웹 소스코드에 은닉된 Javascript URL 점검체계

박휘랑*, 조상일*, 박정규O, 조영호(교신저자)O
*대한민국 공군
O국방대학교 국방관리대학원 국방과학학과 컴퓨터공학/사이버전 협동전공
e-mail :lala16239@gmail.comO, yhcho94@gmail.comO

Detection System of Hidden Javascript URLs in Web Source Codes

Hweerang Park*, Sangil Cho*, JungKyu ParkO, Youngho ChoO
*Republic of Korea Air Force
OGraduate School of Defense Management, Korea National Defense University

● 요 약 ●

최근 웹 변조 공격은 대형 포탈, 은행, 학교 등 접속자가 많은 홈페이지에 악성 URL을 불법 삽입하여 해당 URL을 통해 접속자 PC에 자동으로 악성코드 유포하고 대규모 봇넷(botnet)을 형성한 후 DDoS 공격을 수행하거나 감염 PC들의 정보를 지속적으로 유출하는 형태로 수행된다. 이때, 홈페이지에 삽입되는 악성 URL은 탐지가 어렵도록 Javascript 난독화 기법(obfuscation technique) 등으로 은밀히 삽입된다. 본 논문에서는 웹 소스코드에 은닉된 악성 Javascript URL들에 대한 일괄 점검체계를 제안하며, 구현된 점검체계의 prototype을 활용하여 점검성능에 대한 시험결과를 제시한다.

키워드: 은닉 URL 탐지(Hidden URL Detection), 사이버 공격(Cyber Attack), 네트워크 보안(Network Security)

I. 서론

웹 변조 공격(web defacement attack)은 악의적인 목적으로 인터넷 홈페이지를 허가되지 않은 방법을 통해 무단 변경하는 일련의 공격을 말한다[1]. 과거에는 정치적인 목적 달성 또는 해킹 실력을 과시하기 위해 홈페이지 메인화면의 이미지 등을 무단으로 변조하는 공격 형태가 많았다. 최근의 공격사례를 살펴보면, 공격자는 웹 페이지(또는 소스코드)에 악성 URL을 삽입하고 해당 URL link를 통해 수많은 홈페이지 접속자 PC에 악성코드를 자동으로 은밀히 유포한다(Fig. 1. 참조). 감염된 PC들은 공격자(C&C)의 통제를 받는 봇넷(botnet)을 형성하여 DDoS 공격에 동원되거나, PC에 저장된 중요정보를 지속해서 공격자에게 유출하게 된다[2]. 이러한 이유로 악성 URL을 삽입하는 변조 공격에 대한 효과적 대응이 매우 중요하다.

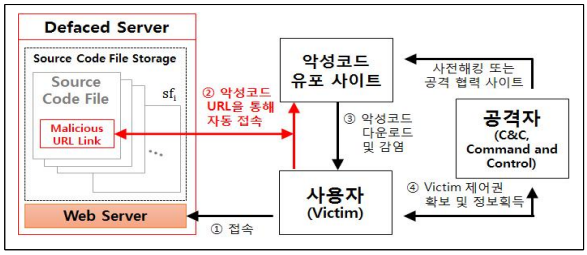

Fig. 1. 홈페이지에 무단 삽입된 URL에 의한 공격행위

악성 URL 삽입은 웹 소스코드에 그대로 보이도록 삽입되기보다는 관리자에 의해 쉽게 발견되지 않은 상태에서 오랫동안 공격을 할 수 있도록 은밀하게 삽입된다. 이때, 가장 많이 사용되는 방법이 Javascript 난독화(obfuscation) 기법이다. 난독화 기법은 원래 소스코드 보호를 위해 개발되었으나, 공격자가 자신의 악성 행위를 은폐할 목적으로 악용되기도 한다. 난독화 기법은 encoding obfuscation, randomization obfuscation, data obfuscation으로 분류되는데, 이 중에서도 Fig. 2.와 같이 악성 URL에 해당하는 Javascript code를 16진수로 변환하여 소스코드에 삽입하는 encoding obfuscation 방법이 가장 많이 사용된다[3]. Javascript 난독화 기법을 사용하면 악성 URL을 생성해내는 Javascript code가 서버에 있을 때는 난독화되어 은닉되어 있으므로 난독화 해석 기능이 없는 서버 측 정보보호체계(백신, 방화벽, IPS 등)에는 탐지되지 않는다. 또한, 해당 Javascript code가 담긴 웹 페이지에 사용자가 접속했을 때, 해당 code가 웹 브라우저에 의해 처리된 후에야 악성 URL로 복호화되어 활성화된 후, 사용자도 모르게 악성코드를 다운로드 받아 은밀히 설치되기 때문에 악성코드에 의한 피해를 인지하여도 이것이 Javascript 악성 URL과 연관이 있다는 것을 분석해내는 것은 일반 사용자에게는 매우 어려운 일이다.

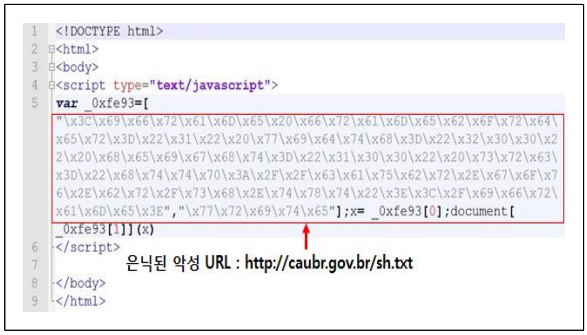

Fig. 2. Javascript 난독화기법으로 삽입된 악성 URL

Javascript obfuscation 기법 기반의 악성코드 탐지에 관한 연구로는 소스코드의 string pattern 분석이나 머신러닝 기법을 활용한 것이 있다[4, 5]. 이들은 난독화된 Javascript code들 중에서 악성코드인 것들만 선별하려고 노력하였으나, 탐지 정확도가 완전하지 않으며, 사용자 PC에 점검체계가 반드시 설치되어야 한다는 점과 사용자가 접근한 웹 페이지에 대해서만 점검이 이루어진다는 제한사항이 있다.

한편, 악성 URL이 홈페이지 소스코드에 은밀히 삽입되는 시점은 두 가지의 경우이다 (Fig. 3. 참조). 참고로, 본 연구에서는 소스코드에 명시적으로 노출된 악성 URL에 대한 점검은 어렵지 않은 것으로 보고 연구범위에서 제외한다.

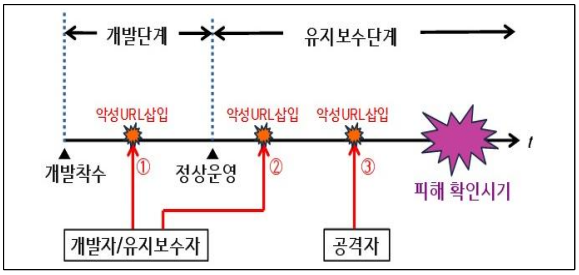

Fig. 3. 웹 응용체계(소스코드)에 악성 URL 삽입 시점

우선, 악성 URL은 웹 응용체계의 개발단계에서 소스코드에 삽입될 수 있다. 예를 들어, 개발자가 공격자일 경우 의도적으로 삽입하는 경우와 개발자가 개발을 용이하게 수행하기 위해 인터넷에 공개된 감염 소스코드를 의심 없이 사용(copy & paste)하는 경우이다. 본 논문에서는 연구범위의 명확성을 위해 전자는 고려하지 않는다. 다음으로, 악성 URL은 웹 응용체계가 개발되어 서버에 설치된 이후 정상운영 중에 삽입될 수 있다. 예를 들어, 앞의 설명과 유사하게 체계개발자(또는 유지보수자)가 응용체계를 유지보수하는 과정에서 외부 소스코드 (악성코드 감염)를 의심 없이 사용하는 경우와 외부공격자가 서버, 미들웨어, 웹 응용체계의 취약점을 통해 악성 URL을 은밀히 삽입하는 경우이다.

악성 URL이 은밀히 삽입되는 시점을 봤을 때, 악성 URL에 의한 피해를 최소화하기 위한 소스코드 점검 시점을 다음의 세 가지 시점으로 본다: ① 개발 후 서버에 설치되는 시점, ② 유지보수가 완료된 후 서버에 설치되는 시점, ③ 가능한 실시간으로 또는 주기적으로 서버의 소스코드를 대상으로 점검하는 것이다; 앞의 두 경우는 개발자의 실수에 대응하는 차원이고 마지막의 경우는 공격자에 의한 삽입에 대응하여 피해를 최소화하기 위해서이다. 또한, 점검대상은 개발서버(또는 PC)와 운영 서버의 전체 웹 소스코드를 대상으로 하여야 한다.

따라서, 본 논문에서는 웹 소스코드에 은닉된 Javascript 악성 URL을 일괄 점검하는 점검체계를 제안한다. 제안체계는 서버에서 실시간 또는 주기적으로 운용하도록 하고, Javascript code 상에 은닉된 URL을 찾아 악성 URL Repository와 유사도 비교 알고리즘을 활용하여 악성, 의심, 정상으로 판정한다.

II. 제안 점검체계 설계

1. 점검절차 및 알고리즘

본 논문에서 제안된 체계는 1) Javascript 블록 분석 및 URL 추출(STEP 1), 2) 은닉 URL 탐지(STEP 2), 3) 악성 및 의심 URL

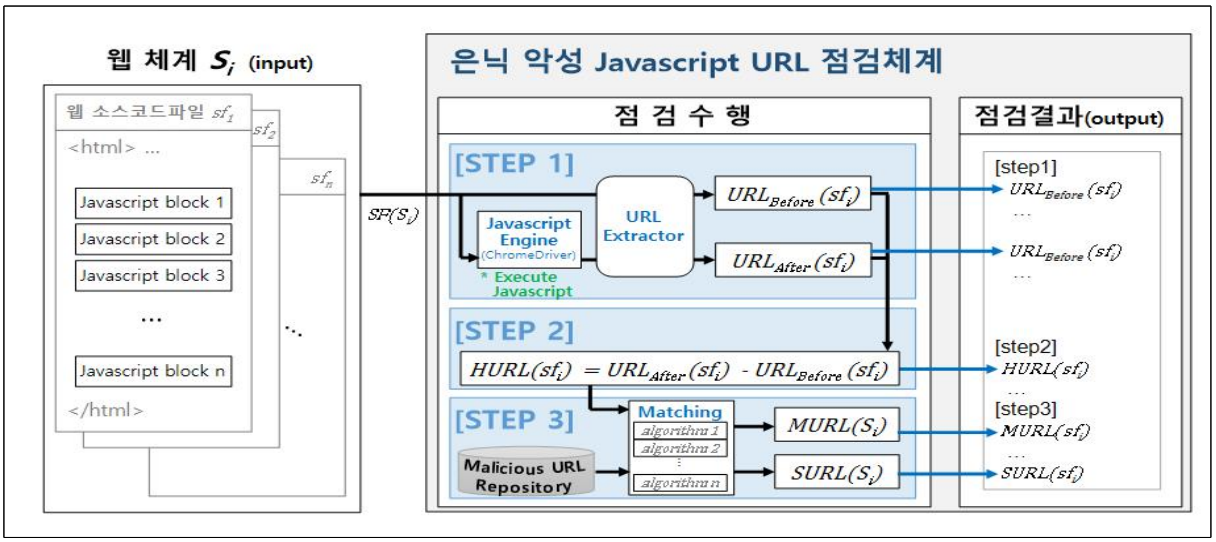

Fig. 4. 제안 점검체계 설계 및 동작

탐지(STEP 3)의 3단계로 동작하여 소스코드에 은닉된 악성 및 의심 URL을 점검하며, 각 단계의 세부 동작은 다음과 같다([그림 4] 참조).

1.1 STEP 1 : Javascript 블록 분석 및 URL 추출 (Extracting URLs from Javascript blocks in web source codes)

점검 대상인 웹 응용체계를 S_i라 하자. S_i를 구성하는 특정 소스파일을 sf_j라 하고, S_i를 구성하는 전체 소스파일 집합을 $SF(S_i)$라 하자. 이때, 소스파일의 개수가 n이면, $SF(S_i) = \{sf_1, sf_2, \cdots, sf_n\}$이다. STEP 1에서는 $SF(S_i)$를 input으로 하여 sf_j의 Javascript 블록에 명시적으로 코딩된 URL 목록을 모두 추출하여 저장한다. 이를 $URL_{Before}(sf_j)$라 하자. 이후 점검체계의 Javascript engine을 활용하여 sf_j의 Javascript 블록을 parsing 하여 전체 URL 목록을 추출한다; 이를 $URL_{After}(sf_j)$라 하자. Javascript engine은 모든 웹 브라우저에 내장되는데, 별도 점검 도구 개발을 위해 Javascript API를 제공하고 stand- alone으로 동작하는 ChromeDriver[6]을 활용한다.

1.2 STEP 2 : 은닉 URL 탐지(Detecting Hidden URLs)

STEP 1에서 분석 추출한 두 가지 URL 목록 $URL_{Before}(sf_j)$과 $URL_{After}(sf_j)$를 활용하여 소스코드 sf_j에 은닉된 Javascript URL 목록 $HURL(sf_j)$을 **(1)**과 같이 생성한다.

$$HURL(sf_j) = URL_{After}(sf_j) - URL_{Before}(sf_j) \quad \textbf{(1)}$$

이와 같은 방법으로 S_i의 모든 소스파일에 대해 점검하면, S_i에 은닉된 Javascript URL 목록 $HURL(S_i)$은 **(2)**와 같다.

$$HURL(S_i) = \bigcup_{j=1}^{n} HURL(sf_j) \quad (2)$$

1.3 STEP 3 : 악성 및 의심 URL 탐지(Detecting malicious and suspicious URLs)

STEP 2에서 탐지한 은닉 URL 목록인 $HURL(S_i)$와 악성 URL repository에 저장된, 알려진 악성 URL들을 matching algorithm으로 비교하여, 정보체계의 악성 URL의 포함 여부를 점검한다. 이때, 점검된 악성 URL 목록을 $MURL(S_i)$이라 하자. 악성 URL repository는 제안체계에 직접 구축하거나 분리된 별도 체계로 구축하여 연동할 수 있다. 한편, $MURL(S_i)$의 각 URL_k에 대해 유사도 측정 알고리즘(matching algorithm)으로 유사도를 측정한 결과값 SV_k는 0과 1 사이의 값으로 나타낼 수 있으며, SV_k와 관리자가 설정하는 두 개의 경계값 T_1(하위 경계값)과 T_2(상위 경계값)에 따라 해당 URL_k에 대해서 **(3)**과 같이 악성(m : malicious), 의심(s : suspicious), 정상(n : normal)으로 판정한다(이때, $0<T_1<T_2<1$).

$$D(URL_k) = \begin{cases} m & \text{if } T_2 \le SV_k \le 1 \\ s & \text{if } T_1 \le SV_k < T_2 \\ n & \text{if } 0 \le SV_k < T_1 \end{cases} \quad \textbf{(3)}$$

즉, 유사도 값이 1보다 비교한 결과 100% 일치하지 않으나 높은 수준으로 유사한 경우에는 추가적인 조사가 필요하므로 의심(s)으로 분류하고 의심 URL 목록 $SURL(S_i)$에 포함하여 관리한다. 두 경계값 T_1과 T_2는 점검체계 운영을 통해 정확도가 향상되도록 관리자에 의해 조정될 수 있다. 한편, 악성 URL repository는 KISA와 같은 신뢰성이 높은 정보보호 기관이나 Zone-H[7]과 같이 국제적으로 사이버침해사고에 대한 정보를 제공하는 홈페이지를 통해 지속해서 업데이트하여 관리한다.

III. 결론 및 향후 연구계획

본 논문에서는 Javascript obfuscation 기법을 악용하여 웹 응용체계 소스코드에 은닉된 악성 Javascript URL들에 대한 일괄 점검체계를 제안한다. 제안 점검체계는 웹 체계의 전체 소스코드에 대해, ① Javascript 블록 분석 및 URL 추출, ② 은닉 URL 탐지, ③ 악성 및 의심 URL 탐지의 3단계로 수행된다. 본 제안체계는 서버에 저장된 대량의 소스코드 파일들을 일괄 점검할 수 있도록 하여 은닉된 악성 URL에 의한 피해를 최소화하도록 한다.

향후 연구계획은 다음과 같다. 수많은 웹 소스코드를 대상으로 본 연구에서 제안한 점검 알고리즘의 정확도 및 속도를 측정하고 결과에 따라 성능 최적화 요소를 식별하여 개선할 예정이다. 또한, 실제 홈페이지 소스코드 서버에서 실시간 또는 주기적인 점검 운용이 가능하도록 보완하고 은닉 URL이 탐지된 경우 즉시 차단 및 격리하여 피해를 최소화하도록 개선할 예정이다.

References

[1] G. Davanzo et al., "Anomaly detection technique for a web defacement monitoring service," Expert Systems with Applications, 2011, pp. 12521-12530.

[2] S. Khattak et al., “A Taxonomy of Botnet Behavior, Detection, and Defense,” IEEE Communications Survey & Tutorials, Vol. 16, No. 2, Second Quarter, 2014, pp. 898 - 924.

[3] W. Xu et al., "The Power of Obfuscation Techniques in Malicious Javascript Code: A Measurement Study," Int. Conf. Malicious and Unwanted Software, 2012, pp. 9-16.

[4] W. Xu et al., "JStill : Mostly Static Detection of Obfuscated Malicious Javascript Code," CODASPY, USA, 2013, pp. 117-128.

[5] C. Curtsinger et al., "Zozzle: Fast and Precise In-Browser

Javascript Malware Detection," USENIX SEC 11, 2011.
[6] ChromeDriver, Available at http://chromedriver. chromium.org/home.
[7] Zohn-H, Available at http://www.zone-h.org.