악성 자바 스크립트를 탐지하는 분석 엔진

추현록, 정종훈, 임채태
한국인터넷진흥원
e-mail:hlchu@kisa.or.kr, jjh2640@kisa.or.kr, chtim@kisa.or.kr

The Analysis Engine for Detecting The Malicious JavaScript

Hyun-lock Choo, Jong-Hun Jung, Chae-Tae Im
Korea Internet&Security Agency

요 약

JavaScript는 AJAX와 같은 기술을 통해 정적인 HTML에 동적인 기능을 제공하며 그 쓰임새는 HTML5 등장 이후 더욱 주목받고 있는 기술이다. 그와 비례하여 JavaScript를 이용한 공격(DoS 공격, 기밀정보 누출 등) 또한 큰 위험으로 다가오고 있다. 이들 공격은 실제적인 흔적을 남기지 않기 때문에 JavaScript 코드 상에서 악성 행위를 판단해야 하며, 웹브라우저가 JavaScript 코드를 실행해야 실제적인 행위가 일어나기 때문에 이를 방지하기 위해선 실시간으로 악성 스크립트를 분별하고 파악할 수 있는 분석 기술이 필요하다. 본 논문은 이런 악성 스크립트를 탐지하는 분석엔진 기술을 제안한다. 이 분석 엔진은 시그니쳐 기반 탐지 기술을 이용한 정적 분석과 행위 기반 탐지 기술을 사용하는 동적 분석으로 이루어진다. 정적 분석은 JavaScript 코드에서 악성 스크립트 코드를 탐지하고 동적 분석은 JavaScirpt 코드의 실제 행위를 분석하여 악성 스크립트를 판별한다.

1. 서론

기존의 Web Application 공격의 형태는 Drive-By-Download 형태로 악성 코드를 설치 후에 이루어지는 방법이었다. 하지만 근래에는 기존과 다른 방식인 악성 JavaScript를 이용한 DDoS 공격이나 사용자의 동의 없이 개인 정보 수집 등의 문제가 발생하고 있다. 이 악성 스크립트 공격이 크게 위협이 되는 요소는 다음과 같다.

- JavaScript는 자체적으로 별도의 목적코드가 생성된 Binary 형태로 존재하지 않는다. 그렇기 때문에 기존의 Malware, Worm 등을 차단/방지하는 IDS, IPS 기술들을 우회하고 있다.

- JavaScript의 코드 난독화는 사용자 코드를 보호가 위해서 자주 사용되는 기술이다. 악성 JavaScript는 이 기술을 통해 코드 자체만으로 탐지 기술들을 우회하고 있다.

- HTML5에서는 JavaScript를 이용하여 웹 브라우저 저장소에 접근할 수 있고 video, audio, canvas 등의 다양한 media 객체들을 제어할 수 있으며 ActiveX를 대체할 수 있는 기술로 부각되고 있다.

본 논문의 위에서 정리한 요소들을 반영하여, 악성 행위를 실시간으로 탐지하는 분석 엔진 기술을 소개하고자 한다.

분석 엔진은 정적 분석과 동적 분석으로 구성되어있다. 정적 분석은 시그니쳐 기반 탐지 기술을 사용한다. 여기서 시그니쳐의 의미는 악성 스크립트의 코드 패턴 정보를 가진 객체를 뜻한다. 그래서 정적 분석은 이 Signature를 이용하여 JavaScript 코드가 악성인지를 파악한다. 정적 분석은 알려진 악성 스크립트는 탐지는 할 수 있으나 새롭거나 변종인 악성 스크립트를 파악할 수 는 없다. 이를 보완하기 위해 동적 분석이 존재한다. 동적 분석은 행위 기반 탐지기술을 이용한다. 이 기술은 JavaScript 코드가 실제 동작 중에 일어나는 행동을 불려진 API 흐름으로 파악하여 악성 스크립트를 판별하는 기술이다. 실제로 악성 스크립트의 행동의 완성은 시스템에 접근할 수 있는 JavaScript API를 통해서 이루어지기 때문에 코드 난독화나 다형화를 통해 문법은 변경은 될 수 있으나 그 행위를 결정하는 API의 흐름은 변경되지 않는 것이다[11].

우리는 분석 엔진에 대해서 나머지 장에서 자세한 설명을 진행할 것이다. 2장의 관련 연구에서는 분석엔진의 정적 분석과, 동적 분석에서 사용하는 상세 기술에 대해서 설명하고, 3장에서는 분석 엔진에 대한 전반적인 설명을 할 것이다. 4장의 연구 동향에서 악성 스크립트를 탐지하는 연구들에 대해 설명 하고, 마지막인 5장에서는 이 논문의 결론과 향후 연구로 마무리한다.

2. 관련 기술

분석 엔진에서 사용할 요소 기술에 대한 설명을 한다.

2.1 Conjunction 패턴

Conjunction 패턴[1]은 입력되는 여러 문서들로부터 후보 Token 들을 구성하고 전체 문서들에서 사용 횟수를 기준으로 Token들을 정제해 재구성한 Token들의 묶음이다. 그래서 생성된 패턴에는 입력된 모든 문서에서 가장 많이 사용 된 Token들이 남아있게 된다.

2.2 YARA

YARA는 Malware를 식별하고 구별하는 기술로서 이용된다. 가장 큰 특징이라고 한다면 다양한 표현을 담을 수 있는 Rule Set을 생성이 가능하다. 일반적인 텍스트 문자열, Hex 문자열 그리고 정규표현식을 담을 수 있으며, 또한 각 문자열들을 조건식을 통해 구성하여 좀 더 세세한 표현을 담을 수 있다. 그리고 2.0.0 이상의 버전에서는 속도도 많이 개선되었다[2].

이 후, 악성 스크립트의 Conjunction Pattern를 담고 있는 YARA Rule Set을 “Signature”라고 명한다.

2.3 JavaScript Engine

JavaScript Engnin이란 ECMA-262[3]의 표준을 지원하는 JavaScript Interperter이다. JavaScript Engine에서 Javascript가 실행이 되면 코드에서 사용되었던 구문들이 해석 되고 실행되어 실제로 호출이 되는 함수 이름과 그 함수에 입력되는 정보를 확인할 수 있다. 이를 이용하여 스크립트의 수행 정보를 추적할 수 있다. 참고로 JavaScript Engine은 DOM API[4]를 지원하지 않는다. 그렇기 때문에 DOM API를 추가적으로 Porting이 되어야 한다.

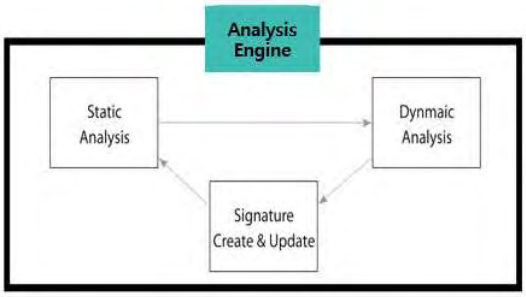
이 후, 이 JavaScript Engine의 스크립트 수행 정보를 “Call Trace”라 명하고 탐지에 사용되는 악성 스크립트의 Call Trace를 “Call Trace Signature”라고 명한다.

2.4 SimHash

SimHash는 LSH(Local Sensitive Hashing)을 이용한 유사도 분석 방법이다. LSH는 일반적인 암호학에서 사용하는 Hash 함수와는 다르게 유사한 항목에 대한 충돌을 회피하기 보단 최대화 하는 것이다[5]. 즉, 비슷한 항목에 대해서는 비슷한 결과 값을 생성하는 것이다. 이 함수를 통해 입력 값의 크기에 상관없이 일반적인 Hash 함수의 결과물처럼 비트 형태의 배열로 FingerPrint를 생성하고 이를 Hamming Distance[6]를 이용하여 유사도를 측정 하는 것이다[7].

3. 분석엔진

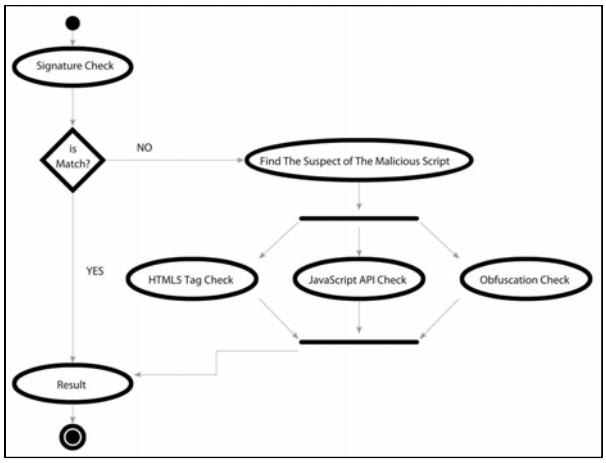
정적 분석에서 1차 악성 스크립트를 탐지한다. 탐지될 경우, 검사한 JavaScript 코드에 Signature의 악성 스크립트의 코드 패턴이 발견됨을 뜻한다. 탐지되지 않을 경우, 악성이 의심스러운 JavaScript를 분별하여 동적 분석에서 2차 악성 스크립트 탐지를 시도한다. 2차 탐지될 경우, 동적 분석의 탐지 결과가 Signature에 반영되어 추후 정적 분석에서 사용할 수 있게 하는 것이 분석 엔진의 흐름이다.



(그림 1). 분석 엔진 Flow

3.1 정적 분석

정적 분석에서는 그림 2와 같은 흐름으로 진행된다.



(그림 2). 정적 분석 Flow

3.1.1 Signature 검사

Signature를 통해 이전에 발생된 악성 스크립트의 코드 패턴이 JavaScript 코드에 존재하는지 확인한다. YARA를 통해 패턴에 포함된 Token들의 코드 상의 위치 정보를 확인하고 이를 이용하여 악성 JavaScript 코드를 추적한다. 추적된 악성 스크립트 코드는 추후 제거되거나 안전한 코드들로 대체 되는 등의 악성 요소를 제거하는데 사용된다.

3.1.2 악성이 의심스러운 스크립트 판별

Signature 검사를 통해 탐지되지 않는 스크립트 중에서 동적 분석이 필요한 스크립트는 존재한다. 새로운 공격 스크립트이거나 변종 스크립트 일 경우 정적 분석을 통해 발견되지 않을 수 있다. 이런 스크립트는 의심스러운 부분들, 아래 설명하는 요소들을 확인하여 이상이 없는지를 확인한다.

의심스러운 부분이 없을 경우, 안전한 스크립트 코드라고 인식한다.

3.1.2.1 HTML5 태그 검사

HTML5의 새로운 태그들의 등장으로 기존의 HTML4에서 적용되었던 탐지 기술을 회피할 수 있다. OWASP 2013에서 핫 이슈가 되었던 XSS(Crioss-site scripting)가 video나

audio 등의 HTML5 새로운 태그를 이용하여 공격할 수 있는 가능성이 제기되었다[8].

이런 새롭게 등장한 태그들이 스크립트 코드에서 사용되었는지 확인한다.

3.1.2.2 JavaScript API 검사

실제 악성 스크립트에서 자주 사용하는 JavaScript API가 존재한다. 대표적으로 eval, document.write 함수 등이 공격에 자주 사용된다. 이는 악성 스크립트가 악성 행위를 하기 위해서는 최종적으로 JavaScript API를 통해서만 완료되기 때문이다.

이런 공격에 자주 사용되는 API가 사용되었는지를 확인한다.

3.1.2.3 난독화 검사

난독화 기법은 실제 악성 스크립트에서 탐지 기술을 회피하기 위해 많이 사용되는 기술이다[9]. 그 종류도 다양하기 때문에 Signature로 관리되기 힘든 부분이다. 난독화에 대한 판단은 스크립트 코드의 N-Gram(특정 바이트의 반복적인 사용 횟수), Entropy(구성된 바이트들의 분포도), Word Size(문자열의 길이)를 통해 확인한다[10].

난독화가 발견된 스크립트 코드는 동적 분석으로 필히 진행할 수 있도록 한다.

3.2 동적 분석

3.1.2에서 분류된 악성이 의심스러운 스크립트를 JavaScript Engine에서 실행시켜 Call Trace를 생성한다. 이 생성된 Call Trace와 Call Trace Signature를 비교해 악성 스크립트를 탐지 한다.

3.2.1 Call Trace의 생성

앞서 정의한대로 Call Trace는 JavaScript Engine의 스크립트 수행 정보이다. 이 정보에는 JavaScript API와 DOM API 이름 정보와 그리고 API에 입력되는 정보들이 표현된다. 이 후, JavaScript API와 DOM API를 Native API라고 칭한다. 사용자 함수 이름과 그 함수에 입력되는 정보는 필요에 따라 쉽게 변조할 수 있다. 하지만 핵심적인 악성 행위를 하는 Native API 이름과 입력된 데이터는 변경되지 않는다. 난독화 등의 기법으로 구문은 변경하지만 그 행위를 변경하지는 않는 것이다[11].

악성 코드의 위치 정보를 파악하기 위해서 Native API를 Call하는 스크립트 코드의 위치 정보를 기록한다. 이 기록 정보는 추후 악성 요소 제거와 Signature 생성 및 갱신에 사용된다.

3.2.2 Call Trace 비교

Call Trace와 Call Trace Signature의 비교는 앞서 밝힌 SimHash를 이용하여 비교한다. Call Trace와 Call Trace Signature의 크기에 상관없이 두 데이터의 유사도를 분석해 비교한다. Call Trace가 Call Trace Signature와 SimHash 유사도가 일정 이상 높으면 Call Trace Signature 해당되는 악성 스크립트의 변종임을 탐지 한 것이다. 그렇지 않는 경우는 안전한 스크립트라고 인식한다.

3.3 Signature 생성 및 갱신

Call Trace 비교에서 악성 스크립트를 탐지한다는 것은 정적 분석에서 탐지가 되지 않았다는 것이다. 그것은 Signature가 생성되거나 갱신이 필요한 것을 의미한다. 그렇기 때문에 동적 분석에서 탐지된 악성 스크립트의 코드와 위치 정보를 이용하여 Signature를 생성하거나 갱신을 한다. 생성과 갱신은 비슷한 방법으로 진행된다. 새로 발견된 악성 스크립트 코드와 기존의 악성 스크립트 코드를 이용하여 Conjunction 패턴을 생성하고 이를 Signature로 재구성하는 것이다.

4. 연구 동향

FLAX[12]와 같은 데이터 흐름을 분석하는 많은 탐지 방법이 존재한다[13][14]. 이들은 흐름을 분석하기 위해 JavaScript Interpreter을 사용[12]하거나 별도의 구문 해석을 하게 된다[13][14]. 하지만 이들 분석 방법들의 공통된 특징은 여러 실행 경로에 대한 분석을 하기 때문에 실시간 탐지에 부적합하다는 것이다.

IDS와 고정적인 패턴 정보에 의한 탐지 방법도 역시 처리 속도는 빠르나 새로운 공격에 취약하다는 문제점이 있다[15].

분석 엔진은 위 방법들의 단점을 보안하고 장점을 결합하는 기술로서 제안하고자 한다.

5. 결론

실시간으로 탐지를 진행하기 위해선 JavaScript의 구문 해석 부분을 최소화해야 하고 적중률을 올리기 위해선 패턴 정보를 계속 업데이트를 해줘야 한다. 그렇기 때문에 분석 엔진의 구성을 정적과 동적 분석으로 구성하였다. 정적 분석에서는 과거에 발견된 악성 스크립트의 Signature를 이용한 빠른 매칭을 통해 탐지를 하고 동적 분석에서 새로운 악성 스크립트를 발견하기 위해 JavaScript Engine의 수행 정보를 통해 탐지를 하고 그 결과를 이용하여 Signature를 생성하거나 갱신하다.

하지만 동적 분석의 탐지의 중요한 요소인 Call Trace Signature 또한 과거의 발견된 악성 스크립트 정보를 만들어지기 때문에, 난독화나 다형화 된 변종 안성 스크립트에 대한 탐지는 가능하나 전혀 다른 새로운 방식의 악성 스크립트는 탐지가 불가능하다.

그리고 이 논문은 기술의 방향성에 대해 기술하였기 때문에, 실제적인 적용 사례에 대해서 미흡하다.

이 후, 위의 부족한 점을 포괄하는 것을 향후 연구로 계획 중이다.

ACKNOWLEDGMENT

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음 [14-912-06-002, 스크립트 기반 사이버 공격 사전 예방 및 대응 기술 개발]

참고문헌

[1] Newsome, James, Brad Karp, and Dawn Song. "Polygraph: Automatically generating signatures for polymorphic worms." Security and Privacy, 2005 IEEE Symposium on. IEEE, 2005.

[2] YARA Documentation,

http://yara.readthedocs.org/en/latest/index.html
[3] ECMA-262 “EMCAScript Langauge Specification”, http://www.ecma-international.org/publications/standards/Ecma-262.htm
[4] Document Object Model, http://www.w3.org/DOM/
[5] A. Rajaraman and J. Ullman (2010). "Mining of Massive Datasets, Ch. 3.".
[6] Hamming, Richard W. "Error detecting and error correcting codes." Bell System technical journal 29.2 (1950): 147-160.
[7] Charikar, Moses S. "Similarity estimation techniques from rounding algorithms." Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. ACM, 2002.
[8] Dong, Guowei, et al. "Detecting cross site scripting vulnerabilities introduced by HTML5." Computer Science and Software Engineering (JCSSE), 2014 11th International Joint Conference on. IEEE, 2014.
[9] Xu, Wei, Fangfang Zhang, and Sencun Zhu. "The power of obfuscation techniques in malicious JavaScript code: A measurement study." Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on. IEEE, 2012.
[10] Choi, YoungHan, et al. "Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis." Future Generation Information Technology. Springer Berlin Heidelberg, 2009. 160-172
[11] Lee, Jusuk, Kyoochang Jeong, and Heejo Lee. "Detecting metamorphic malwares using code graphs." Proceedings of the 2010 ACM symposium on applied computing. ACM, 2010.
[12] Saxena, Prateek, et al. "FLAX: Systematic Discovery of Client-side Validation Vulnerabilities in Rich Web Applications." NDSS. 2010.
[13] Chugh, Ravi, et al. "Staged information flow for JavaScript." ACM Sigplan Notices. Vol. 44. No. 6. ACM, 2009.
[14] Xie, Yichen, and Alex Aiken. "Static Detection of Security Vulnerabilities in Scripting Languages." USENIX Security. Vol. 6. 2006.
[15] Chowdhary, Mahak, Shrutika Suri, and Mansi Bhutani. "Comparative Study of Intrusion Detection System." (2014).