情報保護學會論文誌
第 22 卷 第 4 號, 2012. 8

난독화된 자바스크립트의 자동 복호화를 통한 악성코드의 효율적인 탐지 방안 연구*

지 선 호,† 김 휘 강‡
고려대학교 정보보호대학원

An Enhanced method for detecting obfuscated Javascript Malware using automated Deobfuscation*

Sun Ho Ji,† Huy Kang Kim‡
Graduate School of Information Security, Korea University

요 약

웹 서비스의 증가와 자동화된 공격 도구의 발달로 최근 대부분의 악성코드 유포 경로는 웹 서비스를 통하여 이루어지고 있다. 또한 웹의 기본 언어인 자바스크립트를 이용한 난독화 기법을 통해 악성코드 은닉 사이트의 URL이나 공격 코드를 숨기기 때문에, 기존 패턴 매칭 기반의 네트워크 보안 솔루션으로는 탐지에 한계가 존재하게 된다. 이를 해결하기 위하여 사용자의 웹브라우저에서 악성 자바스크립트를 탐지하기 위한 여러 방안이 제시되었지만, 최근 APT공격과 같이 특정 기업이나 조직 네트워크에 침투하기 위한 고도화된 공격에 대응하기에는 한계가 존재한다. 이런 유형의 공격에 대응하기 위해, 외부에서 유입되는 트래픽에 대해 난독화된 악성코드가 웹을 통해 유입되는지 일괄적인 탐지가 필요하며, 기존 패턴 매칭 기반 솔루션에서 탐지율의 한계를 극복하기 위해 난독화된 자바스크립트를 복호화 하여 숨겨진 악성코드를 탐지할 수 있는 새로운 방법이 필요하다. 본 논문에서는 오픈소스인 Jsunpack-n[1] 을 개량하여 자바스크립트의 함수 오버라이딩 기법과 별도의 자바스크립트 인터프리터를 통해 악성코드에 적용된 난독화 기법에 상관없이 숨겨진 악성코드를 자동적으로 탐지할 수 있는 도구를 제안한다.

ABSTRACT

With the growth of Web services and the development of web exploit toolkits, web-based malware has increased dramatically. Using Javascript Obfuscation, recent web-based malware hide a malicious URL and the exploit code. Thus, pattern matching for network intrusion detection systems has difficulty of detecting malware. Though various methods have proposed to detect Javascript malware on a users' web browser, the overall detection is needed to counter advanced attacks such as APTs(Advanced Persistent Treats), aimed at penetration into a certain an organization's intranet. To overcome the limitation of previous pattern matching for network intrusion detection systems, a novel deobfuscating method to handle obfuscated Javascript is needed. In this paper, we propose a framework for effective hidden malware detection through an automated deobfuscation regardless of advanced obfuscation techniques with overriding JavaScript functions and a separate JavaScript interpreter through to improve jsunpack-n.

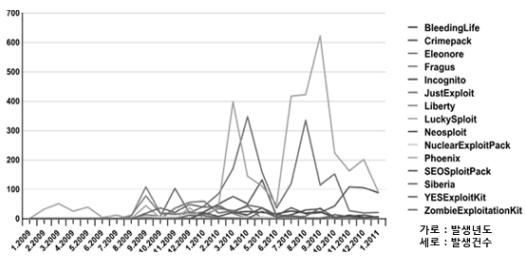
Keywords: Javascript; Malware; Obfuscation; Intrusion Detection

접수일(2012년 06월 20일), 수정일(2012년 08월 14일), 게재확정일(2012년 8월 15일)
* 본 연구는 국방과학연구소의 지원을 받아 수행하였습니다. (UD110051ED)
† 주저자, jisunho@korea.ac.kr
‡ 교신저자, cenda@korea.ac.kr

I. 서 론

인터넷 환경의 급속한 발전과 웹 서비스를 제공하는 사이트들의 폭발적인 증가로 이를 이용한 악성코드들의 대량 유포 사례가 빈번히 발생하고 있다. 최근 몇 년간 누적 악성코드 수치는 '09년도 약 2천백만,'10년도 약 3천만, '11년도 약 4천 4백만 개 가량으로 기하급수적으로 증가하였으며[2] 이에 따라 악성코드 유포에 가장 많이 사용되는 악성코드 은닉 사이트 URL의 숫자도 평균 2만 건 이상으로 꾸준히 증가하고 있다[3]. 이는 상업적인 목적을 토대로 악성코드 제작 그룹의 전문화와 조직화가 이루어졌기 때문이며, 그로 인해 유료로 판매되는 웹 공격용 툴킷(Web Exploit Toolkit) 을 이용한 웹상에서의 악성코드 유포가 [그림 1] 과 같이 증가하였기 때문이다[4]. 이에 대응하기 위한 기존 방안으로 네트워크 기반의 보안 솔루션과 클라이언트에서의 안티바이러스 솔루션의 지속적인 패턴 업데이트를 통하여 해당 악성코드들의 유포를 사전 차단하고 있지만, 각종 우회기법을 이용한 유포 시도로 인하여 기존의 방안으로는 탐지율이 떨어지며 신속한 대응에 한계가 발생하게 된다. 대표적으로 웹에서 기본적으로 사용되는 언어인 자바스크립트의 난독화를 이용하여 악성코드 은닉사이트의 URL 과 익스플로잇 코드를 난독화된 자바스크립트 코드 안에 은폐하여 각종 보안 솔루션을 우회하는 기법이 가장 많이 사용되고 있으며, 다중 난독화나 별도의 인코딩 함수 구현 등 다양한 난독화 기법의 개발로 패턴 매칭을 통한 탐지에 더욱 어려움을 가져오고 있다. 최근에는 PDF 파일 형식과 어도비 플래시의 SWF 파일 형식에서도 내부에 자바스크립트 언어를 포함할 수 있도록 지원이 가능하기 때문에 이를 이용한 보안 솔루션 우회 시도도 더욱 증가하고 있다.

이러한 패턴 매칭 기반 보안 솔루션의 한계를 극복하기 위해 웹 서비스의 최종 단계인 웹브라우저에서



[그림 1] 웹 공격용 툴 킷(Web Exploit Tool Kit) 의 증가 통계

악성 자바스크립트를 탐지하기 위한 여러 방안이 제시되었다. 대표적으로 웹브라우저에 샌드박스 개념을 도입하여 Internet Explorer 9 의 보호 모드나 Google Chrome 의 시크릿 창과 같은 기능을 제공하고 있으며, 특정 브라우저에서 자바스크립트의 처리를 수행하는 모듈의 후킹을 통해 난독화된 자바스크립트의 복호화된 코드를 획득하는 방안도 소개되고 있다[5]. 또한 대형 인터넷 검색 사이트에서는 자체적으로 크롤링(crawlering)을 수행하여 악성코드가 은닉된 웹 페이지들의 정보를 DB 로 구축하여 사용자가 검색 사이트를 경유하여 접근하는 웹 페이지에 대한 악성코드 은닉 여부를 알려주는 서비스를 수행하고 있다. 그러나 최근 APT(Advanced Persistent Threat) 공격과 같이 특정 기업이나 조직 네트워크에 침투하기 위한 고도화된 공격에 대응하기 위해서는 클라이언트 각각의 탐지 정책보다는 외부망과 연결되는 라우팅 구간에서의 통합적인 탐지 및 모니터링이 필요하게 된다. 이와 같은 상황에서 기존 패턴 매칭 기반 탐지의 한계를 극복하기 위해서는 난독화된 자바스크립트를 자동 복호화 하여 획득한 원문에서 악성코드 은닉 사이트 URL이나 별도의 공격 코드를 탐지하는 기술이 필요하다. 따라서 본 논문에서는 자바스크립트 언어를 정의하고 실행할 수 있는 별도의 인터프리터를 이용한 자바스크립트 함수의 오버라이딩(Overriding) 기법을 통해 난독화 기법이나 다중 난독화에 상관없이 자동 복호화를 구현하는 방안을 제시하고, 관련된 오픈소스를 활용하여 네트워크 구간에서의 난독화된 악성 자바스크립트에 대한 자동 복호화 및 악성코드 은닉 사이트URL 과 공격 코드를 탐지할 수 있는 동적 탐지 도구를 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 난독화된 자바스크립트의 탐지와 관련된 연구 결과를 살펴본 뒤, 자동 복호화를 구현하기 위한 방안을 살펴보고, 자바스크립트 언어의 특징을 활용한 함수 오버라이딩 기법을 통해 난독화된 코드를 복호화할 수 있는 기법을 확인한다. 또한 현재 개발된 난독화된 자바스크립트의 복호화 도구들의 특징을 살펴보고, 본 논문에서 제안하는 탐지 도구 제작에 기본 바탕이 되는 오픈소스의 특징과 문제점을 확인한 뒤, 3장에서는 해당 오픈소스의 약점을 보완하기 위한 방안과 제안 도구의 전체 구성도 및 세부 구성요소를 살펴본다. 4장에서는 3장에서 제안한 도구를 활용하여 기존 탐지 기법과의 탐지율을 비교하고 마지막 5장에서는 결론 및 향후 연구방향을 제시한다.

〔표 1〕 자바스크립트 난독화 기법 동향

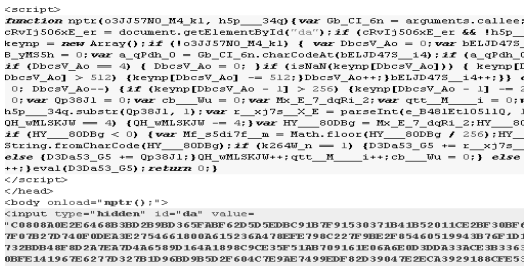
구분	기존 난독화	최근 난독화
자바스크립트 기능함수 활용	escape, eval , onload , document.write/writeln 과 같은 자바스크립트의 기능함수들과 이벤트를 이용하여 부호화된 형태의 자바스크립트 코드를 실행	기존 자바스크립트 난독화 기법에서 복호화를 어렵게 하기 위하여 복호화 키를 공격자의 서버에서 별도로 제공하거나 인코딩 함수를 별도제작.
문자열 인코딩	BASE64 , XOR , 문자열 Split, 8-bit ASCII 과 같이 기존의 암호화 기법이나 문자열 표현 방식의 차이점을 이용하여 자바스크립트 코드를 난독화	
자바스크립트 난독화 전용 도구	여러 자바스크립트 난독화 기법을 활용하여 별도로 제작한 전용 도구. 웹 서비스나 별도 패키지로 제공. Windows Script Encoder, Dean Edward's〔6〕 JavaScriptObfuscator〔7〕 등이 있음.	대규모의 악성코드 유포에 활용되는 웹 공격용 툴킷에서 별도의 자바스크립트 난독화 수행. Phoenix Exploit Toolkit, NeoSploit Exploit Toolkit 등이 있음

II. 관련 연구

본 장에서는 우선적으로 최근 자바스크립트의 난독화 기법을 이용한 악성코드 은닉 방법을 살펴보고 기존의 난독화된 자바스크립트 탐지에 관한 연구 결과와 현재 개발된 복호화 도구들의 특징에 관해 살펴본 뒤, 오픈소스로 개발된 Jsunpack-n의 특징과 문제점을 파악하였다.

2.1 최근 자바스크립트 난독화 기법의 특징

최근의 자바스크립트 난독화 기법은 〔표 1〕과 같이 자바스크립트의 기존 함수들을 응용하는 방법에서 발전하여 난독화/복호화를 수행하는 함수를 별도로 제작하거나 복호화에 사용되는 키값이나 변수 값들을 사용자가 접속할 때마다 변경하여 일반적인 패턴 매칭 기반 솔루션에서 탐지가 어렵도록 구성되어 있다. 또한 단계별로 다중 난독화를 적용하여 복호화 작업이 어렵도록 하고 있다. 〔그림 2〕 는 웹 공격용 툴 킷중 하나인 NeoSploit에서 볼 수 있는 난독화된 코드의 일부이며 〔그림 3〕은 복호화 과정에 필요한 비밀 키를 별도의 URL과 파라메터 값을 통해 생성하는 코드의 일부이다〔6〕.



〔그림 2〕 NeoSpolit의 난독화 코드 일부

```
)
if (Wj50EQ0_H525S){
  Wj50EQ0_H525S = Wj50EQ0_H525S.substr(0, 10);
  var bVsN83LG = "";
  for (var nOy_C_Lw_VP = 0; nOy_C_Lw_VP < Wj50EQ0_H525S.length; nOy_C_Lw_VP ++ ){
    var WAE1P40 = Wj50EQ0_H525S.charCodeAt(nOy_C_Lw_VP).toString(16);
    if (WAE1P40 < 2)bVsN83LG += "0";
    bVsN83LG += WAE1P40;
  }
  while (bVsN83LG.length < 20){
    bVsN83LG += "00";
  }
  g_yT5dkbUPf += "L" + bVsN83LG;
}
var b3Wn8i2 = document.createElement("script");
b3Wn8i2.setAttribute("type", "text/javascript");
b3Wn8i2.setAttribute("src",
"http://begswe.info/page/shop.php/t002106Xc1768ea8Y8df30af8Z0100f070" + g_yT5dkbUPf);
document.body.appendChild(b3Wn8i2);
```

〔그림 3〕 복호화 비밀 키를 별도로 생성하는 코드

고도화된 난독화 기법 외에도 PDF 나 SWF 파일에서 자바스크립트 언어를 지원하는 점을 악용하여 이메일 첨부나 링크페이지를 통해 악성 자바스크립트가 포함된 해당 파일들을 이용한 악성코드 유포도 문제가 되고 있다. PDF 파일은 많은 객체(object)들의 상호참조로 구성되어 있는데, 이러한 객체에 〔그림 4〕와 같이 악성 자바스크립트나 PE 파일과 같은 데이터를 저장해 두고 참조하여 악성행위를 유발할 수 있으며, 객체에서 FlatDecode 방식의 압축을 통해 보안 솔루션의 탐지를 우회하는 경우가 많다. SWF 파일 또한 Javascript 와 유사한 Actionscript 지원을 통해 〔그림 5〕처럼 파일 내부에 악성코드 은닉 사이트 URL 이나 취약점을 공격하는 쉘코드를 삽입하여 악

〔그림 4〕 PDF 객체에 저장된 악성 자바스크립트

```
[HEADER] File version: 6
[HEADER] File is zlib compressed. Ratio: 86%
[HEADER] File size: 296 (Depacked)
[HEADER] Frame rate: 12.000000
[HEADER] Frame count: 1
[HEADER] Movie width: 10.00
[HEADER] Movie height: 10.00
[009] 3 SETBACKGROUNDCOLOR (ff/ff/ff)
[00c] 263 DOACTION
( 259 bytes) action: GetUrl URL:"http://www.cgi5-eby.com/ws2/eBayISAPI.dll
( 0 bytes) action: End
[001] 0 SHOWFRAME 1 (00:00:00,000)
[000] 0 END
```

〔그림 5〕 SWF 내부 Actionscript 를 통해 실행되는 악성코드 은닉 사이트 URL

성코드를 유포하게 된다.

2.2 자바스크립트 난독화 코드 탐지 관련 연구

위와 같은 난독화 기법의 고도화로 각각의 난독화 기법에 대한 복호화 알고리즘을 구현하는 것은 한계가 존재하게 되며, 이에 따라 난독화된 자바스크립트의 자동 복호화를 구현하기 위해서는 다양한 난독화 기법에 관계없이 복호화를 수행할 수 있는 범용적인 방안이 필요하다. 이와 관련된 기존 연구 결과를 살펴보면 난독화된 자바스크립트의 패턴 자체를 분석하여 탐지하는 기법과, 별도의 자바스크립트 인터프리터나 웹브라우저의 자바스크립트 관련 라이브러리 후킹 기법을 이용한 복호화 기법으로 크게 분류할 수 있다.

〔표 2〕 자바스크립트 난독화 패턴 탐지 관련 연구

제목	요약
Obfuscated Malicious Javascript Detection using Classification Techniques〔7〕	난독화된 자바스크립트의 65 가지 특성들을 선택하여 학습을 통해 자바스크립트의 난독화 여부를 탐지 수행.
Automatic Detection for Javascript Obfuscation Attacks in Web Pages through String Pattern Analysis〔8〕	정적 데이터 흐름 분석을 통한 패턴 데이터 후보군들을 추출하고 3가지 통계 데이터를 이용하여 난독화 여부를 결정하여 탐지 수행.
Microsoft, Zozzle: Low-overhead Mostly Static Javascript Malware Detection〔9〕	악성코드로 추측 가능성이 높은 구문 요소를 식별하기 위하여 자바스크립트 AST(Abstract Syntax Tree) 의 계층적 특징을 통한 Bayesian 분류 기법을 사용하여 난독화된 자바스크립트의 탐지 수행.
Malicious Web Content Detection by Machine Learning.〔10〕	혼합된 171 가지 특성들을 이용하여 DHTML 페이지의 실행 과정에서 난독화 여부를 확인하여 탐지 수행.

2.2.1 자바스크립트 난독화 패턴 자체의 탐지 관련 연구

자바스크립트의 난독화 패턴 자체를 탐지하는 기법은 주로 난독화된 문자열의 특징을 데이터마이닝을 통해 분석하여 탐지 패턴을 생성해 내는 방법으로, 〔표 2〕의 관련 연구들과 같이 반복적인 학습을 통해 정상 자바스크립트와 악성 자바스크립트 코드를 구분하게 된다. 이는 네트워크 구간에서의 많은 양의 트래픽을 탐지할 때 효율적이지만, 패턴 학습 시간이 필요하며, 자바스크립트의 원문 자체를 확인할 수 없기 때문에 실제로 정상적인 자바스크립트 코드에 난독화를 적용한 경우에 대한 명확한 구분이 어려워 오탐율이 높아질 수 있다.

2.2.2 자바스크립트 난독화 코드의 원문 획득 관련 연구

난독화된 자바스크립트의 범용적인 복호화를 위한 기법은 〔표 3〕의 관련 연구들과 같이 자바스크립트 언어가 인터프리터에 의해 처리될 때 난독화된 코드가 실행되는 구간에서 중간에 실행 결과에 대한 원문을 가로채거나 추적하여 화면에 출력하는 방식이 대표적이다. 이는 자바스크립트가 어떠한 난독화 기법이 사용되었는지, 몇 번의 난독화를 적용 하였는지 와 관계

〔표 3〕 자바스크립트 난독화 코드의 원문 획득 관련 연구

제목	내용
SpyProxy: Execution-based Detection of Malicious Web Content〔11〕	가상 머신 형태의 프록시 솔루션을 구축하여 사용자의 브라우저에 웹 콘텐츠가 전달되기 전에 실행 기반의 분석을 통해 알려지지 않은 위협 및 제로데이에 대한 방어를 수행
Detection and Analysis of Drive-by Download Attacks and Malicious Javascript Code〔12〕	사용자가 알지 못하는 사이에 다운로드 되어 실행되는 “Drive-by Download” 공격에 대응하여 자동으로 악성 자바스크립트를 식별하고 에뮬레이션을 통해 변종을 검출할 수 있는 분석 시스템을 개발
ADSandbox: sandboxing Javascript to fight malicious websites〔13〕	BHO(Browser Helper Object)를 통해 웹브라우저를 실행하는 클라이언트에서 별도의 샌드박스 구간을 설정하여 자바스크립트를 사전 실행한 뒤 특정 기준표에 의해 악성여부를 판단함.
Websense: The Ultimate Deobfuscator	윈도우의 기본 웹브라우저인 인터넷 익스플로러의 Jscript.dll 과 mshtml 의 특정 함수들을 후킹하여 난독화된 코드의 원문을 확인

〔표 4〕 난독화된 자바스크립트의 복호화 관련 도구

구분	이름
별도의 자바스크립트 인터프리터를 이용	Malzilla, JSUNPACK, WEPAWET, Rhino, Spidermonkey + V8, The Mina
웹브라우저의 자바스크립트 관련 모듈의 후킹을 이용한 방식	Firebug - Firefox plug-in, Javascript Deobfuscator, Microsoft IE8 Developer Tools, Microsoft Script Debugger, Google Chrome Developer Tools

없이 실행 과정에서는 스스로 복호화 과정을 거칠 수 밖에 없으며 이를 이용하여 복호화 과정에서 사용되는 자바스크립트의 함수 내용이나 발생되는 관련 이벤트의 내용을 통해 자바스크립트의 원문을 획득할 수 있게 된다. 일반적으로 자바스크립트를 처리하는 대표적인 인터프리터인 웹브라우저의 내부 모듈에서 결과를 가져오는 것이 가장 성능이 좋겠지만 웹브라우저를 벗어나 여러 자동화 용도로 도구를 만들기 위해서는 별도의 인터프리터를 이용하여 웹브라우저와 유사한 환경을 만들어 복호화 작업을 수행하게 되는데 현재까지 제작된 관련 도구들은 〔표 4〕과 같다.

2.3 자바스크립트 함수 오버라이딩

자바스크립트에서의 함수(Function) 는 자바스크립트에서 다루는 객체의 특별한 타입으로 볼 수 있다. 함수 자체가 객체로 간주되기 때문에 해당 객체에 대한 참조가 가능하며 동적 함수 생성을 통해 기존 함수를 새롭게 정의하여 덮어쓰는 함수 오버라이딩 행위도 가능하게 된다. 오버라이딩이란 상위 클래스에 있는 메서드와 똑같은 메서드를 하위 클래스에서 다시 만드는 행위를 말하며, 위에서 말한 자바스크립트 기능함수를 재 정의할 수 있는 특성을 오버라이딩의 개념으로 표현할 수 있다. 해당 기법을 통하여 자바스크립트에서 난독화 작업 시 이용되는 주요 기능함수들을 〔그림 6〕과 같이 재정의하여 코드 실행 시에 복호화된 코

```
var window = this;
window.unescape = unescape;
window.parent = window;
window.execScript = eval;
window.eval = eval;
window.open = function (url){
    print("//url open = " + url)
};
```

〔그림 6〕 자바스크립트 기능함수 재정의 예

〔표 5〕 웹브라우저 별 사용되는 자바스크립트 인터프리터

브라우저 구분	자바스크립트 인터프리터 구분
Internet Explorer	Chakra
Firefox	Spidermonkey
Chrome	V8
Safari	JavaScriptCore
Opera	Carakan

드가 출력되도록 유도하는 작업이 가능하게 되며 이를 이용하여 난독화된 자바스크립트의 범용적인 복호화 작업이 가능하게 된다.

2.4 자바스크립트 인터프리터

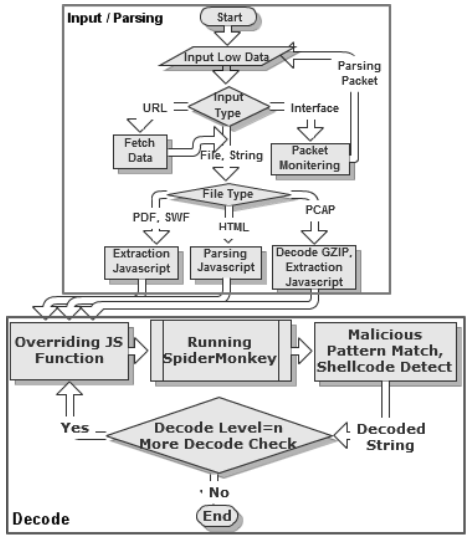
자바스크립트 인터프리터는 자바스크립트 언어를 정의하고 실행할 수 있는 별도의 해석자를 의미하며, 현재 〔표 5〕와 같은 인터프리터들이 각종 웹브라우저에서 사용되고 있다. 2.3에서 설명된 자바스크립트 함수 오버라이딩을 이용한 복호화 작업을 진행하기 위해서는 웹브라우저와 분리된 별도의 자바스크립트 인터프리터가 필요하며, 여러 인터프리터 중 오픈소스로 공개되어 수정이 가능한 Spidermonkey를 이용하여 난독화된 자바스크립트의 복호화 작업을 진행할 수 있다.

2.5 Jsunpack-n

본 논문의 주제와 같이 네트워크 구간에서의 난독화된 악성 자바스크립트의 복호화를 위해서는 별도의 자바스크립트 인터프리터를 이용하여 2.3에서 설명한 함수 오버라이딩을 통해 복호화 작업을 수행하는 것이 적합하다. 2.2.2에서 조사한 자바스크립트 복호화 도구들의 기능과 확장성을 비교하여 본 논문의 주제에 활용될 수 있는 소스를 확인한 결과 Jsunpack-n 이 가장 적합한 도구로 판단되었다. Jsunpack-n은 현재 Google Code Project에서 오픈소스로 공개된 도구로서 Blake Hartstein 이 제작하여 Shmoocon 2010 컨퍼런스에서 발표한 도구이다. Jsunpack-n 도구의 전체 구성도는 〔그림 7〕 과 같으며 크게 입력, 추출, 복호화의 3단계로 구성되는데, 실행 과정대로 구현된 기능을 정리하면 다음과 같다.

2.5.1 프로그램 입력 형식 지정

복호화를 위해 분석할 데이터 입력 형식을 크게 두



〔그림 7〕 Jsunpack-n 전체 구성도

가지로 나누어 네트워크 트래픽 모니터링 및 별도의 대상 파일 경로를 지정해주는 정적 입력형식과, 분석을 수행할 웹 사이트의 URL을 입력하여 해당 URL에서 연결되는 페이지들을 크롤링 하는 동적 입력형식으로 구현하게 된다. 정적 입력형식에서 분석되는 파일 내에 URL 정보가 존재할 경우 별도의 옵션을 통해 해당 URL을 크롤링 하여 연관된 모든 정보를 수집하게 된다.

2.5.2 네트워크 트래픽 실시간 모니터링 및 재조합

네트워크 인터페이스로 유입되는 트래픽에 대한 모니터링 및 IP 계층에서 데이터그램의 분할된 조각의 재조합과 TCP 스트림 데이터의 리어셈블링을 통해 패킷 데이터를 저장한다. 해당 프로그램에서는 libnids(NIDS, Network Intrusion Detection System)라이브러리의 python 버전을 사용하여 해당 기능을 구현하게 된다.

2.5.3 패킷 데이터 파싱 및 특정 파일 형식 추출

TCP 스트림 데이터에서 GZIP으로 압축된 HTTP 형식의 데이터를 복호화하고 난독화된 자바스크립트 데이터 추출 및 PDF , SWF 형식의 파일 데이터를 추출하고 파일 내부의 자바스크립트 데이터를 추출한다. 해당 프로그램에서는 오픈소스로 Junpack-n 에 포함된 모듈을 사용하여 해당 기능을 구현하게 된다.

〔표 6〕 가상 브라우저 환경

PDF리더	웹브라우저
Version 9.1 Version 8.0 Version 7.0	IE7/XP IE8/Vista Opera, Firefox

2.5.4 웹사이트 동적 크롤링 및 특정 파일 형식 추출

복호화 과정에서 URL을 크롤링 하여 연결된 페이지들에 대한 상세 데이터를 파싱할 때 페이지에 존재하는 PE 파일이나 PDF 형식의 데이터와 자바스크립트 코드, 쉘코드를 추출하여 별도로 저장하게 된다. 이를 통해 난독화된 자바스크립트에 대한 자동 복호화 과정에서 악성행위와 연관된 콘텐츠를 동적으로 수집하여 악성코드가 포함된 사이트에 대하여 기존의 크롤링 기반의 악성코드 유포 사이트 탐지 솔루션에 비해 더욱 상세한 분석을 수행할 수 있다.

2.5.5 다양한 자바스크립트 실행 환경 구성

웹브라우저의 종류와 버전이 다양하고, PDF 리더의 버전에 따라 해석되는 파일 구조가 상이할 수 있기

〔표 7〕 복호화를 위해 재정의 되는 메서드

종류	이름	설명
Javascript	setTime Out	지정한 시간 이후에 자바스크립트 코드나 기능함수를 호출
ActiveX Control	SaveTo File	시스템에서 파일을 저장할 수 있음.
ActiveX Control	OpenWeb File	파일 다운로드 및 실행
Javascript	set Attribute	지정된 Attribute에 값을 할당. URL 정보 확인을 위해 출력
ActiveX Control	Create Object	FSO(FileSystemObject) 개체를 만들기 위한 메서드. 로컬 리소스 접근 정보 확인을 위해 출력
Javascript	append Child	개체에 자식 Element 를 첨부함.
Javascript	Create Element	새로운 Element를 생성함
Javascript	open	새로운 window를 열고 지정된 URL 에 접근
Javascript	eval	문자열의 형태로 표시되어 있는 Javascript 코드를 계산하거나 수행.

때문에 실제 다양한 환경에서 자바스크립트가 실행되는 것과 같은 가상 브라우징 환경 구현이 필요하다. 이를 위해 해당 프로그램에서는 복호화 실행전 〔표 6〕와 같이 별도로 브라우징 환경 값을 설정하는 모듈을 적용하여 해당 기능을 구현하게 된다.

2.5.6 자동 복호화를 위한 기능 구성

복호화 과정에서는 2.5.1 - 2.5.5 과정에서 추출된 난독화된 자바스크립트 데이터에 2.3 에서 설명한 자바스크립트 함수 오버라이딩 기법을 적용한 뒤별도 자바스크립트 인터프리터인 Spidermonkey를 이용하여 해당 자바스크립트를 실행하게 된다. 〔표 7〕에서처럼 복호화를 위한 자바스크립트의 메서드를 재 정의하여 난독화된 자바스크립트가 실행되는 과정에서 악성 행위로 판단될 수 있는 메서드의 호출이나 URL 정보, Eval 기능함수가 호출될 때 실행 결과를 화면에 출력할 수 있도록 한다.

2.5.7 복호화된 원문에 대한 기존 악성코드 패턴 비교

Jsunpack-n 은 자바스크립트 복호화 기능 외에 추가적으로 도출된 원문에 대하여 사전 정의된 악성코드 패턴 데이터를 대입하여 자동 복호화와 함께 해당 데이터에 악성코드에 대한 존재유무와 종류를 확인할 수 있는 네트워크 구간의 악성코드 자동 탐지 솔루션의 기능도 수행할 수 있다. 악성코드 패턴의 정의는 〔그림 8〕과 같이 Yara Project〔14〕에서 제공하는 형식의 파일 데이터와 〔그림 9〕의 자바스크립트 함수 오버라이딩 과정에서 추가되는 JS 형식의 파일에도 정의되어 악성코드 종류를 판단하게 된다. 이는 악성코드와 관련된 자바스크립트 함수가 호출되는 행위에서 바로 함수 호출시 출력되는 메시지에 해당 악성코드의 종류를 출력해주는 방식이기 때문에 복호화된 원문 데이터에 대한 악성코드 탐지 속도와 성능을 더욱

```
rule MSOfficeWebComponents
{   //Expect ActiveX with it, OWC10.Spreadsheet OWC11.Spreadsheet
    meta:
        ref = "CVE-2009-1136"
        impact = 7
    strings:
        $cve20091136_1 = "msDataSourceObject" nocase fullword
        $cve20091136_2 = "OWC10.Spreadsheet" nocase fullword
        $cve20091136_3 = "OWC11.Spreadsheet" nocase fullword
    condition:
        1 of them
}
```

〔그림 8〕 Yara rules 의 정의된 예

```
var media = {
        newPlayer : function(a){
                if (a == null){
                        print("//alert CVE-2009-4324 media.newPlayer with NULL parameter"
                }
                else {
                        print("//warning CVE-2009-4324 media.newPlayer access");
                }
        },
        createPlayer : function(a){
                print("//warning CVE-2009-4324 media.newPlayer access");
        },
};
```

〔그림 9〕 함수 오버라이딩 과정에서 해당 함수와 관련된 악성코드 종류 출력

향상시키게 된다.

III. 제안하는 난독화 악성 자바스크립트의 자동 복호화 도구

2.5 에서 소개한 Jsunpack-n 은 이미 공개된 오픈소스로도 난독화된 자바스크립트의 복호화 작업에 활용될 수 있지만 공격자들의 오픈된 소스코드 분석이용이하여 이를 이용한 복호화 방지 코드 추가 가능성이 존재하며, 여러 자바스크립트 난독화 기법을 테스트한 결과 복호화가 불가능한 난독화 기법이 존재하는 취약점을 가지고 있다. 본 장에서는 Jsunpack-n 의 오픈소스를 활용하여 기존 소스에서의 한계점을 해결하고 기능을 확장하여 네트워크 기반에서 자바스크립트의 자동 복호화를 통해 악성코드의 유입을 사전 차단할 수 있는 도구를 제시한다.

3.1 Anti-Deobfuscation 방어 코드 추가

Anti-Deobfuscation 이란 자바스크립트 인터프리터를 통한 난독화된 코드의 복호화 기법을 차단하기 위하여 악성 스크립트 상에서 해당 스크립트의 실행 환경을 판별하여 실제 브라우저 환경이 아닐 경우 별도의 인터프리터를 통한 분석행위로 간주하여 해당 스크립트의 분석을 사전 차단하는 코드를 말한다. 총 3 가지 경우의 Anti-Deobfuscation 이 발생 가능한 코드를 수정하여 업데이트 하였다.

3.1.1 window.location 을 이용한 분석환경 탐지 대응

자바스크립트의 window.location 객체는 현재 열려있는 웹 페이지의 URL 정보를 얻어올 수 있다. location 객체 속성 중 host 속성을 이용하여 URL 정보를 가져오게 되면 정상적으로 웹 서버를 통해 열리는 페이지는 도메인 정보를 포함하고 있지만, 분석을 위하여 별도의 자바스크립트 인터프리터를 통해 입

력 값을 파일로 받을 경우 도메인 형태가 아닌 Linux OS 에서 '/' 문자열이 포함되는 시스템 파일 경로 형태의 문자열이 반환되므로 해당 차이점을 이용하여 별도의 자바스크립트 인터프리터 환경을 탐지할 수 있게 된다. 이를 해결하기 위하여 입력 값에 '/' 문자열이 포함되지 않도록 분석 파일의 경로를 프로그램 실행 지점 상에 동일한 경로로 이동하여 작업을 수행하도록 수정하였다.

3.1.2 특정 객체명과 전역 변수 명을 이용한 분석환경 탐지 대응

오픈소스인 Jsunpack-n은 소스코드 상에서 별도로 선언된 자바스크립트 전역 변수와 객체들의 이름이 모두 공개되어 있다. 이를 이용하여 자바스크립트 코드 실행 시에 Jsunpack-n에서 별도로 선언한 자바스크립트 전역 변수 또는 객체의 이름을 검색하는 공격코드를 이용하게 되면 해당 이름이 발견될 경우 별도의 자바스크립트 인터프리터 환경을 탐지할 수 있게 된다. 이와 같은 방법으로 별도의 인터프리터 환경을 탐지할 수 없도록 기존 Jsunpack-n에서 선언된 전역 변수명과 객체들의 이름이 랜덤하게 선언되도록 수정하였다.

3.1.3 toString 메서드를 통한 분석환경 탐지 대응

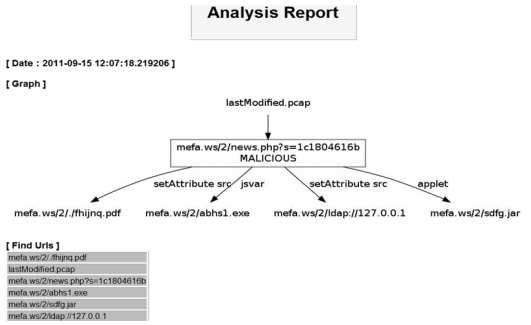
자바스크립트의 toString 메서드는 모든 객체에 기본적으로 제공되는 자바스크립트 객체들의 구성원 중 하나이다. 그러나 객체의 유형에 따라 다른 기능을 가지게 되는데, 특히 객체가 문자열이 아닌 함수일 경우에는 해당 기능함수의 생성자나 객체 타입을 반환하게 된다. 이를 이용하여 자바스크립트 오버라이딩을 위하여 사전 정의된 자바스크립트 기능함수에 대한 탐지를 할 수 있게 된다. 예를 들어, 난독화된 자바스크립트 코드에서 app.setInterval.toString ().match(/print/) 와 같은 toString 메서드를 이용한 코드를 삽입하게 된다면 기존 자바스크립트와 다를 수밖에 없는 app.setInterval 객체 내부에 정의된 print 구문을 탐지하게 되어 Anti-Deobfusc ation 기법으로 활용이 가능하다. 또한 자바스크립트의 argument 배열변수의 callee property 는 실행중인 함수의 객체이기 때문에 arguments. callee.toString() 을 이용하여 현재 실행되는 함수의 내부 코드 자체를 출력할 수가 있어 Anti-Deobfuscation 기법으로 활용될 수 있다. 이를 방지하기 위해 app.setInterval.tostring() 자체를 재정의 한다고 해도 app.setInterval. tostring.tostring() 과 같은 구문으로 다시 내부참조가 가능하다. toString 을 연속적으로 삽입하여 기존의 자바스크립트 객체와 다른 성격의 코드를 찾아내면 현재 환경이 별도의 인터프리터를 통한 분석이라는 것을 탐지할 수 있는 것이다. 이를 해결하기 위해 자바스크립트 객체들의 일반적인 메서드를 함수 선언 시 재정의 할 수 있는 __defineGetter__ 구문이 활용될 수 있다. [그림 10]과 같이 toString 을 재 정의하여 복호화를 위해 재 정의된 자바스크립트의 기능함수들을 탐지할 수 없도록 수정하였다.

```
var ToStringFix = function(object, fn_header) {
    object.__defineGetter__("toString", function() {
        var original = this.__proto__.toString;
        this.__proto__.toString = function() {
            var a = original.call(this);
                    if (a.match(Rcheck))
                return original.toString();
            return a;       };
        return function() {
            [Rcheck]
            var BODY = " {¥n¥t[native code]¥n}";
            return fn_header + BODY;        }
    });};
ToStringFix(window.open, 'function open()');
ToStringFix(alert,'function alert()');
```

[그림 10] _defineGetter_ 를 이용한 toString 재정의

3.2 Window script encoder 복호화 코드 추가

Window script encoder 는 Microsoft에서 제공하는 자바스크립트 난독화 도구로서, 스크립트에 포함되어 있는 지적 소유권을 보호하기 위해 제공되지만 악성코드 유포를 목적으로 하는 웹 페이지의 악성 자바스크립트 코드를 은폐하기 위하여 악용되기도 한다. 해당 도구를 이용한 난독화 형식은 자바스크립트의 조합으로 행해지는 암호화가 아닌 Microsoft에서 자체적인 암호화 엔진을 제공하여 인터넷 브라우저상에서 해석되므로, Spidermonkey 와 같은 별도의 자바스크립트 인터프리터에서는 실행이 되지 않는 문제점이 발생한다. 이를 해결하기 위하여 Spidermonkey를 이용한 자바스크립트 실행전에 window script encoder 에 의해 난독화된 코드에서 볼 수 있는 "VBScript.encode", "Jscript.encode" 문자열을 검색하여 난독화 여부를 판단한 뒤, 복호화 작업을 별도로 선행한 뒤에 자바스크립트 인터프리터에 의해 실행이 되도록 수정하였다. window script encoder 에 대한 복호화 작업은 인터넷상에 암호화에 사용된



〔그림 11〕 HTML 형식의 분석 리포팅 화면 일부

키값이 공개되어 있기 때문에 Brownstone 이 작성한 window script decoder〔15〕 소스코드를 이용하여 모듈을 추가하였다.

3.3 출력 형식의 가독성 향상 및 상세 리포트 작성

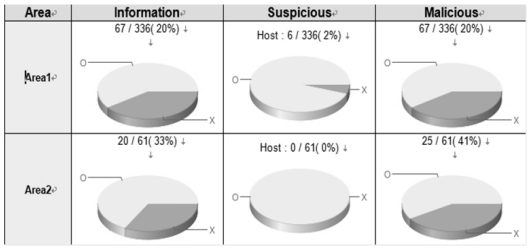
복호화된 자바스크립트 코드는 문장 간의 공백이나 줄 바꿈 처리가 되어있지 않아 식별이 어렵기 때문에 가독성을 향상시키기 위하여 js-beautify〔16〕 python 모듈을 프로그램의 복호화 단계에 적용하여 가독성을 향상시켰으며 크롤인 되는 URL의 연결 관계를 트리 형 그래프로 표현하여 복호화 및 악성코드 분석 결과를 〔그림 11〕과 같이 HTML 형식으로 상세 정리하여 리포팅 할 수 있는 모듈을 추가하였다.

3.4 복호화 결과 실시간 확인 그래프 작성

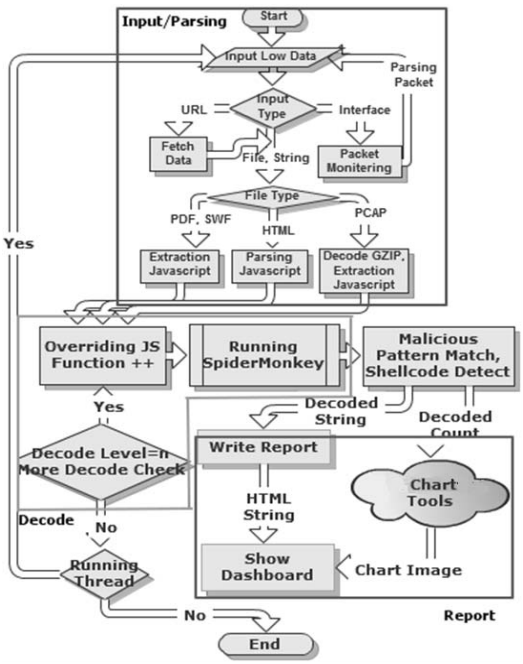
네트워크 구간에서 해당 도구를 이용하여 실시간으로 복호화 작업이 이루어질 때 출력되는 정보들을 Apache 웹 서버에서 웹 페이지 형식으로 그래프들을 확인할 수 있도록 〔그림 12〕와 같이 일정한 시간 간격으로 그래프 이미지를 생성하여 웹페이지에 출력하는 기능을 추가하였다. 그래프 생성에 필요한 모듈은 Google Chart Tools〔17〕 에서 제공하는 온라인 서비스를 이용하여 그래프 이미지를 생성하였다.

3.5 전체 구성 및 실행 단계

제안하는 난독화 악성 자바스크립트 자동 복호화 도구의 기본적인 실행 흐름도는 〔그림 13〕 과 같다. 데이터 입력과 패킷 파싱과 같은 복호화 작업 이전의 절차는 기존 Jsunpack-n 의 입력, 추출 모듈을 그대로 활용하고 난독화된 자바스크립트의 복호화 성능



〔그림 12〕 자바스크립트 복호화 결과 실시간 그래프



〔그림 13〕 자바스크립트 자동 복호화 도구 전체 구성도

향상 및 실시간 모니터링과 리포팅 기능 추가를 통하여 네트워크 구간에서 실시간으로 자동 복호화를 수행하게 된다.

3.5.1 입력 구간 실행 흐름

입력 구간은 복호화 작업을 수행할 데이터의 입력 형식을 판단하고 입력 형식에 맞게 파싱 작업을 수행하게 된다. 만약 입력 값이 Interface 일 경우 run_nids 모듈을 통해 실시간 패킷 탐지를 수행하며, 입력 값이 URL 일 경우 URL fetch 를 통해 해당 웹 페이지에 존재하는 데이터들을 파싱하게 된다. 입력 값이 File 형식인 경우 MAGIC 라이브러리를 통해 파일 헤더를 파악하여 파일 형식에 맞는 파싱 모듈

을 호출하게 된다.

3.5.2 복호화 구간 실행 흐름

복호화 구간은 난독화된 자바스크립트의 실질적인 복호화를 수행하는 구간으로 입력 구간에서 파싱된 자바스크립트에 함수 오버라이딩을 위하여 재 정의된 자바스크립트 코드를 삽입하여 Spidermonkey 를 통해 실행하게 된다. 실행전에 Window Script Encoder로 난독화된 데이터에 대해서는 3.2에서 추가한 별도의 복호화 함수를 통해 복호화를 수행하게 되며, 복호화된 데이터는 level 변수 값이 1 씩 증가하게 되고, 복호화 작업 후 발생되는 텍스트에 난독화된 자바스크립트 코드가 포함되지 않을 때까지 재귀적으로 반복 작업을 수행하게 된다.

3.5.3 패턴 매칭 구간 실행 흐름

패턴 매칭 구간에서는 복호화된 자바스크립트 문자열에서 2.5.7에서 설명한 대로 기존에 정의된 악성 패턴 데이터들과의 매칭을 통해 악성행위를 발생시키는 패턴이 있는지 확인하게 된다. 정의된 악성 패턴들은 〔그림 14〕와 같이 Impact 변수에 위험도에 따라 일정한 값을 가지게 되고 만약 5이상의 Impact 값이 5번 이상 탐지될 경우 악성 코드로 정의되며 5 이하의 Impact 값이 5번 이하로 탐지될 경우 의심스러운 코드로 정의가 된다.

```
if impact > 5:
    type = 'malicious'
elif impact > 0:
    type = 'suspicious'

if self.malicious > 5:
    intro = '[malicious:%d] %s'
elif self.malicious > 0:
    intro = '[suspicious:%d] %s'
```

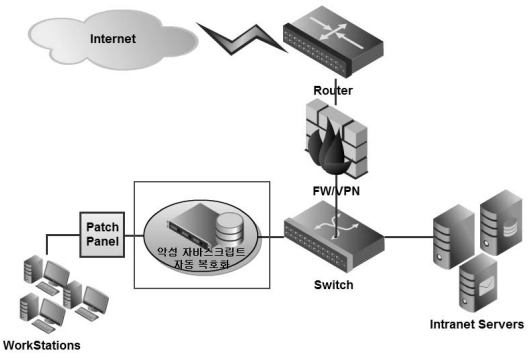
〔그림 14〕 악성 코드 탐지 임계치 값 비교

3.5.4 리포팅 구간 실행 흐름

리포팅 구간은 복호화 실행 과정에 포함되어 복호화 작업과 동시에 파일형태로 기록이 이루어지며 3.3에서 설명한대로 HTML 형식의 상세 보고서를 출력하기 위해서는 복호화 작업이 완료된 뒤에 수행한 복호화 작업의 정보를 담고 있는 모듈을 호출하여 리포트를 생성하게 된다.

IV. 적용 및 실행 결과

4.1 난독화 자바스크립트 자동 복호화 도구의 적용 방안



〔그림 15〕 제안 도구의 내부망 적용 화면

3장에서 제안한 난독화된 악성 자바스크립트의 자동 복호화 도구의 실제 적용을 위해 〔그림 15〕와 같이 기본적인 보안 솔루션이 구성된 사내망 내부에 약 10여대의 가상 클라이언트를 구성한 뒤, 네트워크 장비인 스위치의 미러링 포트를 통하여 클라이언트에서 전송되는 트래픽을 모니터링 할 수 있도록 환경을 구성하였다.

4.2 제안 도구의 적용 결과 분석

본 논문에서는 난독화된 자바스크립트의 복호화 가능 여부와 포함된 악성코드에 대한 탐지 여부를 중점적으로 확인해야 하기 때문에 다양한 난독화 방식의 자바스크립트 샘플 파일에 대한 복호화 테스트와 실제 환경에서의 악성 URL 접근 시 전송되는 악성 자바스크립트에 대한 탐지여부의 확인이 필요하다.

4.2.1 자바스크립트 난독화 방식에 대한 범용적인 복호화 가능 여부

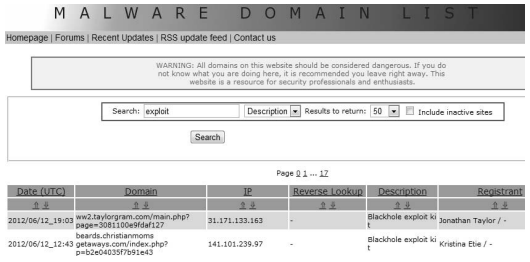
다양한 자바스크립트 난독화 방식에서 필수적인 요소가 되는 주요 자바스크립트 기능함수와 문자열 인코딩 기법에 대한 복호화 가능 여부와 제안 모델인 기존 오픈소스 Jsunpack-n에서 복호화가 불가능하던 코드에 대한 복호화 가능 여부, 난독화 전용 도구를 이용한 코드와 웹 공격용 툴킷에 대한 탐지 가능 여부를 확인하였다. 실험 결과 〔표 8〕과 같이 대부분의 자바

스크립트 난독화 기법에 대한 복호화와 이를 통한 악성코드의 탐지가 가능하였지만 jjencode 방식의 난독화 전용 도구와 Blockhole Exploit Kit에서 일부 복호화가 불가능한 코드가 존재하였다. 확인결과 jjencode 난독화 도구는 기존 난독화 기법과 다르게 2-3개의 문자열로 구성된 난독화 코드를 만들어내어 별도의 자바스크립트 인터프리터를 이용한 실행이 불가능하였으며, Blackhole Exploit Kit 에서는 Java Applet 취약점 공격 관련 코드가 자바스크립트 인터프리터에서 실행할 수 없기 때문에 탐지가 안 되는 것으로 확인되었다.

〔표 8〕 자바스크립트 난독화 방식의 범용적인 복호화 가능 여부 확인

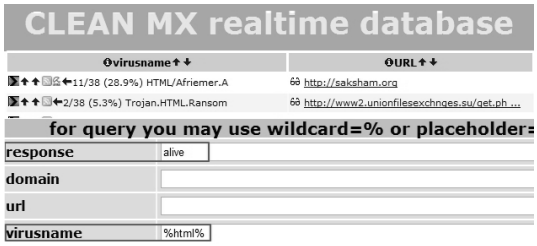
구분	상세 구분	결과
자바스크립트 기능함수/문자열 인코딩	Escape 함수	가능
	Eval 함수	가능
	Base64	가능
	Base85	가능
	Base62	가능
	XOR	가능
	8-BIT ASCII	가능
기존 Jsunpack-n에서복호화 불가능한 샘플에 대한 복호화 가능여부	window.location 을 이용한 분석환경 탐지 우회 코드	가능
	특정 객체명과 전역 변수 명을 이용한 분석환경 탐지 우회 코드	가능
	toString 메서드를 통한 분석환경 탐지 우회 코드	가능
	VBScript.encode,Jscript.encode 인코딩	가능
난독화 전용 도구를 이용한 코드	Dean Edward's JavascriptObfuscator	가능
	Free Javascript Obfuscator	가능
	Javascript Obfuscator	가능
	jjencode	불가
	Online Javascript compressor	가능
웹 공격용 툴킷 탐지	Target-Exploit	가능
	Smart pack	가능
	My poly sploit	가능
	multisploit	가능
	mypack-009	가능
	mypack-081	가능
	Mpack	가능
	Ice-pack-1	가능
	G-pack	가능
	Fire Pack -2	가능
	Neospolit	가능
	Blockhole Exploit Kit	일부 불가
	SEO SPLOIT PACK	가능

4.2.2 실제 환경에서의 자바스크립트 난독화 탐지 여부



〔그림 16〕 Malware Domain List 의 검색조건

실제 환경에서 난독화된 악성 자바스크립트를 확인하기 위해서는 악성URL 에 인위적으로 접속하는 과정이 필요하며 이를 위하여 실시간으로 악성코드 샘플을 수집하여 정보를 제공하는 사이트의 정보를 활용하여 테스트를 진행하였다. 악성코드 샘플을 제공하는 사이트들은 다수 존재하지만 본 실험 목적에 부합되는 정보를 제공해주는 사이트인 Malware Domain List〔18〕, CLEAN MX realtime database〔19〕 두 사이트에서 난독화된 자바스크립트를 포함하는 URL 정보를 수집하였다. Malware Domain List 에서는 〔그림 16〕과 같이 "Description" 행에서 "exploit" 이라는 문자열로 검색된 현재 활성화된 URL 들의 목록을 추출하였으며 CLEAN MX realtime database 에서는 〔그림 17〕과 같이 "Response" 행에 "alive" 문자열이 들어가고 "Virusname" 행에서 와일드카드를 이용하여 html 문자가 포함된 모든 결과를 확인할 수 있도록 "%html%" 문자열로 검색된 URL들의 목록을 추출하였다. 추출된 URL 정보는 약 80% 가량만이 실제



〔그림 17〕 Clean MX realtime database 의 검색조건

〔표 9〕 실 환경에서의 자바스크립트 난독화 탐지 여부

구분	Malware Domain List	Clean MX realtime database
Google Safe Browsing	89/112 (79%)	123/238 (51%)
McAfee Site Advisor	83/112 (74%)	122/238 (51%)
제안도구	92/112 (82%)	154/238 (64%)

접근이 가능하기 때문에 10일간 약 310개의 악성 자바스크립트가 포함된 URL에 접근을 시도하였다. 해당 실험을 기준으로 기존 악성 URL 탐지 도구인 Google Safe Browsing〔20〕과 McAfee Site Advisor〔21〕의 탐지 결과를 비교한 결과는 〔표 9〕와 같다.

실험 결과 "Malware Domain List" 의 악성 URL 에서는 Google Safe Browsing 에 비해 약 3%, McAfee Site Advisor 에 비해 약 8% 의 탐지율 향상을 확인하였으며, "Clean MX realtime database" 의 악성 URL 에서는 Google Safe Browsing, McAfee Site Advisor 에 비해 약 13% 의 탐지율 향상을 확인할 수 있었다. 수집된 악성 URL 중 사이트가 차단되거나 자바스크립트 난독화 코드가 포함되지 않은 경우를 제외하게 되면 기존 솔루션보다 탐지율이 더욱 향상된 것으로 확인할 수 있다.

V. 결론

자동화된 웹 공격용 툴킷의 발전으로 이제 악성코드는 웹상에서 대부분 유포가 이루어지고 있다. 또한 자바스크립트의 난독화 기법을 조금씩 바꿔가며 패턴 매칭 기반의 보안 솔루션들을 우회하여 지속적으로 대량 유포가 이루어지고 있기 때문에 이에 대한 한계점을 극복하기 위하여 난독화된 자바스크립트 코드에 대한 자동 복호화 방안이 반드시 필요한 상황이다. 이러한 점에서 본 논문을 통해 제안한 도구는 여러 자바스크립트 난독화 기법에 상관없이 범용적인 복호화 작업을 수행할 수 있기 때문에 패턴 매칭 기반의 보안 솔루션의 한계를 극복할 수 있는 좋은 대안이 될 수 있다. 제안 모델을 활용하기 위한 방안으로 기존의 IPS, 웹 방화벽과 같은 패턴 매칭 기반의 보안 장비와 결합하여 해당 도구를 통해 복호화된 데이터를 기존 보안 솔루션의 패턴을 통해 탐지하여 기존 보안 장비와의 결합을 통해 탐지율을 향상시킬 수 있는 방안도 존재하며, 미러링 포트를 연결하거나 외부 네트워크와의 접점에 모니터링 서버를 구축하여 별도의 보안 솔루션으로도 활용이 가능하다. 또한 허니팟/허니넷과 같은 의도적으로 설치해 둔 시스템의 구성 요소로서 시스템의 직접 감염을 통한 분석 절차 없이 악성코드가 포함된 URL 에 대한 탐지 작업을 수행할 수 있다.

제안 도구를 통해 난독화된 자바스크립트의 자동 복호화를 이용하여 기존 솔루션에서 탐지가 어려웠던 난독화된 자바스크립트에 포함된 악성코드에 대한 탐지 방안이 마련되었지만, 중소규모 이상의 기업 내부망과 같은 곳에 적용하기 위해서는, 대용량의 네트워크 트래픽을 처리하기 위한 병렬처리 기법의 적용이 필요하다. 또한 4.2.1에서 탐지가 불가능하였던 Java Applet을 이용한 악성코드 유포 방법과 같이 자바스크립트가 아닌 Java Applet 관련 취약점을 이용한 악성코드 유포 탐지에 대해서는 별도로 연구가 필요할 것으로 보인다.

본 논문에서 제작한 해당 프로그램은 Google Code Project 에 오픈소스로 등록하여 배포하였으며, 향후 지속적으로 개선시켜 나갈 예정이다. http://code.google.com/p/js-deobfuscator/

참고문헌

〔1〕 Jsunpack-n Source, https://code.google.com/p/jsunpack-n/source/browse/#svn%2Ftrunk
〔2〕 AVTEST, http://www.av-test.org/en/statistics/malware/
〔3〕 안랩 보안통계, http://www.ahnlab.com/kr/site/securitycenter/statistics/
〔4〕 Symantec Internet Security Threat Report Volume 16, http://www.symantec.com/about/news/resources/press_kits/detail.jsp?pkid=threat_report_16
〔5〕 The Ultimate Deobfuscator, http://securitylabs.websense.com/content/Blogs/3198.aspx
〔6〕 Shedding Light on the NeoSploit Exploit, http://labs.m86security.com/2011/01/shedding-light-on-the-neosploit-exploit-kit/
〔7〕 Likarish, P, Eunjin Jung, Insoon Jo,

"Obfuscated Malicious Javascript Detection using Classification Techniques", "Malicious and Unwanted Software (MALWARE) 2009 4th International Conference", pp. 47-54, 2009.

〔8〕 Younghan Choi, Taeghyoon Kim, Seokjin Choi, and Cheolwon Lee. "Automatic Detection for Javascript Obfuscation Attacks in Web Pages through String Pattern Analysis", In Proceedings of the 1st International Conference on Future Generation Information Technology (FGIT '09), pp. 160-172, 2009.

〔9〕 Zozzle: Low-overhead Mostly Static Javascript Malware Detection, http://research.microsoft.com/apps/pubs/?id=141930

〔10〕 Yung-Tsung Hou, Yimeng Chang, Tsuhan Chen, Chi-Sung Laih, and Chia-Mei Chen. "Malicious web content detection by machine learning", Expert Syst. Appl. 37, 1 (January 2010), pp. 55-60, 2010.

〔11〕 Alexander Moshchuk, Tanya Bragin, Damien Deville, Steven D. Gribble, and Henry M. Levy, "SpyProxy: execution-based detection of malicious web content", Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, pp. 1-16, 2007.

〔12〕 Marco Cova, Christopher Kruegel, and Giovanni Vigna. "Detection and analysis of drive-by-download attacks and malicious Javascript code", In Proceedings of the 19th international conference on World wide web (WWW '10), pp. 281-290, 2010.

〔13〕 Andreas Dewald, Thorsten Holz, Felix C. Freiling, "ADSandbox: sandboxing Javascript to fight malicious websites", Proceedings of the 2010 ACM Symposium on Applied Computing, pp 1859-1864, 2010.

〔14〕 Yara-project, http://code.google.com/p/yara-project/

〔15〕 Windows Script Decoder, http://www.virtualconspiracy.com/download/scrdec18.c

〔16〕 Js-beautify, https://github.com/einars/js-beautify

〔17〕 Google Chart Tools, https://developers.google.com/chart/

〔18〕 Malware Domain List, http://www.malwaredomainlist.com/

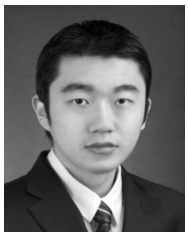
〔19〕 CLEAN MX realtime database, http://support.clean-mx.de/clean-mx/viruses

〔20〕 Google. Safe Browsing API. http://code.google.com/apis/safebrowsing/

〔21〕 McAfee Site Advisor, http://www. siteadvisor.com/

〔22〕 신화수, 문종섭, "악성코드 은닉사이트의 분산적, 동적 탐지를 통한 감염피해 최소화 방안 연구.", 정보보호학회논문지, 21(3), 89-100, 2011년

〈著者紹介〉



지 선 호 (Ji Sun Ho) 학생회원
2008년 2월: 수원대학교 컴퓨터학과 졸업
2010년 9월~현재: 고려대학교 정보보호대학원 정보보호학과 석사과정
〈관심분야〉 정보보호, 침입탐지, 역공학

김 휘 강 (Huy Kang Kim) 종신회원
1998년 2월: KAIST 산업경영학과 학사
2000년 2월: KAIST 산업공학과 석사
2009년 2월: KAIST 산업및시스템공학과 박사
2004년 5월~2010년 2월: 엔씨소프트 정보보안실장, Technical Director
2010년 3월~현재: 고려대학교 정보보대학원 조교수
〈관심분야〉 온라인게임 보안, 네트워크 보안, 네트워크 포렌식