（手動で行うか、FOSSology[18]、ScanCode[19]などのツールを使用）。
2. 必要な情報（ライセンステキスト、および必要に応じて文書によるオファー）を含むファイル、またはファイルの集合（たとえば、SPDX形式のSBOMファイル[20]または個別のテキストファイル）を作成し、WebAssembly バイナリの隣に保存する。
3. ユーザーが法的情報を含むファイルを探し出せるようにする。
    a. ファイルへのリンクを HTML または JavaScript のソースコードに含める。
    b. ライセンス情報を含むファイルへのリンクを Web ページに表示する。

「Hello World」の場合、「hello.c」から抽出したライセンス情報を、SBOM ファイルまたは別ファイルで提供し、Webサーバー経由で利用できるようにし、Web サイトでリンクするか、HTML コードで指定します。

シナリオ2：
バイナリファイルの隣に、対応する完全なソースコードを提供

第2のシナリオでは、バイナリファイルの隣に対応する完全なソースコードを提供します。そのメリットは、ライセンス表示、著作権表示、著者表示などがすでにソースコードに含まれているため、ソースコードからの抽出が不要になることです。それ以外については、それほど大きな違いはありません。

1. 必要なソースコード（ソースコードのライセンスによってはビルドスクリプトも含む）でアーカイブを作成し、WebAssembly バイナリの隣に保存する。
2. ユーザーがソースコードのあるファイルを探し出せるようにする。
    a. HTML または JavaScript のソースコードに、ソースコードアーカイブへのリンクを含める。
    b. ソースコードアーカイブへのリンクを Web ページに表示する。

「Hello World」プログラムの場合、これはソースコードファイルをWebサーバーにアップロードし、Webサイト上でリンクするか、HTMLコードで指定することになります。

動的リンクと派生物

いくつかのライセンス（特にコピーレフトライセンス）において、派生物や動的リンクを構成するものを理解することがコンプライアンス活動で繰り返し話題になります。そのため、WebAssembly のコンテキストでも可能かどうか、可能であればどのように可能かを検証する必要があります。もちろん、他のプログラムの派生物であるかどうかは、実際のコードそのものを調べない限りは言えませんし、一概に言うことはできません。

WebAssembly のプログラムは、必ずしも独立したプログラムというわけではなく、WebAssemblyプログラムを他のソフトウェア、主に JavaScript と組み合わせることができるように設計されています。

JavaScript ラッパー

WebAssembly は JavaScript と連係して動作します。WebAssemblyは、パフォーマンス感度が高く、JavaScriptでは十分な速度が得られない操作をスピードアップさせます。JavaScript はウェブページ上でデータを操作でき、難しい作業を WebAssembly のコードに引き継ぎます。この仕組みは、JavaScript とWebAssembly に特有のものではなく、他のプログラミング言語でも、他の言語で書かれたコードを呼び出すことができます。例えば、Python プログラムはC言語で書かれたモジュールで拡張可能で（これは使用する Python インタープリターによるので、すべてのPythonインタープリターで動作したり、うまく動作するとは限らない）、Java には「Java Native Interface」（JNI）という仕様があるなど、さまざまな工夫が見られます。

Mozillaのウェブサイトでは、WebAssembly の JavaScript ラッピングについて、次のように説明しています[21]：

> 「それ以上に、その利点を享受するために利用者は WebAssembly のコードをどのように作成するのか知る必要さえ有りません。 WebAssembly モジュールはウェブ (あるいは Node.js) アプリにインポートすることができ、WebAssembly の機能は JavaScript を経由して他の領域から利用できる状態になります。 JavaScript 製フレームワークでは、大幅なパフォーマンス改善と開発中の新機能をウェブ開発者が容易に利用できるようにするために WebAssembly を用いることができます。」

JavaScript のコードは、ラッパー機構[22,23]を使用して WebAssembly 関数を呼び出すことができます。ラッパー機構はまず、JavaScript から WebAssembly に渡さなければならないデータを、JavaScript で使用されるデータ型から WebAssembly でサポートされるデータ型に変換します。そして、変換されたパラメータを使って仮想マシン内の WebAssembly 関数を呼び出して結果を取得し、その結果をJavaScript のデータ型に変換し、JavaScript 関数を利用できるようにし、さらにその結果を処理できるようにします。

これらのラッパーは、標準的な2つの方法で作成できます。最初の方法は、関数参照[24]を持つテーブルを定義し、WebAssembly バイナリコードをフェッチして読み込み、WebAssembly コード内の関数をテーブルのスロットに割り当てる方法です。2つめの方法は、まず WebAssembly コードをフェッチして読み込み、仮想マシンによって外部にエクスポートされる WebAssembly モジュール[25]の関数にアクセスする方法です。WebAssembly 関数は、WebAssemblyからJavaScriptにエクスポートされ、「エクスポート」と呼ばれます。

WebAssembly のコードは、JavaScript の関数にもアクセスできます。この関数は JavaScript からWebAssembly にインポートされ、「インポート」と呼ばれます。

インポートおよびエクスポートされた関数は、WebAssembly と JavaScript コードの間に結合を生成できます。この結合の強さは、状況によって異なります。最小限の結合（例えば、関数をエクスポートする一般的な WebAssembly モジュール）の場合もあり得るが、WebAssembly コードと JavaScript コードの間にもっと密接な結合がある場合もあります。結合が派生物や他のプログラムに基づく著作物を構成する程度かどうかを判断するには、コードが他のコードとどのように相互作用し、どのメソッドがインポートおよびエクスポートされ、それらがどのように使用されるかを理解することが必要です。

動的リンク

WebAssembly のコードは動的にリンクすることができます。これについては、コンパイラー間でサポートされる標準的なメカニズムはなさそうなので、サポートは各コンパイラに依存することになります。Emscripten コンパイラー[26]のコンテキストでは、動的リンクを可能にする共有モジュールが (いくつかの制限付きで) サポートされています。他の WASM コンパイラーでは、同じアプローチが通用しないかもしれません。Emscripten は、読み込み (メインモジュールで読み込み) 時、またはランタイム (プログラム実行後にサイドモジュールを呼び出し) 時の動的リンクをサポートしています。WebAssembly　のこの分野での標準化についてはまだ初期段階であり、その影響をよく理解するには、さらなる検討が必要です。

WebAssembly からネイティブコードにアクセスする

仮想マシン内外の特定の機能にアクセスしようという提案がされており、WebAssembly System Interface (WASI) [27]はその一例です。WASI の目的は、標準化されたプラットフォーム ニュートラルな API を提供し、仮想マシンが実施したり、公開したりする機能にアクセス可能にすることです。この提案は現在まだ開発中で、すべての仮想マシンに実装されていない可能性があります。WASI インターフェースによって、WebAssembly アプリケーションがホスト機能へアクセスできるようになり、例えば、アプリケーションがローカルホストシステムにファイルを書き込んだり、保存したりすることができます[28]。

WebAssembly の逆コンパイル

利用可能なコードがバイナリ コードまたはそのテキスト表現のみで、ソース コードが必要な場所にない状況があります（例えば監査の場合、ソース コードは通常 WebAssembly のバイナリ コードまたはそのテキスト表現よりも監査しやすくなっています）。ソースコード的なものを得る方法の1つとして、WebAssembly のバイナリコードを逆コンパイルするやり方があります。 これは、wabt パッケージのwasm-decompileツール[29]でできます。これは次のように呼び出すことができます。

```
$ wasm-decompile hello.wasm
```

出力されるのはソースコードに似たファイルですが、必ずしも元のソースコードと似ているわけではありません。元のhello.cファイルは6行ですが、逆コンパイルされたコードは1,800行以上になっています。

これは驚くことではありません。WebAssembly の仮想マシンは、意図的に小さな命令セットでかなりシンプルになっています。その結果、複雑なプログラムを高級言語で書く場合、そのコードを仮想マシンの単純な命令に翻訳する必要があり、WebAssembly には同じ表現力がないため、多くの WebAssembly 命令が生成されることになります。低レベルの命令から高レベルの言語に戻すのは大変な作業ですし、また、元々書かれていたことを再現するのは不可能かもしれません。

WebAssembly のプログラムはC、C++、Rustなどさまざまな言語で書くことができ、それらはすべて同じWebAssembly の命令にコンパイルされます（また、1つのプログラムにまとめることもできます）。元々どの言語で書かれたプログラムなのかの追加情報が残っていない限り、元のプログラムに類似したものを再現することは困難です。

将来、より賢い、あるいはより高度なデコンパイラーが開発される可能性は十分にありますが、それまでは、WebAssembly プログラムの逆コンパイル出力が、元のソースコードの構造や特定の内容を反映していると信頼しない方が賢明です。

アウトバウンド・ライセンシング

これまでのところ、インバウンドライセンスに焦点を当ててきました。つまり、再利用や再配布されるオープンソースコードを含むサードパーティからのコードをどうするかということについてです。みなさんの貢献をどのようにライセンス化するかも、考えるべきことでしょう。選択したライセンスによって、みなさんのコードを再配布したいと考える下流の受信者がより簡単に再配布できるように、みなさんができることもあります。

ライセンスの選択

最初のステップは、ライセンスの選択です。このとき考慮すべきことがいくつかあります。

1. あなたが使っている他のコンポーネントはどのようにライセンスされているのでしょうか？他のコンポーネントのライセンスと互換性のないライセンスをあなたのコードに選択しても意味がありません。ライセンスの互換性をチェックしてください。

2. あるライセンス下でコードをライセンス化する場合、どのような義務があるでしょうか？

選択したライセンスは、コードの下流の受信者がそのコードを再配布したい場合に影響を及ぼします。同じエコシステム内の他の人がどのように彼らのコードをライセンスしているかを理解しておくのはいいことです。というのは、結局このコードが、みなさんのコードに結合される可能性が最も高いコードだからです。多くのエコシステムでは、特定のライセンスが好まれる傾向があります。

標準ライセンスヘッダーを選択する

ライセンスヘッダーは独自のものを作成するのではなく、標準のものを使用することを強くお勧めします。ライセンススキャナーや 法律専門家は、標準ライセンスヘッダーを認識するので、新しいバージョンを書くことによって、無用に物事を複雑にしてしまいます。SPDXライセンスWebサイト[30]には、多くのライセンスに共通する標準ヘッダーのテンプレートが掲載されており、これを使用することができます。

アクセスしやすいフォーマットでライセンス情報を利用可能にする

自分のコードのために、ライセンステキストや対応する完全なソースコードの開示などのライセンス義務を伴うライセンスを選択した場合、ソフトウェア部品表 (SBOM) も提供するか、最低でもSBOMファイルを作成するのに必要な情報を提供すれば、下流の受信者は非常に助かります。SBOMファイルを作成するには、SPDX[31]を参照してください。プロジェクトのライセンス情報伝達のための他の軽量メカニズムは、ABOUT[32]またはREUSE[33] (後者はSPDXメタデータフィールドに基づく) です。

結論

WebAssemblyは比較的新しい技術ですが、新しいからといって他の技術と同じソフトウェアの配布ルールから逃れられるというわけではありません。つまり、ライセンス条項は引き続き適用されます。オープンソースソフトウェアが使用されることは間違いなく、これは WebAssembly ソフトウェアを配布する際にオープンソース ライセンス コンプライアンスが必要であることを意味します。幸いなことに、すでにコミュニティで利用が可能な、多くの専門知識と経験を活用することができます。

本ドキュメントでは、WebAssembly のオープンソース コンプライアンスについて、どの形式が WebAssembly エコシステムで使用されているかや、ライセンステキストや他のライセンス開示文書をどこに掲載する（あるいは掲載しない）か、（Javascriptラッパー、ダイナミックリンクなどの）他のコードとのやりとり、アウトバウンドライセンスなどのいくつかの問題となりうる領域を検討しました。エコシステムに最適なライセンス コンプライアンスのベストプラクティスを生み出すことができるかどうかは、WebAssembly のエコシステム次第なのです。

著者について

Armijn Hemel, MSc, は、オランダ出身のオープンソースライセンスコンプライアンスエンジニアの第一人者。過去 15 年間で、コピーライト・トロールとの法廷闘争から市場投入前のソフトウェア監査まで、数百社のオープンソースコンプライアンスを支援してきました。また、オープンソースのサプライチェーン管理を支援するオープンソースツールの作成にも積極的に取り組んでいます。

謝辞

草稿にコメントと改良を加えていただいた、Luis Villa (Tidelift)、Steve Winslow (Boston Technology Law)、Mike Dolan (The Linux Foundation) に特に感謝します。また、世界中の組織や管轄区域でオープンソースのエコシステムが発展するように WebAssembly の理解を広めるなど、あらゆる形で準拠した技術革新やオープンソース革新を支援する法界のメンバーにも感謝します。

免責事項

本レポートは、「そのまま」で提供されます。The Linux Foundation およびその著者、寄稿者、およびスポンサーは、このレポートに関連する商品性、非侵害、特定目的への適合性、またはタイトルに関する暗黙の保証を含むいかなる保証 (明示、暗黙、またはその他) も明示的に放棄します。The Linux Foundation およびその著者、寄稿者、スポンサーは、契約違反、不法行為 (過失を含む)、その他に基づくかどうか、またそのような損害の可能性を通知されていたかどうかにかかわらず、このレポートに関するあらゆる種類の訴因から生じた利益の損失、またはあらゆる形態の間接的、特別、付随的、あるいは派生的損害について、いかなる第三者に対しても一切責任を負わないものとします。本レポートの作成にあたってスポンサーになったとしても、スポンサーがその調査結果を保証するものではありません。

巻末資料

1 https://webassembly.org/docs/use-cases/

2 “minified JavaScript” は、サーバーからウェブブラウザに送るべきコードをより小さくするために、ほとんどのマークアップ (改行など) を置き換えたJavaScriptの形式のことです。

3 https://github.com/bytecodealliance/wasm-micro-runtime

4 https://github.com/wasmerio/wasmer

5 https://webassembly.org/

6 https://developer.mozilla.org/ja/docs/WebAssembly

7 https://emscripten.org/

8 https://pythondev.readthedocs.io/wasm.html

9 https://developer.mozilla.org/ja/docs/WebAssembly/Understanding_the_text_format

10 https://github.com/WebAssembly/design/blob/main/Rationale.md

11 https://developer.mozilla.org/ja/docs/WebAssembly/C_to_wasm

12 ヘッドレスクライアントは、ユーザーに結果を視覚的に表示しないプログラムです。

13 https://webassembly.github.io/spec/web-api/index.html

14 https://github.com/WebAssembly/wabt

15 https://www.openchainproject.org/

16 https://spdx.org/licenses/MIT.html

17 https://www.linuxfoundation.org/blog/blog/solving-license-compliance-at-the-source-adding-spdx-license-ids

18 https://www.fossology.org/

19 https://github.com/nexB/scancode-toolkit/

20 https://spdx.dev/

21 https://developer.mozilla.org/ja/docs/WebAssembly/Concepts

22 https://developer.mozilla.org/ja/docs/WebAssembly/Exported_functions

23 https://webassembly.org/getting-started/js-api/

24 https://developer.mozilla.org/ja/docs/WebAssembly/JavaScript_interface/Table

25 https://developer.mozilla.org/ja/docs/WebAssembly/JavaScript_interface/Instance/exports

26 https://emscripten.org/docs/compiling/Dynamic-Linking.html

27 https://github.com/WebAssembly/WASI

28 https://wasmbyexample.dev/examples/wasi-hello-world/wasi-hello-world.assemblyscript.en-us.html

29 https://github.com/WebAssembly/wabt/blob/main/docs/decompiler.md

30 https://spdx.org/licenses/

31 https://spdx.dev/

32 https://github.com/nexB/aboutcode-toolkit/blob/develop/docs/source/specification.rst

33 https://reuse.software/

THE LINUX FOUNDATION | Research

2021 年に設立されたLinux Foundation Researchは、拡大するオープン ソースのコラボレーションを検証し、新たな技術トレンド、ベスト プラクティス、オープン ソース プロジェクトのグローバルな影響に関する情報提供を行なっています。プロジェクトのデータベースとネットワークを活用し、定量的かつ定性的な手法でベスト プラクティスを追求することで、Linux Foundation Researchは世界中の組織のためにオープン ソースに関する情報を提供するライブラリーを構築しています。

twitter.com/linuxfoundation

facebook.com/TheLinuxFoundation

linkedin.com/company/the-linux-foundation

youtube.com/user/TheLinuxFoundation

github.com/LF-Engineering

2022年12月

Copyright © 2022 The Linux Foundation

このレポートは、Creative Commons Attribution-NoDerivatives 4.0 International Public License の下でライセンスされています。

著作を引用する場合は、以下のように引用のこと：Armijn Hemel, "WebAssembly (WASM) for legal professionals: Exploring current parameters in license compliance", The Linux Foundation, December, 2022.