5L-3

JavaScriptにおける動的情報流解析のソースコード変換による実装

佐藤寛之 † 小宮常康 ††
電気通信大学†,††

概要

JavaScript コードをブラウザが実行すると，ブラウザが保持するユーザの機密データが漏洩する場合がある．情報流解析と呼ばれるプログラム解析を用いることで，このようなコードを検出することが可能である．既存の JavaScript における情報流解析器の実装 [1] はブラウザの JavaScript エンジンを改造しているので，ユーザに広めることが困難である．そこで本研究では JavaScript コードに対して Austin らの動的情報流解析 [3] を行うコードを埋め込む変換器を実装した．この変換器によって JavaScript コードをソースコード変換することで，オリジナルの JavaScript エンジンを内蔵した既存のブラウザで情報流解析を行うことが可能になる．

1 機密データの漏洩

Web 上の悪意のある JavaScript コードをブラウザで実行させることで，攻撃者はクライアントの機密データを盗むことが可能である．図 1 にその例を示す．このプログラムは `if` 文において `document.cookie` の評価結果が真であれば `true` を返し，そうでなければ `false` を返す JavaScript コードである．この JavaScript コードは機密データである `document.cookie` の値を明示的には漏洩させていないが `document.cookie` に関する情報を漏洩させる．

2 関連研究

プログラム中の機密情報の伝播を解析する手法として，情報流解析がある．情報流解析を用いることで，機密情報を漏洩するプログラムを検出することができる．Austin らの動的情報流解析 [2] は，プログラム中の各データに機密 (H) あるいは公然 (L) を表すラベルを付加し，機密なラベルが付加されたデータの情報が外部に漏洩する可能性があるかを解析する．Austin らが提案した解析手法の一つである Universal Labeling Semantics(ULS) はプログラム中の全ての値に，ラベルを付加する評価規則である．例えば定数 c を評価した場合，その定数 c にラベル pc が付加された値 c^{pc} が返される．pc はプログラムカウンタに付加されるラベルである．また，Austin らは文献 [2] の動的情報流解析手法を JavaScript のサブセットである Featherweight JavaScript に適用した [3].

```
var x;
if(document.cookie){x = false;}
else{x = true;}
if(x){return false;}
else{return true;}
```

図 1. 機密データが漏洩するコード例

Seth Just らは文献 [2] の ULS を JavaScript エンジンに取り入れ，改造 JavaScript エンジンを実装した [1].

3 設計と実装

本研究では JavaScript コードに対して Austin らの動的情報流解析 [3] を行うコードを埋め込む変換器を実装した．変換器が出力する JavaScript コードは，既存のブラウザで実行することができるため，ブラウザを改造する必要がない．変換器の実装言語は Scheme である．本節では変換器が出力する JavaScript コードの設計の一部について述べる．

図 2 に ULS の評価規則の一つである `new` 演算子を評価するにあたって適用される評価規則 (NEW) を示す．Austin らの動的情報流解析では，オブジェクトの本体とポインタの両方にラベリングを行い，割り当てられたオブジェクト $\{\}^{pc}$ へのポインタ p^{pc} を返す．本研究では ULS の評価規則におけるラベリングを JavaScript コードによって実現するためのコード変換手法の設計をした．図 3 に変換器が出力する `new` 演算子のコード例を示す．この変換後のコードは NEW によって生成されるコードや，NEW によって生成されたポインタのラベルの伝播の説明に必要なコードのみを示しており，それに関係しない変換コードは省略している．本研究ではオブジェクトの本体のラベルをオブジェクトのプロパティとして管理し，$\{\}^{pc}$ を `{label:__pc}` と表現するようにした．`__pc` はプログラムカウンタのラベル pc を表現するグローバル変数である．一方，p^{pc} は生成したオブジェクトのポインタとして扱われるが，JavaScript ではポインタに直接ラベルを付加させることは不可能である．そこで図 3 のように変数 `x` に p^{pc}

A Translation-Based Implementation of Dynamic Information Flow Analysis for JavaScript

† Hiroyuki Sato (satoh@spa.is.uec.ac.jp)
†† Tsuneyasu Komiya (komiya@spa.is.uec.ac.jp)
Graduate School of Information Systems, The University of Electro-Communications†,††

$$\frac{p \notin dom(\sigma)}{\sigma, \theta, \texttt{new} \Downarrow_{pc} \sigma[p := \{\}^{pc}], p^{pc}} \quad \text{(NEW)}$$

図 2. 評価規則 NEW

```
//変換前
1:var x=new Object();
2:func(new Object());
↓
//変換後
1:var x=new Object();
2:var x_lab=__pc;
3:x.label=__pc;
4:var v=new Object();
5:var v_lab=__pc;
6:v.label=__pc;
7:tmp_lab=v_lab;
8:func(v);
```

図 3. 変換コード例

が代入される場合，ポインタのラベルを格納する変数 x_lab を用意し，そこへポインタのラベルを格納するように設計した．一方，変換前の 2 行目のコードのように，関数呼び出しの引数として new 演算子が使われる場合は，生成されたオブジェクトのポインタを格納する変数が存在しないので，引数の値とポインタのラベルを保持する変数 (v と v_lab) に NEW の評価結果を格納するように設計した．しかし，関数 func の仮引数は変数 v の値のみ受け取るため，ポインタのラベルが格納された変数 v_lab を呼び出された関数へ伝播させることができない．そこで，本研究では引数のラベルを格納するグローバル変数 tmp_lab を導入し，関数の本体でローカル変数 ⟨ 引数名 ⟩_lab に tmp_lab の値を代入するようにした．図 3 の場合，グローバル変数 tmp_lab に引数の v_lab を代入して関数 func を実行することで，関数呼び出しの引数のポインタのラベルの伝播を表現している．

JavaScript におけるプログラム中の変数は全てオブジェクトのプロパティであり，関数の本体で var 宣言されるローカル変数はスコープオブジェクトのプロパティである．スコープオブジェクトは関数呼び出し時に生成される．このときスコープオブジェクトの本体とポインタに対してもラベリングを行う必要がある．そこで，スコープオブジェクトのポインタのラベルを管理するローカル変数_scope_plab を導入し，スコープオブジェクトの本体のラベルについては，ローカル変数_scope_label を導入した．これらの変数は関数本体の冒頭において__pc の値が代入される．同様に，プログラムのトップレベルで var 宣言されるグローバル変数はグローバルオブジェクトのプロパティである．グローバルオブジェクトもスコープオブジェクトの場合と同様に，グローバル変数 globalObj_plab と globalObj_label を導入し，プログラムの冒頭で宣言するようにした．

Austin らの動的情報流解析では変数 x を評価する場合，x を ⟨*globalObj*⟩.x あるいは ⟨*scopeObj*⟩.x とみなしてグローバルオブジェクトまたはスコープオブジェクトの実体とポインタのラベルを取り出す．本研究では，変換器は変数のレキシカルレベルを参照し，_scope_label と_scope_plab または globalObj_label と globalObj_plab を取り出すことで，Austin らの動的情報流解析と等価なラベリングを行う．

他の評価規則についても同様に，評価規則と等価なラベリングを行う JavaScript コードを出力することで，動的情報流解析を埋め込んだ JavaScript コード変換を実現する．

4 性能評価

JavaScript で記述された Scheme インタプリタ上でフィボナッチ関数を実行し，性能評価を行った．測定は 2.66GHz の Intel Core i7 およびメモリ 4GB(DDR3) を搭載した MacBookPro で行った．その結果，変換したプログラムは実行時間は約 17 倍の実行時間を要した．

5 結論

本研究では JavaScript コードを Austin の動的情報流解析を埋め込んだ JavaScript コードに変換するソースコード変換器を実装した．変換器によって出力された JavaScript コードはオリジナルの JavaScript エンジンで実行可能である．今後の課題として，文献 [2] において 50%オーバヘッドの削減を示唆する sparse labeling semantics の実装に取り組み，実行オーバヘッドを削減することがあげられる．

参考文献

[1] Seth Just et al. Information flow analysis for javascript, In *Proceedings of the 1st ACM SIGPLAN international workshop on Programming language and systems technologies for internet clients,* pp.9–17, 2011

[2] Thomas H. Austin et al. Efficient purely-dynamic information flow analysis, In *Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security,* pp.113–124, 2009

[3] Thomas H. Austin et al. Dynamic Information Flow Analysis for Featherweight JavaScript. In *Technical Report UCSC-SOE-11-19*, September 22, 2011