ReDoS 취약점 탐지 도구의 동향 분석 및 개선을 통한 취약점 분석 연구

이동하* 박성진** 강성현*** 최지혁*** 김찬호*** 김민욱**** 이수영****

*가천대학교(학부생), **LIG넥스원(연구원), ***LIG넥스원(연구원), ***가천대학교(학부생), ***AI-Ver.Se labs(연구원), ****고려대학교(학부생), ****한국정보보호교육센터(선임연구원)

Analysis and Improvement of ReDoS Vulnerability Detection Tools for Enhanced Vulnerability Analysis

Dong-Ha Lee* Sung-Jin Park** Seong-Hyeon Kang*** Ji-Hyeok Choi*** Chan-Ho Kim*** Min-Uk Kim**** Su-Young Lee****

*Gachon University(Undergraduate student), **LIG Nex1(Rearch Engineer), ***LIG Nex1(Rearch Engineer), ***Gachon University(Undergraduate student), ***AI-Ver.Se labs(연구원), ****Korea University(Undergraduate student), ****KISEC(Rearch Engineer)

요 약

정규 표현식 서비스 거부(ReDoS) 취약점은 비효율적인 정규 표현식(Regex) 설계 때문에 발생하는 처리 과정의 지연으로 인해 소프트웨어 애플리케이션에 상당한 보안 위험을 가져올 수 있다. 이 논문에서는 정적 및 동적 분석 방법의 장점을 결합한 하이브리드 형태의 새로운 ReDoS 취약점 탐지 방법을 제안한다. 처음에는 정적 분석을 사용하여 신속하게 정규식의 안전성을 평가하고, 잠재적 취약점이나 모호한 패턴을 발견하면 동적 분석으로 넘어가서 핫스팟 기반 성능평가를 진행한다. 문자열의 길이를 조작하여 처리 시간의 증가율을 측정함으로써 ReDoS 취약점을 실질적으로 평가한다. 이 방법을 통해 Node.js 프로젝트에서 취약한 정규식 246개를 발견했으며, CVE-2023-43646을 발급받았다. 하이브리드 접근 방식은 정적 분석의 즉각성과 동적 분석의 심층성을 결합하여 ReDoS 취약성 탐지의 정확도와 효율성을 향상시키는 방법이다. 연구 결과는 이 방법론이 ReDoS 취약점을 신속하게 탐지하는 것뿐만 아니라 실시간 소스 코드 크롤링 및 자동화된 보고서 작성 프로세스를 통해 광범위한 오픈 소스 보안 강화에 기여할 수 있다는 것을 확인시켜 준다. 이 연구는 현대 소프트웨어에서 정규식으로 인한 취약점 문제를 해결하는 것의 중요성을 강조하고, ReDoS 공격과 관련된 위험을 줄이려는 실무자와 연구자들에게 예방 조치를 제공한다.

I. 서론

정규 표현식(Regular Expression)은 문자열 탐색과 패턴 매칭 등을 위한 강력한 도구로 전 세계적으로 프로그래밍 언어(Python, Java, Javascript 등), 데이터베이스 쿼리 등에서 광범위하게 사용되고 있다. 이렇게 광범위하게 사용되고 있는 정규 표현식은 잘못 설계 된 경우 매칭 시간이 polynomial, exponent 하게 증가해 정규 표현식으로 인해 발생하는 서비스 거부 공격인 ReDos를 유발할 수 있다. 대표적으로 2016년에 StackOverflow가 ReDos 공격으로 인해 마비되었으며 2019년에는 Cloudflare의 서비스가 마비되기도 했다. 그러한 영향성에 불구하고 아직 정규 표현식에 대한 검증은 미비하다. 연구[1]에 따르면 Python, Java, Javascript 프로젝트를 검증한 결과 10000개 이상의 모듈과 Node.js 및 Python 핵심 라이브러리에서 수천 개의 Super linear 정규 표현식을 발견하였다.

이에 따라 ReDos 취약점을 탐지하기 위해 많은 연구가 이루어졌고 크게 Static[2], Dynamic[3] 그리고 Hybrid 방식[4]-[5]으로 나누어진다. Static 방식은 사전에 ReDos 취약점 패턴을 정의한 후 그 것을 기반으로 탐지를 수행하는 것이고 Dynamic 방법론은 실제 테스팅을 통해 수행 시간을 측정하여 ReDos 취약 여부를 판정하는 것이다. 최근에는 Hybrid 방식[4]-[5] 이 제안되었으며 Static 과 Dynamic 방법론을 함께 사용해 높은 정확도를 보인다. 본 논문에서는 Hybrid 방법론에 기반한 ReDos 취약점 분석 방법을 제시한다. 초기 단계에서 패턴 기반 정적 분석을 활용하여 정규식의 안전성을 신속

히 평가하고, 위험성이 확인되거나 불분명할 경우 위험한 것으로 분류하여 동적 분석을 진행한다. 동적 분석에서는 처리 시간을 최대화하는 핫스팟을 탐색하고, 이를 바탕으로 성능 분석을 통해 문자열 길이를 조정하면서 처리 시간의 증가율을 평가한다. 이 과정을 통해, 문자열 길이를 증가시켜가며 ReDoS 취약성의 유의미한 평가가 가능한 방법론을 적용하였으며 제시한 방법론을 적용하여 node.js 프로젝트에 대해 분석을 수행해 246개의 취약한 정규식을 발견하였고 제보하여 CVE-2023-43646을 발급 받았다.

II. 기존 ReDos 탐지 연구 분석

2.1 기존 연구 분석

정규 표현식의 ReDoS(Regular Expression Denial of Service) 취약점은 정보 보안 분야에서 지속적으로 주목받아온 문제 중 하나로 다양한 연구들이 이 취약점의 탐지 및 대응 방안을 모색해왔다.

ReScue[3]에서는 정규식의 ReDoS 취약점을 강조하기 위해 그레이박스 분석 기술에 기반한 새로운 방법론을 제시하였다. 이 방법론은 주어진 정규식에서 취약한 부분을 자동으로 식별하고, 이를 활용해 ReDoS 공격 문자열을 생성하는데 초점을 맞추고 있다. 첫 번째 단계에서는 유전알고리즘 검색을 활용하여 가능한 모든 문자열 조합 중 가장 적합한 시드 문자열을 생성한다. 다음 단계에서는 이 시드 문자열을 기반으로 또 다른 유전알고리즘 검색을 통해 시드를 활성화 시킨다. 마지막 단계에서는 활성화된 시드 문자열과 정규식 전용 알고리즘을 결합하여 검색 시간이 최대화되는 문자열을 생성한다. 이 방법론을 오픈소스 프로젝트에 적용하여 29,088개의 실용적인 정규식을 분석했을 때, 기존의 방법론에 비해 약 49% 더 많은 공격 문자열을 찾아내는 성과를 보였다. 이를 통해 ReScue는 소프트웨어 개발 초기 단계에서 개발자나 제3자 소프트웨어 품질 팀이 ReDoS 보안 취약점을 효과적으로 검사하고 대응할 수 있게 되었다.

ReDoSHunter 논문[4]에서는 정규식의 ReDoS 취약점을 탐지하기 위한 취약성 패턴 기반의 방법론을 상세하게 제시하였다. 이 방법론은 NQ, EOD, EOA, POA, SLQ라는 다섯 가지 주요 취약성 패턴을 중심으로 개발되었다. 각 패턴은 문자열의 길이와 관련하여 다양한 시간 복잡도 문제를 야기하는 특성을 가지고 있다. ReDoSHunter는 이러한 패턴들을 사전에 정확하게 파악하고, 이에 기반하여 취약성을 유발할 수 있는 공격 문자열을 생성하는 메커니즘을 포함하고 있다. 생성된 문자열은 정규식의 취약점을 정밀하게 검사하며, 실제 취약점이 발견되면 그 문자열을 활용하여 실시간 공격 시나리오를 동적으로 검증한다. 이 과정을 통해 ReDoSHunter는 정규식의 ReDoS 취약점을 더욱 정밀하게 탐지하고, 이를 활용한 실제 공격의 가능성까지 평가할 수 있게 되었다.

Effective ReDoS Detection 논문[5]에서는 ReDoS 취약점의 식별 및 공격 문자열 생성에 중점을 둔 방법론을 체계적으로 제시하였다. 초기 단계에서는 정적 분석을 통해 ReDoS 취약성이 의심되는 정규식 후보를 체계적으로 탐색한다. 이후 이러한 후보 정규식의 특징과 구조를 분석하여, 그에 기반한 공격 문자열을 시리즈로 생성한다. 이 방법론의 핵심은 정규식의 구조와 특징을 깊이 분석하여 공격 가능성이 있는 문자열을 더욱 효과적으로 생성하는 것에 있다. 이렇게 생성된 공격 문자열은 실제 시스템 환경에서 ReDoS 취약점을 활용한 공격에 얼마나 효과적인지를 검증하는데 사용되며, 이를 통해 시스템의 보안성을 근본적으로 강화할 수 있게 되었다.

2.2 기존 연구 한계

[2-1]에서 다룬 기존 연구에는 ReDos 탐지에 대해 여러 가지 한계점을 안고 있다.

ReScue와 같은 그레이박스 분석 기술을 사용하는 접근법에서는 유전알고리즘 검색 알고리즘을 활용하여 시드를 생성하고 활성화하는 과정을 거친다. 이러한 과정에서 발생하는 무의미한 테스트 케이스의 반복 실행은 효율성의 저하를 초래한다. 또한, 동적 검사의 시간 소모가 크며, ReDoS 취약점의 식별 기준에 대한 모호성이 존재한다.

ReDoSHunter는 정규식의 취약성 패턴을 기반으로 공격 문자열을 생성하는 방식을 취하고 있다. 이 접근법에서는 많은 수의 공격 샘플 문자열을 동적으로 실행해야하기에 처리 시간이 상당히 길어질 수 있는 문제점이 있다. 더불어, 임계값 설정에 따른 취약점 판단의 주관성도 무시할 수 없는 문제로 다가온다.

Effective ReDoS Detection의 경우, 정적 분석을 통해 ReDoS 취약점의 후보를 식별하는 방법을 제시한다. 그러나 이 방법 또한 동적 분석단계에서 모든 샘플을 실행한다는 점에서 처리 시간이 길어진다는 문제를 안고 있으며, ReDoS 취약점의 식별 기준의 모호함 문제를 완전히 해결하지 못한다.

또한, 전통적인 ReDoS 탐지 연구들은 주로 취약점 존재의 가능성을 중심으로만 탐구되어 왔다. 이러한 연구들은 이론적 환경에서는 ReDoS 취약점을 정확히 탐지할 수 있지만, 실제 환경에서의 다양한 변수와 조건들을 고려하지 않았다. 이러한 이유에서 이론적으로 ReDoS 취약성을 탐지하는 것 이외에, 실제 운용되고 있는 OpenSource에서 ReDoS를 검사하고 탐지하는 부분에서의 효율성 및 적용 가능성에는 기존 탐지론의 한계가 존재했다.

III.새로운 ReDos 탐지 방법론

3.1 기존 한계를 극복한 ReDos 탐지 방법론

이러한 기존 연구들의 한계점을 극복하기 위해, 동적 분석의 장점과 정적 분석의 장점을 결합한 새로운 방법론을 제안한다. 우리의 새로운 방법론은 초기에 정적으로 처리 가능한 영역을 먼저 탐색하여 동적 분석의 효율성을 높이는 방향으로 설계되었다.

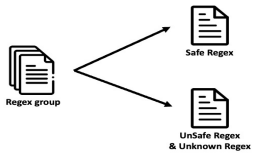

Figure 1. Safe/UnSafe 분류

[1번 단계]: 정적분석 / 기존의 ReDosHunter와 같은 방법으로 ReDoS가 발생할 수 있는 복잡도, Backtracking 유발 구문의 존재 여부, 문자열 길이 제한 등 다양한 조건들을 검토하여 Figure 1과 같이 정규식의 취약점 존재 가능성을 빠르게 판단한다. 이러한 고려사항들을 바탕으로 정규식을 safe와 unsafe로 분류하는데, 만약 safe와 unsafe의 구분이 명확하지 않은 경우에는 모두 unsafe로 간주한다.

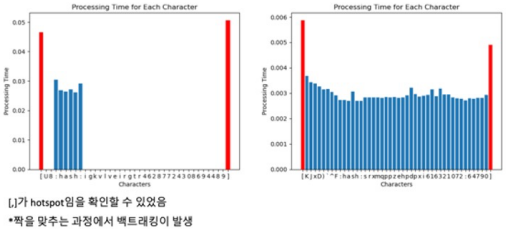

Figure 2. Hot-spot 검사

[2번 단계]: 동적 핫스팟 분석 / 정규식 처리시간이 극대화되는 케이스는, 정규식에 문자열이 매치하지 않지만, 매치하지 않는다는 것을 확인하는데 까지 많은 처리과정을 거치는 케이스다. (1) 정규식에 항상 참이 되는 샘플문자열을 하나 생성하고, 이 문자열을 랜덤하게 하나씩 지우거나, n번 반복하는 등의 노이즈 생성 과정을 거친다. (2) 과정 1번에서 노이즈가 발생했을 때 처리시간이 Figure 2와 같이 유의미하게 변화하는 부분(특정 문자: 핫스팟)을 발견하고 해당 부분에 여러차례 노이즈를 발생시켜 ReDos 가능성을 평가한다. *[2 번 단계]의 경우 기존의 연구보다 효율적인 테스트 케이스 생성이 가능하지만, 결국 동적으로 케이스를 테스트한다는 점은 여전히 기존 방식과 같기에, 처리 시간에 대한 한계점을 여전히 가지고 있다. 이를 극복하기 위해 [3번 단계]를 도입하였다.

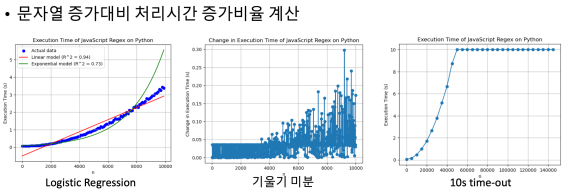

Figure 3. 탐지 가속 과정

[3번 단계]: 탐지 가속 / [2번 단계]에서 찾아낸 핫스팟을 기반으로 공격 문자열 케이스를 n, 2n, 4n, 8n 자 등으로 k배수만큼 문자열 길이를 펌핑하여 테스트한다. 이에 따른 처리시간이 k배수로 증가하는지 (선형적), 또는 그 이상(지수적)으로 증가하는지를 판단한다. 효율적인 분석을 위해 n은 충분히 작게 설정하여 처리시간을 최소화하면서도 유의미한 취약성 평가가 가능하다. 성능 지표의 변화 추이를 정량적으로 분석하기 위해 Figure 3과 같이 로지스틱 회귀(Logistic Regression)라는 머신러닝 알고리즘을 사용하였다. 로지스틱 회귀를 통해 입력 크기와 처리시간 사이의 관계를 모델링하고, 이를 바탕으로 성능이 선형적으로 증가하는지 혹은 그 이상으로 증가하는지를 판단한다.

결론적으로, 우리의 방법론은 기존의 정적 및 동적 탐지 방식의 한계점을 극복하여 더욱 빠르고 정확하게 ReDoS 취약성을 탐지할 수 있게 되었다.

3.3 OpenSource 대상 ReDos 취약점 탐지

깃헙(GitHub)을 대상으로 한 자동 크롤링 기능을 구현하여 실시간으로 최신의 코드와 프로젝트를 수집하였다. 이를 통해 ReDoS 취약점이 존재할 가능성이 있는 정규식을 광범위하게 식별할 수 있었다. 수집된 코드에서는 정규식을 자동으로 추출하는 알고리즘을 적용, 해당 정규식들을 분석하여 잠재적인 ReDoS 취약점을 자동으로 탐지하였다.

탐지된 취약점에 대하여 자동으로 Proof of Concept(POC) 코드를 생성하는 기능을 구현하였다. 이를 통해 실제 환경에서 해당 취약점이 exploit가능한지 빠르게 평가하고 검증할 수 있었다.

또한, 탐지된 취약점에 대한 정보와 생성된 POC코드를 바탕으로 취약점 제보 리포트를 자동으로 작성하는 기능도 도입하였다. 이를 통해 신속하게 취약점 제보를 진행하였으며, 광범위한 OpenSource 프로젝트에 빠른 대응과 보완이 가능하게 되었다.

3.4 새로운 탐지론의 우수성

새로운 ReDoS 탐지 방법론은 기존의 동적 및 정적 분석 방식의 한계를 극복하여 ReDoS 취약성의 정확하고 효율적인 탐지를 가능하게 한다.

초기 단계에서는 정규식의 복잡도, 백트래킹 유발 구문 및 문자열 길이 제한과 같은 다양한 조건들을 고려하여 정규식의 취약성 가능성을 빠르게 평가한다. 이후, 동적 핫스팟 분석을 통해 정규식 처리 시간이 극대화되는 경우를 식별하고, 특정 부분에서 노이즈가 발생했을 때의 처리 시간 변화를 관찰하여 ReDoS의 가능성을 평가한다.

핫스팟 기반의 성능 분석을 통해 공격 문자열 케이스의 처리 시간이 선형적으로 증가하는지, 또는 지수적으로 증가하는지를 판단하여 취약성의 심각도를 정확히 평가한다.

GitHub에서 실시간으로 코드를 크롤링하여 잠재적 ReDoS 취약점을 광범위하게 탐지하고, 탐지된 취약점에 대해 자동으로 증거 코드를 생성하며, 이를 바탕으로 취약점 제보 리포트를 자동 작성하여 신속한 대응이 가능하게 만든다.

이 모든 통합 접근 방식은 ReDoS 취약점 탐지의 속도와 정확도를 크게 향상시키며, 오픈 소스 프로젝트에 대한 빠른 대응과 보완을 가능하게 한다.

IV. 결론

본 논문에서는 정규 표현식의 ReDoS 취약점을 효과적으로 탐지하기 위한 새로운 Hybrid 방법론을 제시하였다. 이 방법론은 기존의 Static과 Dynamic 접근 방식을 결합하여 정규식의 ReDoS 취약성을 더 빠르고 정확하게 식별할 수 있는 체계를 개발하였다.

초기 단계의 정적 분석을 통해 잠재적 취약점을 신속히 평가하고, 위험성이 있는 정규식에 대해서는 동적 분석을 실시하여 처리 시간의 증가율과 패턴을 상세히 분석함으로써, 정규 표현식의 취약성을 보다 정밀하게 탐지할 수 있다. 본 방법론을 적용한 결과, node.js module package에서 246개의 취약한 정규식을 발견하고 CVE-2023-43646을 발급받는 성과를 이루었다는 점에서 그 우수성을 입증하였다.

또한, 본 연구는 실제 오픈 소스 프로젝트에 대한 신속한 취약점 대응과 보완이 가능하도록 하여, 개발 커뮤니티 내에서의 보안 인식 향상과 취약점 관리에 크게 기여할 것으로 기대된다. 향후 연구에서는 더 다양한 프로그래밍 언어와 환경에서의 적용 가능성을 탐구하고, 정규식에 전달되는 문자열 처리 과정을 고려하여 탐지 방법론을 더욱 고도화하여 정규 표현식을 사용하는 모든 영역에서의 보안을 강화할 수 있는 방안을 모색할 것이다.

[참고문헌]

[1] James C. Davis, Christy A. Coghlan, Francisco Servant and Dongyoon Lee. The Impact of Regular Expression Denial of Service (ReDoS) in Practice: An Empirical Study at the Ecosystem Scale. In Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018, pages 246-256, 2018.

[2] D. LLC, “Regexploit: DoS-able Regular Expressions,” 2021, https://github.com/doyensec/regexploit.

[3] Yuju Shen, Yanyan Jiang, Chang Xu, Ping Yu, Xiaoxing Ma, and Jian Lu. ReScue: Crafting Regular Expression DoS Attacks. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018, pages 225-235, 2018.

[4] Li, Yeting, Zixuan Chen, Jialun Cao, Zhiwu Xu, Qiancheng Peng, Haiming Chen, Liyuan Chen, and Shing-Chi Cheung. "{ReDoSHunter}: A Combined Static and Dynamic Approach for Regular Expression {DoS} Detection." In 30th USENIX Security Symposium (USENIX Security 21), pp. 3847-3864. 2021.

[5] Wang, Xinyi, Cen Zhang, Yeting Li, Zhiwu Xu, Shuailin Huang, Yi Liu, Yican Yao et al. "Effective ReDoS Detection by Principled Vulnerability Modeling and Exploit Generation." In 2023 IEEE Symposium on Security and Privacy (SP), pp. 2427-2443. IEEE Computer Society, 2023.