УДК 004.735

Кучеренко О. А., магістрант
(Національний технічний університет України «КПІ». +380 (93) 773 33 91, alexa.kucherenko@gmail.com)

АНАЛІЗ ТА РОЗРОБКА МЕТОДІВ ДЕОБФУСКАЦІЇ СКРИПТІВ ДЛЯ ВИЯВЛЕННЯ ЗАГРОЗ ІНФОРМАЦІЙНОЇ СТІЙКОСТІ КОМП'ЮТЕРА

Кучеренко О. А. Аналіз та розробка методів деобфускації скриптів для виявлення загроз інформаційної стійкості комп'ютера. В роботі досліджуються проблемні питання захисту інформаційних систем та пропонуються методи вирішення цих проблем. Метою є покращення систем захисту комп'ютерної інформації від скриптових вірусів, які потрапляють до комп'ютера з мережі інтернет, а саме при перегляді WEB-сторінок із вбудованим вірусним програмним забезпеченням. Описані принципи створення та побудовані алгоритми протидії обфусцированим скриптовим вірусам.

Ключові слова: обфускація, деобфускація, розшифровування, WEB-сторінка, JavaScript, скриптовий вірус

Кучеренко А. А. Анализ и разработка методов деобфускации скриптов для выявления угроз информационной стойкости компьютера. В работе исследуются проблемные вопросы защиты информационных систем и предлагаются методы решения этих проблем. Целью является улучшение систем защиты компьютерной информации от скриптових вирусов, которые попадают к компьютеру из сети интернет, а именно при пересмотре WEB-страниц со встроенным вирусным программным обеспечением. Описаны принципы создания и построены алгоритмы противодействия обфусцированим скриптовим вирусам.

Ключевые слова: обфускация, деобфускация, расшифровка, WEB-страница, JavaScript, скриптовий вирус

Kucherenko O. A. Analysis and development of deobfuscation scripts methods for the exposure of threats informative firmness of computer. The problem questions of defence of the informative systems are in-process probed and the methods of decision of these problems are offered. A purpose is an improvement of the systems of defence of computer information from script viruses which get to the computer from a network the internet, namely at the revision of WEB-pages with built-in viral software. Principles of creation are described and the algorithms of counteraction obfuscation script viruses are built.

Keywords: obfuscation, deobfuscation, decoding, WEB-page, JavaScript, script virus

Вступ і постановка задачі. З кожним днем зростає кількість інформаційних комп'ютерних систем, які потребують захисту, надійності в обробці й використанні даних. Разом із цим зростає кількість різновидів вірусів та вірусних кодів, які можуть негативно вплинути або взагалі внести критичні зміни в роботу будь-якої інформаційної системи. В процесі виконання наукового дослідження знайдені принципово нові рішення по захисту інформаційної системи від зашифрованих скриптових комп'ютерних вірусів.

Гармонізація понятійної бази. В зв'язку з тим, що проблеми протидії зашифрованим скриптовим вірусам є досить новою, в науково-практичних джерелах є деяка розбіжність в термінології та поняттях. Тому на першому етапі досліджень проведена гармонізація понятійної бази.

Гармонізація базувалась на наступних принципах:

– використані поняття та терміни повинні відповідати загальновизнаним та не протирічити між собою;

– використані поняття та терміни повинні відповідати меті дослідження.

В даній статті особлива увага приділяється обфускації програмного коду та веб-додатків написаних на скриптовій мові JavaScript.

Обфускація – це приведення виконуваного програмного коду до виду, який зберігає його функціональність, але ускладнює аналіз, розуміння алгоритму роботи і модифікації при декомпіляції [1]. Іншими словами це "заплутування" коду. "Заплутування" коду може здійснюватися на рівні алгоритму, сирцевого тексту або асемблерного тексту. Існують спеціальні програми, що виробляють обфускацію, їх називають обфускаторами. Є деяка кількість шляхів обфускації, яка може реорганізовувати програмний код в автоматичному режимі або в ручному.

Цілі обфускації: *ускладнення* декомпіляції та вивчення програм з метою виявлення функціональності; *ускладнення* декомпіляції програм з метою запобігання обходу систем перевірки ліцензій; *порушення* авторських прав програмістів і приховування авторства; *демонстрація* неочевидних можливостей мови і кваліфікації програміста.

Останнім часом, в інтернет середовищі значно поширилась тенденція написання "шкідливого" коду і прихованість його за допомогою методів обфускації. Такі вірусні програми та скрипти легко обходять існуючі антивірусні засоби, тому що виявити "шкідливий" функціонал досить складно. Щоб дослідити такий код треба проводити деобфускацію. В деяких випадках її треба робити в декілька кроків, тому що код може бути обфускованим декілька разів. Тобто, вже обфускований код піддається новій обфускації і т.д.

Деобфускація – це процес, приведення обфускованого коду до виду, який дозволить вивчати та аналізувати його функціонал. Зараз немає систем, які б змогли проаналізувати будь-який обфускований код, але існують деобфускатори (програми, які надають змогу проаналізувати функціонал або привести код до зрозумілого виду), які "заточені" тільки під певний різновид обфускації. Метою дослідження є можливість аналізувати будь-який обфускований код, незважаючи, якими засобами та методами він був обфускований.

1. Методи шифрування (обфускації) скриптової мови JavaScript. Існує дуже велика кількість методів обфускації, яких із кожним днем стає дедалі більше. Ми спробували їх розділити на три категорії.

1.1. Обфускація мінімізаторами. Існують готові програмні засоби, які надають змогу в автоматичному режим зробити "зтискання" коду. Таким видом обфускації користуються в більшій мірі виробники програмного забезпечення. Код, який отримаємо на виході, можна зашифрувати ще одним методом, які будуть розглянуті пізніше в нашій роботі.

Серед загальновідомих мінімізаторів [2] можна виділити JS Packer, JSmin, YUI Compressor, Closure compiler та це далеко не повний їх список.

Написаний код на скриптовій мові JavaScipt виду:

```
function MyClass(){
   this.foo = function(argument1, argument2){
      var addedArgs = parseInt(argument1)+parseInt(argument2);
      return addedArgs;
   }
   var anonymousInnerFunction = function(){
      // do stuff here! }
```

за допомогою мінімізаторів приймає або такий вигляд:

```
function MyClass(){this.foo=function(c,b){var d=parseInt(c)+parseInt(b);return d};var a=function(){}};
```

або такий:

```
var _0xd799=["\x66\x6F\x6F"];function MyClass(){this[_0xd799[0]]=function
(_0xefcax2,_0xefcax3){var _0xefcax4=parseInt(_0xefcax2)+parseInt(_0xefcax3);return _0xefcax4;} ;var
_0xefcax5=function (){} ;} ;
```

або такий:

```
eval(function(p,a,c,k,e,d){e=function(c){return c};if(!''.replace(/^/,String)){while(c--
){d[c]=k[c]||c}k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1};while(c--
){if(k[c]){p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('4 0="3 5!";9
2(1){6(1+"\\7"+0)}2("8");',10,10,'a|msg|MsgBox|Hello|var|World|alert|n|OK|function'.split('|'),0,{}))
```

З першого погляду отриманий код важко проаналізувати та все ж таки не існує нічого неможливого. Подібну обфускацію можно провести і в ручному режимі, що, можливо, навість ускладнить його деобфускацію.

1.2. Методи Braifuck. Такий підхід перетворює код до невпізнання [3]. Навіть незрозуміло, що перед нами виконуваний JavaScript скрипт.

Мізерний скрипт

```
alert(0)
```

можна перетворити в такий вигляд:

```
([][(![]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+(![]+[]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[
+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]][([][(![]+[])[+
[]]+(![]+[]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]
]]+[])[!+[]+!+[]+!+[]]+(![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[!+[]+!+[]]()(([]+[])[+!+[]]+(![]+[])[!+
[]+!+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]))(+[]);
```

або в такий:

ﾟωﾟﾉ= / ｀m´) ﾉ ~┻━┻ //*´∇｀*/ ['_']; o=(ﾟｰﾟ) =_=3; c=(ﾟΘﾟ) =(ﾟｰﾟ)-(ﾟｰﾟ); (ﾟДﾟ) =(ﾟΘﾟ)= (o^_^o)/ (o^_^o);(ﾟДﾟ)={ﾟΘﾟ: '_' ,ﾟωﾟﾉ : ((ﾟωﾟﾉ==3) +'_') [ﾟΘﾟ] ,ﾟｰﾟﾉ :(ﾟωﾟﾉ+ '_')[o^_^o -(ﾟΘﾟ)] ,ﾟДﾟﾉ:((ﾟｰﾟ ==3) +'_')[ﾟｰﾟ] }; (ﾟДﾟ) [ﾟΘﾟ] =((ﾟωﾟﾉ==3) +'_') [c^_^o];(ﾟДﾟ) ['c'] = ((ﾟДﾟ)+'_') [(ﾟｰﾟ)+(ﾟｰﾟ)-(ﾟΘﾟ)];(ﾟДﾟ) ['o'] = ((ﾟДﾟ)+'_') [ﾟΘﾟ];(ﾟoﾟ)=(ﾟДﾟ) ['c']+(ﾟДﾟ) ['o']+(ﾟωﾟﾉ +'_')[ﾟΘﾟ]+ ((ﾟωﾟﾉ==3) +'_') [ﾟｰﾟ] + ((ﾟДﾟ) +'_') [(ﾟｰﾟ)+(ﾟｰﾟ)]+ ((ﾟｰﾟ==3) +'_') [ﾟΘﾟ]+((ﾟｰﾟ==3) +'_') [(ﾟｰﾟ) - (ﾟΘﾟ)]+(ﾟДﾟ) ['c']+((ﾟДﾟ)+'_') [(ﾟｰﾟ)+(ﾟｰﾟ)]+ (ﾟДﾟ) ['o']+((ﾟｰﾟ==3) +'_') [ﾟΘﾟ];(ﾟДﾟ) ['_'] =(o^_^o) [ﾟoﾟ] [ﾟoﾟ];(ﾟεﾟ)=((ﾟｰﾟ==3) +'_') [ﾟΘﾟ]+ (ﾟДﾟ) .ﾟДﾟﾉ+((ﾟДﾟ)+'_') [(ﾟｰﾟ) + (ﾟｰﾟ)]+((ﾟｰﾟ==3) +'_') [o^_^o -ﾟΘﾟ]+((ﾟｰﾟ==3) +'_') [ﾟΘ ﾟ]+ (ﾟωﾟﾉ +'_') [ﾟΘﾟ]; (ﾟｰﾟ)+=(ﾟΘﾟ); (ﾟДﾟ)[ﾟεﾟ]='\\'; (ﾟДﾟ).ﾟΘﾟﾉ=(ﾟДﾟ+ ﾟｰﾟ)[o^_^o -(ﾟΘﾟ)];(oﾟｰﾟo)=(ﾟωﾟﾉ +'_')[c^_^o];(ﾟДﾟ) [ﾟoﾟ]='\"';(ﾟДﾟ) ['_'] ((ﾟДﾟ) ['_'] (ﾟεﾟ+(ﾟДﾟ)[ﾟoﾟ]+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ (ﾟΘﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟｰﾟ)+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ (ﾟｰﾟ)+ ((ﾟｰﾟ) + (ﾟΘﾟ))+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((o^_^o) +(o^_^o))+ ((o^_^o) - (ﾟΘﾟ))+ (ﾟДﾟ)[ﾟεﾟ]+(ﾟΘﾟ)+ ((o^_^o) +(o^_^o))+ (ﾟｰﾟ)+ (ﾟ Дﾟ)[ﾟεﾟ]+((ﾟｰﾟ) + (ﾟΘﾟ))+ (c^_^o)+ (ﾟДﾟ)[ﾟεﾟ]+((o^_^o) +(o^_^o))+ (c^_^o)+ (ﾟДﾟ)[ﾟεﾟ]+((ﾟｰﾟ) + (ﾟ Θﾟ))+ (ﾟΘﾟ)+ (ﾟДﾟ)[ﾟoﾟ]) (ﾟΘﾟ)) ('_'),

де взагалі використовуються символи, яких немає навіть на розкладці клавіатури.

Хоча з першого погляду все виглядає досить страшно та здебільшого до такого виду можна привести в автоматичному режимі. Існує деяка кількість Braifuck-подібних конверторів для JavaScript.

До даного виду обфускації також можна віднести написання коду на езотеричних мовах програмування таких як Whitespace, JAPH та інших.

1.3. Метод запечатування. В першому способі на виході ми отримували код схожий на JavaScirpt, у другому зовсім не схожий, а метод запечатування його робить взагалі прихованим.

Обфускований код буде складатися з двох частин: *видима* частина (може використовуватися будь-який із вищеперерахованих спосіб обфускації) і *прихована* частина.

Реалізовується це таким чином. "Шкідливий код", який ми хочемо сховати, перетворюється на рядок, який складається зі знаків табуляції (біт 1) та пробілів (бит 0). Як результат, ми отримуємо в багато разів більше коду, ніж у нас було. Видима частина буде декодувати приховану частину та її виконувати: декодуватиме біти в числа, а числа перетворить в символ String.fromCharCode(), а далі виконає функцію eval. Приклад:

```
decodeAndEval(document.getElementById("evilCode").innerHTML);
<div id="evilCode">
</div>
```

До цього методу обфускації коду можна також віднести пакування скрипта JavaScript в CSS [4]. На сторінці користувача буде відображатися картинка, яка може бути навіть фоновою, але при виконанні коду нашкодить вашому комп'ютеру. Такий спосіб шифровки реалізовується таким чином: відбувається зтискання коду та з'являється можливість реалізації інтелектуальних систем відновлення зображень винятково на растровій мові. Вказуючи просторове положення такого зображення-коду, ми можемо формувати дуже інтересні структуровані по принципу нарізання (tilling) бібліотеки практично безграничного розміру, високо-оптимізовані по трафіку та часу доступу до конкретної ділянки коду (Рис. 1).

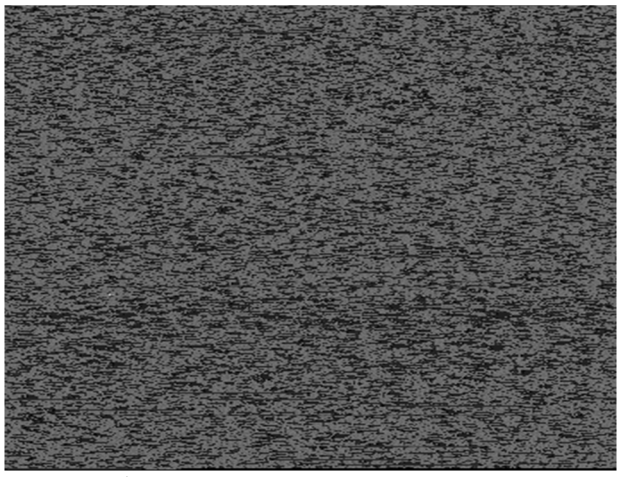

Рис.1. Приклад запечатаного скрипта в зображення

На Рис. 2 приведена узагальнена структурна схема шляхів обфускації.

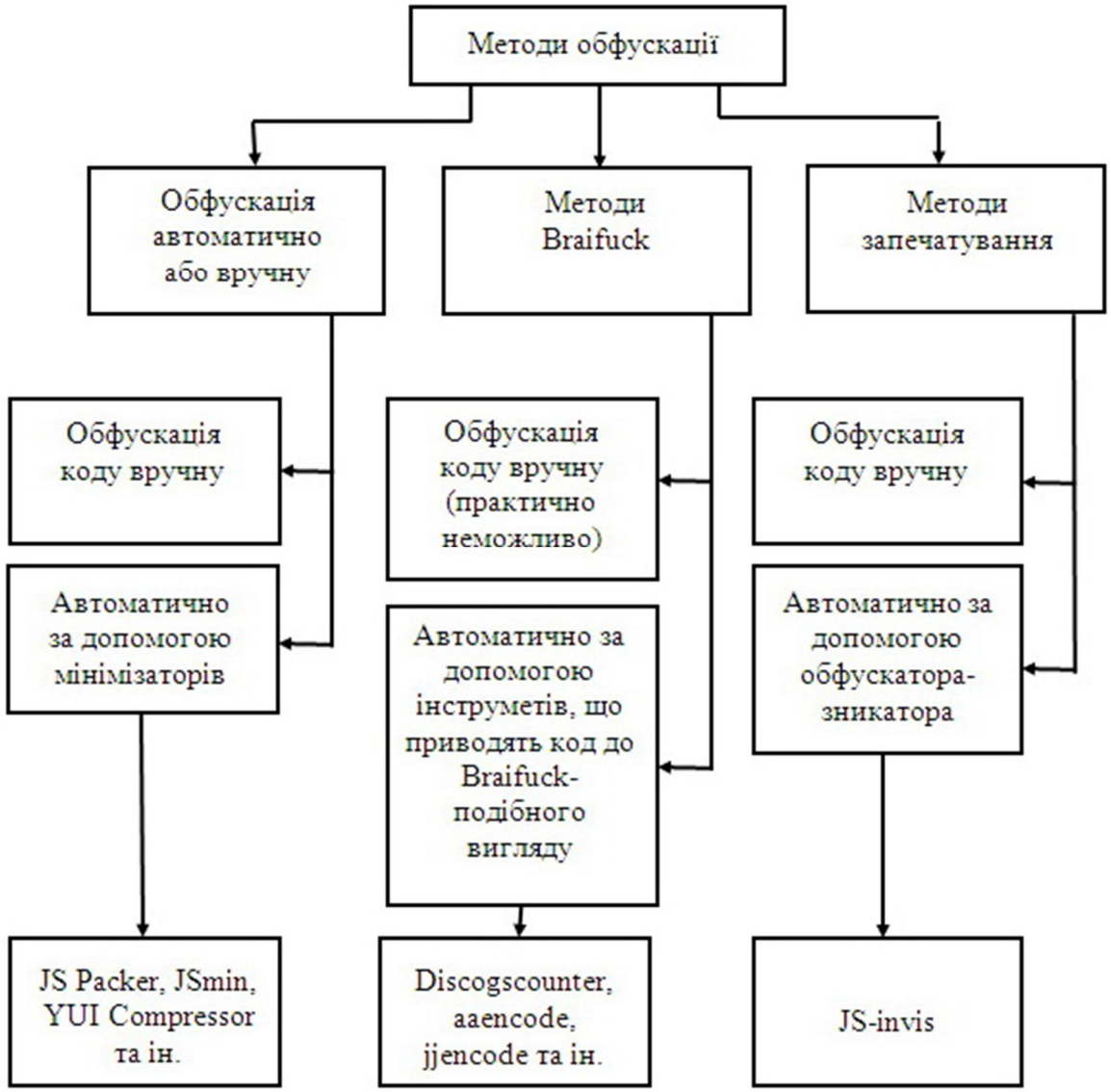

Рис. 2. Узагальнена структурна схема шляхів обфускаціїї

2. Розробка методів деобфускації JavaScript. Аналіз літературних джерел не виявив завершеного алгоритму деобфускації зашифрованого коду JavaScript. Знайдено тільки характерні приклади деобфускації. Тому на основі ідеї розшифровування скриптів та деяких запропонованих шляхів аналізу обфускованого коду запропонуємо деякі методи деобфускації JavaScript.

2. 1. Розшифровування JavaScript, який було обфусковано за допомогою мінімізатора.

1) Визначаємо обфускований код JavaScript. Якщо ми хочемо дізнатися, чи є на певній WEB-сторінці "шкідливий" код, нам треба її просканувати за допомогою інструментів .net fraimwork та виявити наявність JavaScript скриптів. При обфускації код може прийняти зовсім незрозумілий вигляд і втратити характерні ознаки, та це не стане завадою для його коректного виконання на комп'ютері і, як результат, захований вірус може нанести чималу шкоду електронному пристрою. Для визначення зашифрованого скрипта будемо використовувати типові приклади та результати роботи обфускаторів мінімізаторів. Маємо певний набір ключових слів та функцій, які вказують на те, що деякий фрагмент або фрагменти сторінки – це обфускований JavaScript.

2) Починаємо розбір зашифрованого коду. Завантажуємо підозрілий код у рядкову змінну. Спочатку ініціалізуємо перший знайдений зашифрований фрагмент коду WEB-сторінки.

3) Аналізуємо підозрілий скрипт. Оскільки ми визначили, що шифрування відбувалося методом мінімізаторів, то для таких випадків характерним є упорядковування коду. Додаємо символ нового рядка після кожної крапки з комою.

4) Отримуємо імена функцій JavaScript. Для цього окремо скануємо кожний отриманий рядок. Об'явлені змінні в обфускованому коді передаємо до функції alert() для того, щоб отримати реальні імена функцій JavaScript.

5) Аналізуємо отриманий функціонал на безпечність використання. Результатом попередніх дій стало виявлення функцій, які при певному наборі шляхом аналізу можуть дати відповідь на питання чи зашкодить виконуючий скрипт роботі комп'ютерної системи.

6) Якщо на сторінці виявлено декілька зашифрованих скриптів, то результати першого просканованого коду зберігаємо у вихідний файл та переходимо знову до пункту 2, де розпочинаємо аналіз наступного підозрілого фрагменту.

7) Формування висновку про безпечність посилання. Тільки після того, як будуть розшифровані всі підозрілі скрипти, ми зможемо зробити висновок, наскільки є надійним певне посилання.

Для даного методу деобфускації розроблено наступний алгоритм, зображенийна Рис. 3.

2.2. Розшифровування JavaScript, який було обфуцировано за допомогою BraiFuck-подібного методу.

1) Визначаємо обфускований код JavaScript. Для того щоб визначити прихований JavaScript код BraiFuck-подібного вигляду треба використовувати xml-файл, який матиме в собі підбірку типових Braifuck-подібних скриптів та окремих символів.

2) Розпочинаємо розбір зашифрованого коду. Завантажуємо підозрілий фрагмент у рядкову змінну.

3) Розшифровуємо код. Запускаємо цикл FOR по всій довжині вмісту змінної.

4) За одну ітерацію оброблятимемо один символ.

5) Дістаємо значення зашифрованого тексту-коду. Як усім відомо, повернути значення певного Braifuck-подібного зашифрованого скрипта просто, слід лише використати правильний деобфускатор, але для цього треба виконати глибокий аналіз. Тому пропонуємо вміст скрипту отримати універсальним способом. Визначаємо ASCII-код символа:

– Якщо код має значення між 36 і 61 (не включно), то додаємо до нього 25, а символ, відповідний результуючому коду, зберігаємо в нову змінну.

– Якщо код має значення між 61 і 86 (не включно), то віднімаємо від нього 25, а символ, відповідний результуючому коду, зберігаємо у ту ж таки нову змінну.

6) Декодування. Після того, як ми перейшли від Braifuck-подібного вигляду до більш зрозумілого нашій системі розпізнавання вірусів виду, отримане треба декодувати. Для цього нам треба реалізувати виклик функції eval() для запуску результату декодування.

7) Аналізуємо функціонал на безпечність використання. Можемо мати один із двох результатів:

– Якщо результатом деобфускації є код, в якому можна проаналізувати функціонал, тобто ключові слова, які визивають деякі функції, то переходимо до оцінки безпечності такого коду.

– Якщо результатом деобфускації є код, в якому проаналізувати функціонал не вдається, то треба перейти до попереднього методу деобфускації JavaScript, який було розглянуто в 2.1. Такий підхід демонструє, що обфускація була проведена у декілька кроків, де спочатку скористалися методом мінімізації, а потім методом приведення коду до Braifuck-подібного вигляду.

Для даного методу деобфускації розроблено алгоритм, який зображено на Рис. 4.

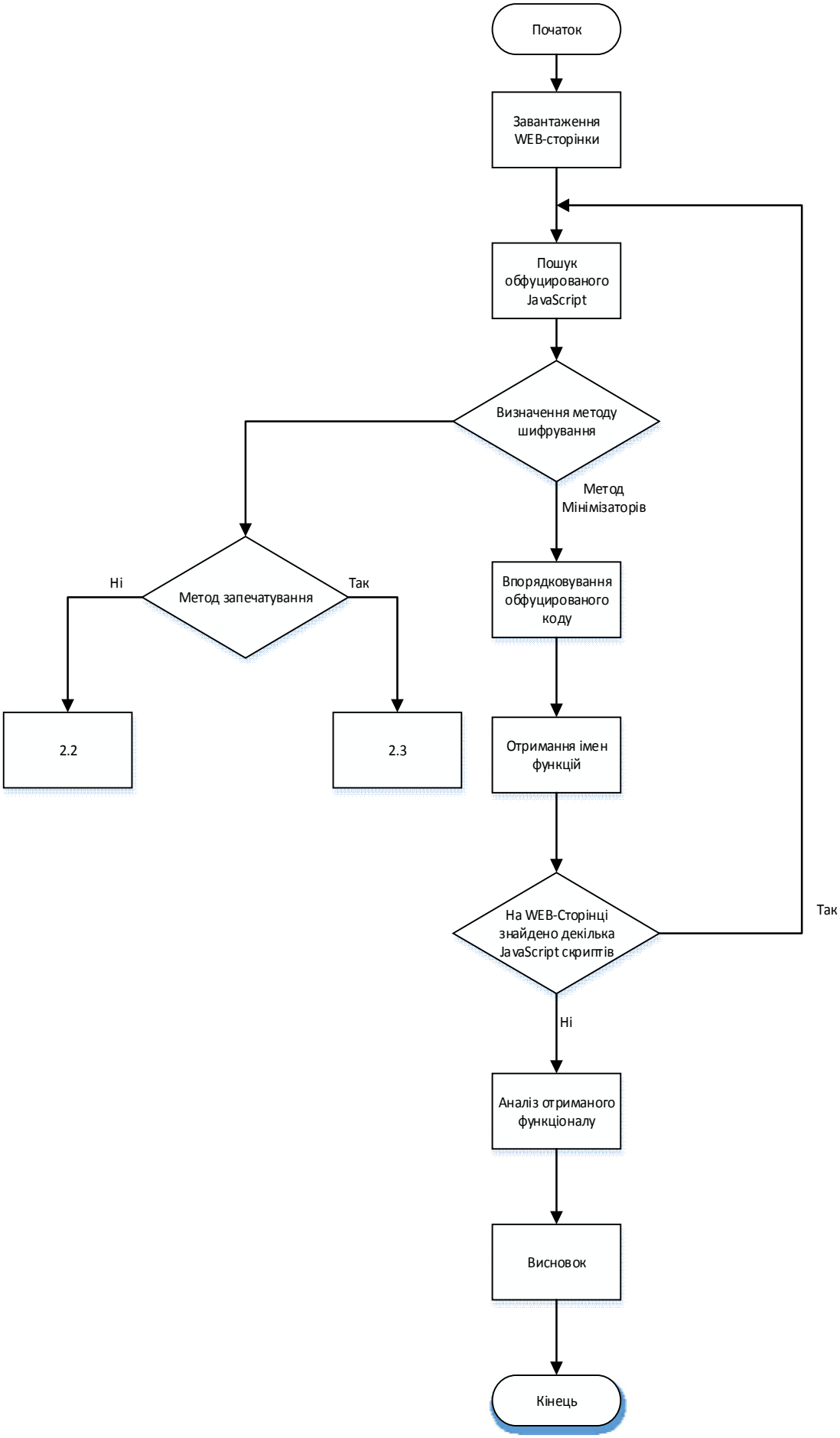

Рис. 3. Алгоритм деобфускаціїї зашифрованих скриптів мінімізаторами

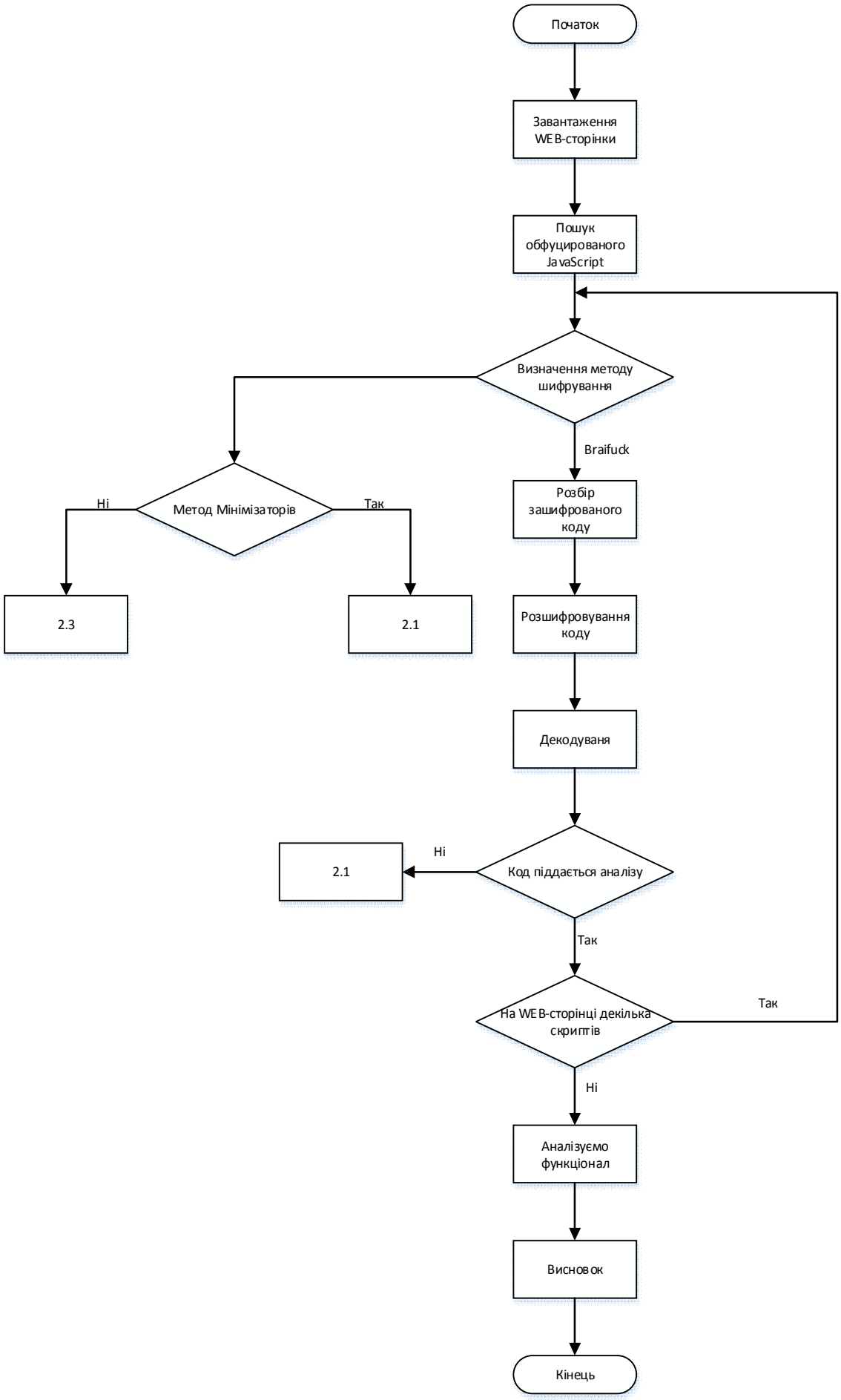

Рис. 4. Алгоритм деобфускаціїї зашифрованих скриптів Braifuck-подібного виду

2.3. Розшифровування JavaScript, який було обфусковано за допомогою методу запечатування. Алгоритм даного методу деобфускації зображено на Рис. 5.

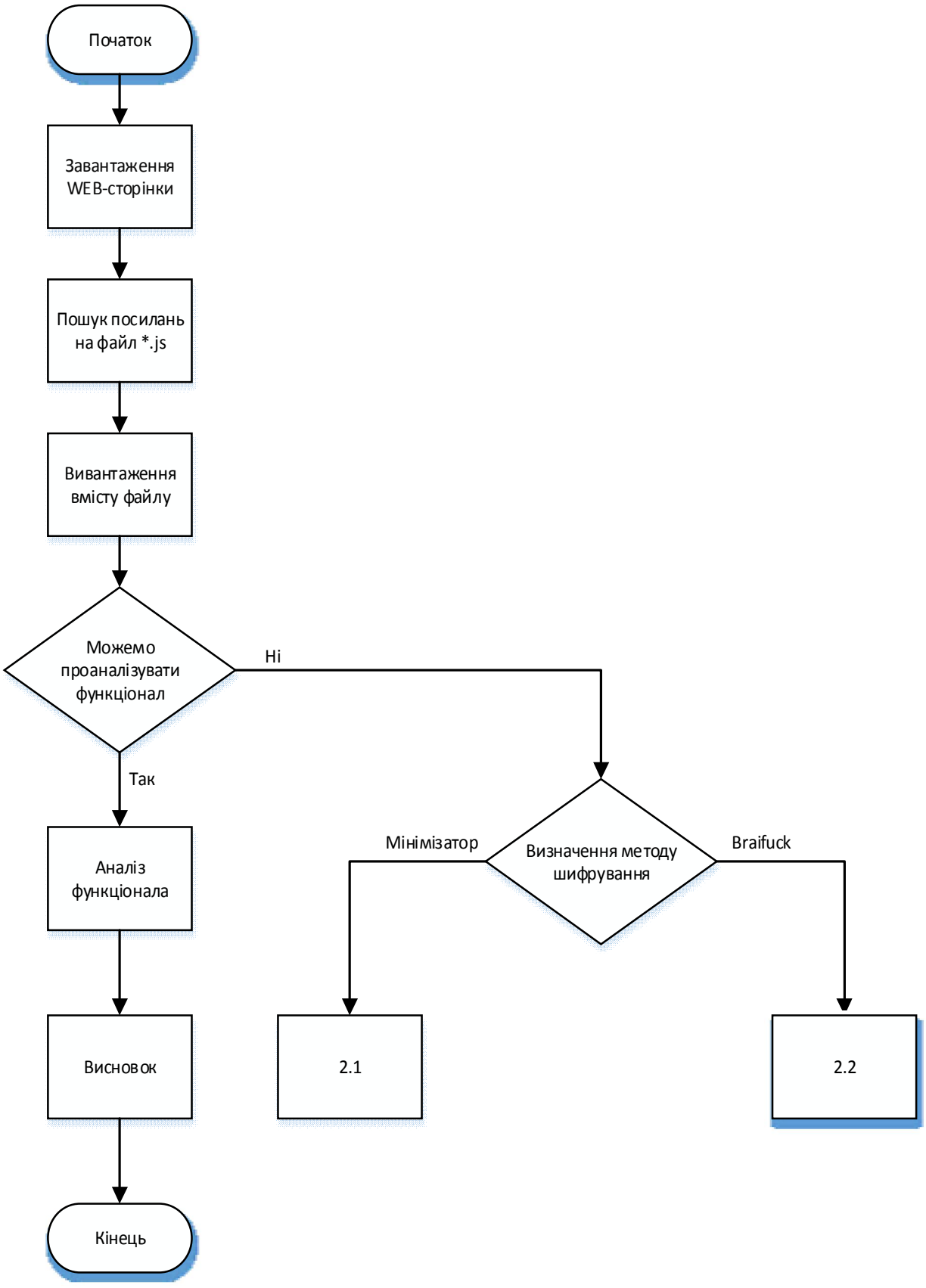

Рис. 5. Алгоритм деобфускаціїї зашифрованих скриптів методом запечатування

Алгоритм полягає в наступному:

1) Визначаємо посилання на файл JavaScript. Для цього скануємо сторінку і шукаємо посилання на файли із розширенням *.js.

2) Розпочинаємо розшифрування коду. Копіюємо вміст файлу в змінну. Після чого розпочинається процес сканування отриманого коду.

3) Визначаємо, чи можемо проаналізувати функціонал, який зберігається у файлі чи потрібно провести процес деобфускації.

4) Визначаємо метод обфускації або напряму проведення аналізу функціоналу. Якщо неможливо провести оцінку надійності коду – значить він прихований, а тому можливо й небезпечний. Для цього треба використати методи розпізнавання обфускованих скриптів із попередніх пунктів. У файлі може зберігатися обфускований код методом мінімізації або методом, який приводить до Braifuck-подібного вигляду. Тому визначаємо, який саме метод використовується та перейдемо до розшифровування використовуючи методику запропоновану у пунктах 2.1 та 2.2.

5) Якщо на WEB-сторінці більше одного посилання на файл, то вони вистроюються в чергу і починаючи з пункту 2 проходять процес деобфускації.

Висновки

Представлені алгоритми та методи деобфускації скриптів рекомендовано використовувати в програмному забезпеченні, інтегрованому із браузерами, для того щоб виявляти вірусну активність на WEB-ресурсах в режимі онлайн. Дані алгоритми потребують деякої модифікації для більш результативної роботи при їх впровадженні, але представлена ідея класифікації обфускованого коду по категоріях є досить новою і надає можливість розробити універсальні підходи до аналізування та розшифровування скриптів.

Література

1. Терейковський І. Нейронні мережі в засобах захисту комп'ютерної інформації / І. Терейковський. – К.: ПоліграфКонсалтинг. – 2007. – 198 с.

2. Badass JavaScript : Badass js is back white some badass obfuscation [Електронний ресурс] // – Режим доступу : http://badassjs.com/post/2929065287/obfuscation/

3. Adamcecc [Електронний ресурс] // – Режим доступу : http://adamcecc.blogspot.com/

4. Ajaxian : Want to pack JS and CSS really well? Convert it to a PNG and unpack it via Canvas [Електронний ресурс] // – Режим доступу : : http://ajaxian.com/