Link-E-Param : 웹 애플리케이션 보안 강화를 위한 URL 파라미터 암호화 기법

정회원 임 덕 병*, 종신회원 박 준 철**°

Link-E-Param : A URL Parameter Encryption Technique for Improving Web Application Security

Deok-Byung Lim* *Regular Member*, Jun-Cheol Park**° *Lifelong Member*

요 약

URL 파라미터는 민감한 정보나 제공된 링크에서 임의로 변경하면 보안 위험이 발생하는 것을 포함할 수 있다. 본 논문에서는 전체 URL 파라미터들의 이름과 값을 동시에 암호화하는 Link-E-Param(Link with Encrypted Parameters) 기법을 제안한다. 이 기법은 기존의 일부 URL 파라미터를 감추는 방식과 달리, 공격자에 의한 악의적 URL 파라미터 분석을 근원적으로 불가능하게 함으로써 URL 분석에 기반 하여 웹 사이트로부터 정보를 빼내려는 시도를 막는 역할을 한다. 제안 기법은 서블릿 필터 형태로 구현되기 때문에 서버에 jar파일 설치 후 설정 파일을 작성하기만 하면 기존 프로그램을 수정할 필요 없이 적용이 가능하다. Link-E-Param에서는 다양한 암호화 알고리즘이 지원되도록 구현하였다. 구현된 필터를 적용하여 실험한 결과, 암호화 및 복호화로 인해 사용자가 느끼는 응답 시간의 증가가 수용 가능한 수준이라 볼 수 있는 2~3% 에 불과함을 보인다.

Key Words : Web Application Security, Web Cracking, Encryption, URL Parameter, Servlet Filter

ABSTRACT

An URL parameter can hold some information that is confidential or vulnerable to illegitimate tampering. We propose Link-E-Param(Link with Encrypted Parameters) to protect the whole URL parameter names as well as their values. Unlike other techniques concealing only some of the URL parameters, it will successfully discourage attacks based on URL analysis to steal secret information on the Web sites. We implement Link-E-Param in the form of a servlet filter to be deployed on any Java Web server by simply copying a jar file and setting a few configuration values. Thus it can be used for any existing Web application without modifying the application. It also supports numerous encryption algorithms to choose from. Experiments show that our implementation induces only 2~3% increase in user response time due to encryption and decryption, which is deemed acceptable.

Ⅰ. 서 론

네트워크나 OS 레벨에서의 공격과 그에 대한 방어 방법은 많이 연구가 되어 있고, 그로 인해 대부분의 취약점은 이미 널리 알려져 방어가 되고 있다. 반면에 웹 애플리케이션 영역은 쏟아지듯 나타나는 신기술과 더불어 새로운 취약점이 속속 나타나고 있다[3]. 최근 발표된 보안 위험에 대한 내용을 보면 그림 1과 같이

※ 이 논문은 2011학년도 홍익대학교 학술연구진흥비에 의해 지원되었음.
* ㈜비에스지파트너스(dblim@bsgglobal.com), ** 홍익대학교 컴퓨터공학과(jcpark@hongik.ac.kr), (° : 교신저자)
논문번호 : KICS2011-06-271, 접수일자 : 2011년 6월 27일, 최종논문접수일자 : 2011년 9월 9일

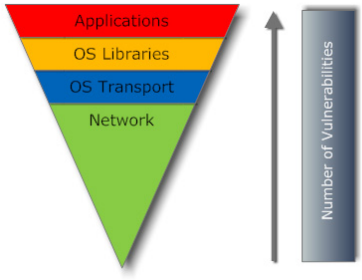


그림 1. 취약점 수 (출처 SANS 2009[7])

애플리케이션 레벨에서 취약점이 상대적으로 많이 나타남을 알 수 있다.

웹 애플리케이션 공격 단계를 그림 2처럼 공격대상 탐색, 정보수집, 시험, 공격계획 수립, 공격의 5단계로 볼 때, 정보수집 및 시험 단계에서 URL[5]을 분석하고 변조를 시도하는 행위가 발생한다.

웹 애플리케이션에서 URL 파라미터를 보호하는 것의 중요성을 예를 들어 살펴보자. 어떤 방송사에서 작성 중인 미발표 특종기사의 초안이 홈페이지를 통해 유출될 수 있음을 보이려 한다. 이 방송사 홈페이지에서 기사를 읽을 때 접속하는 링크의 URL이 www.news.net/view.jsp?article_id=1234 형태라고 하자. 공격자는 가장 최근 기사의 URL 파라미터가 article_id=1234 라는 정보를 수집하고 article_id라는 이름과 1234라는 값을 다른 내용으로 변경하여 웹 서버의 반응을 살펴본다. 실험을 통해서 article_id 파라미터에 다른 값을 넣으면 다른 기사를 볼 수 있다는 사실을 알게 되고, article_id에 최신기사인 1234 보다 더 큰 값을 넣어 보도록 공격계획을 수립한다. 공격의 결과 1240에서 특종기사의 초안을 발견한다.

예로든 뉴스 홈페이지의 웹 애플리케이션은 미공개인 초안 기사와 발행 후 공개된 기사를 구분하는 업무 프로세스가 없는 경우 이런 유형의 공격에 취약함을 드러낸다. 물론 근본적인 해결책은 "발행" 업무 프로세스를 만들고 파라미터 값이 적절한지 확인하는 과정을 거쳐, 발행 전 초안 기사에 대하여 접근 제어 기능을 가지도록 웹 애플리케이션을 수정하는 것이다. 그러나 애플리케이션을 모든 공격 가능성에 대비하여 처음부터 완벽하게 개발하는 것은 계속해서 새로운 공격 방식이 개발되고 있음을 고려할 때 시간과 비용의 측면에서 비효율적이다.

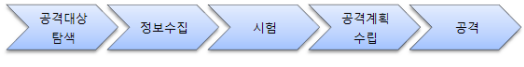


그림 2. 웹 애플리케이션 공격 단계[4]

제안하는 Link-E-Param(Link with Encrypted Parameters)은 정보수집 단계에서 URL 파라미터의 분석을 어렵게 하는 것을 목적으로 한다. 이 방식은 전체 파라미터들 모두에 대해 이름과 값을 암호화 한다. 따라서 파라미터의 이름이나 값 중 노출되는 것이 하나도 없기 때문에 URL 분석을 사실상 불가능하게 만들게 된다. 웹 공격의 시작은 URL 분석부터이므로, 제안 방식은 URL 분석을 통해 웹 사이트 공격하는 것을 효과적으로 방어한다.

인터넷 주소창에 URL을 직접 입력하는 경우도 있지만, 대부분의 경우 링크를 통해서 웹 애플리케이션에 접속한다. 예를 들어, Google의 검색 엔진 웹 애플리케이션에서 나오는 검색 결과나 뉴스 홈페이지에 있는 뉴스 기사 목록은 모두 링크이다. 웹 애플리케이션에서는 이런 링크를 이용하여 사용자와의 상호작용을 동적으로 가능하게 한다. 사용 중인 웹 애플리케이션에 URL 링크 파라미터 암호화 로직을 추가하려면, 파라미터를 처리하는 많은 부분에서 수정이 필요하게 된다. 이러한 불편함을 줄이기 위하여 Link-E-Param은 암호화 및 복호화 기능을 서블릿 필터(servlet filter) 형태로 제공한다. 웹 서버 종류에 따라 필터는 각각 별도의 구현이 필요한데, Link-E-Param은 웹 애플리케이션에서 가장 널리 사용되는 자바 언어 기반의 서블릿 필터로 구현한다. 따라서 서블릿 컨테이너가 있는 웹서버에 jar파일 설치 후 설정 파일을 작성하기만 하면 기존 프로그램에서 링크가 발생하는 부분을 일일이 수정할 필요 없이 쉽게 Link-E-Param을 적용할 수 있다. 표 1은 Link-E-Param의 적용 전과 후에 각각 링크가 어떻게 보이는지 나타낸다.

본 논문의 구성은 다음과 같다. 2장에서는 제안 방식의 이해를 위해 필요한 기본적 배경 지식을 소개하고, 3장에서는 URL 파라미터 암호화가 왜 필요한지 강조한 후 본 논문의 제안 방식과 직접 관련된 기존 연구 결과들을 비교 설명한다. 4장에서는 제안하는 Link-E-Param에 대해 적용 방법, 구조, 설치 및 암호화 키 생성을 중심으로 기술한다. 5장에서는 구현된 Link-E-Param이 응답 시간의 관점에서 실제 활용 가능한 수준임을 실험을 통해 검증하고, 마지막으로 6장에서는 결론을 맺는다.

표 1. Link-E-Param 적용 예

암호화 전	link.jsp?BankID=1234&AccountID=J5678
암호화 후	link.jsp?__E_PARAM=uyGs60sdn8myZ.fZVRjdKtqnlSsWm3PGmSle6Q__

Ⅱ. 배경지식

2.1 URL 파라미터

다음은 RFC 1738 문서[5]에서 정의한 HTTP URL의 형식이다.

```
http(s)://<host>:<port>/<path>?<searchpart>
```

?의 뒤에 위치하는 <searchpart>를 파라미터(또는 쿼리스트링)라고 부른다. 파라미터는 <이름>=<값> 형태로 쌍을 이루는데, 여러 파라미터가 존재하면 파라미터 사이에는 &가 들어간다. 서버 측 JSP 프로그램에서는 request.getParameter(<이름>); 명령문을 사용하여 파라미터의 값을 읽을 수 있다.

URL의 ? 뒤에 붙여서 파라미터를 전송하는 방식을 GET 방식이라 하고, HTTP 요청의 바디에 파라미터를 실어 보내는 방식을 POST 방식이라고 한다. 링크를 만들 때는 주로 GET 방식을 사용한다. POST 방식은 명령 작성이 더 복잡하고, HTML이 아닌 일반 텍스트로 전달할 수 없으며, 즐겨찾기로 등록할 수 없는 등의 한계로 인하여 링크에는 잘 사용되지 않는다.

2.2 서블릿 필터

서블릿 필터는 웹 서버로 많이 사용하는 자바로 작동하는 서블릿 엔진에 설치하여, 웹 애플리케이션 전반에 영향을 주는 모듈을 구현하기 위해 사용한다. 이러한 서블릿 필터로 request(요청)와 response(응답)를 조작할 수 있다. 카메라 렌즈 필터처럼 필터들을 조립식으로 적용하면 적용 순서에 따라 각 필터의 효과가 순차적으로 나타난다.

일반적으로 많이 사용하는 필터로는 문자셋 지정, 인증(로그인 검사), 성능 모니터링, 접근 로그 기록, XML 변환, 압축 등이 있고, 이런 기능들이 웹 애플리케이션 동작에 영향을 미치게 된다.

본 논문에서 제안하는 Link-E-Param은 암호화 기능을 서블릿 필터로 구현하였기 때문에, 필터 적용만 하면 웹 애플리케이션(서블릿, JSP 등)의 수정 없이 response의 URL 파라미터 암호화 및 request의 암호화된 URL 파라미터 복호화가 적용 된다.

2.3 Base64 인코딩

URL 파라미터를 암호화하면 임의의 이진 데이터가 되기 때문에 이를 화면에 표시할 수 있도록 Base64[8] 인코딩을 적용한다. 원래의 Base64의 64문자는 알파벳 52자, 숫자 10자, “+”, “/” 의 64자를 말하며, 추가로 단위를 맞추기 위한 패딩 문자 “=”를 사용한다. 여기에서 “+”, “/”, “=” 세 가지는 URL에서 다른 뜻으로 잘못 인식될 수도 있기 때문에, UrlBase64 인코딩은 이들을 각각 ″-″, ″_″, ″.″ 로 변경하여 URL 파라미터에 적용한다. 제안하는 Link-E-Param에서 사용하는 Base64는 실제로는 UrlBase64를 뜻한다.

2.4 자바 암호화

자바에 표준으로 포함되어 있는 암호화 프레임워크로 JCE (Java Cryptography Extension)가 있다. SunJCE는 암호화, 키 생성, 키 교환, MAC 알고리즘에 대한 구현을 제공한다. SunJCE에서 사용할 수 있는 암호화 알고리즘은 DES, Triple DES(DESede), Blowfish, AES, RC4 등이 있다.

자바 암호화 비교 연구[6]에 따르면 암호화 알고리즘의 선택 기준으로 알고리즘 자체의 보안성, 성능(속도)(그림 3 참고), 알고리즘 사용 시의 제약 사항, 기존 시스템과의 호환성 등을 고려해야 한다.

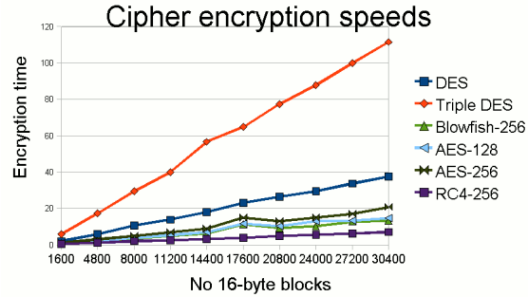


그림 3. 자바 암호화 속도 비교[6]

2.5 TLS/SSL

Link-E-Param과는 웹 애플리케이션에 적용할 수 있는 솔루션이라는 공통점을 가진 TLS/SSL[9]는 세션 단위로 서버와 클라이언트 간 키 생성 및 전송 내용의 암호화 서비스를 제공한다. 설명한 바와 같이 Link-E-Param은 웹 컨텐츠 자체를 암호화 하지는 않고, 단지 링크의 파라미터 부분을 암호화하여 분석 및 변조를 막는다. 즉, 링크의 파라미터 부분을 공개하고 싶지 않은 경우 사용한다. 이와 같이 TLS/SSL과 Link-E-Param은 서로 사용 목적이 다르며, 상호 대체할 수 있는 것이 아니다.

Ⅲ. URL 파라미터 암호화

3.1 URL 파라미터 암호화의 필요성

공격자는 웹 사이트 공격을 위한 정보수집 단계에서 파라미터들을 수집, 목록화[3] 후 이를 분석한다. 공격자는 분석 작업을 통해 파라미터들을 분류하거나, 특정 애플리케이션에 필요한 요소들이 각각 무엇이며 어떤 역할을 수행하는지를 파악한다.

/member/userinfo.jsp?**mode**=view&**id**=tommy

위의 URL요청에서 userinfo.jsp라는 애플리케이션이 mode와 id라는 파라미터를 필요로 함을 알 수 있다. 또한 mode 파라미터의 값이 현재 'view'라는 것을 통해 사용자 정보를 "보여" 주는 것이고, id 파라미터의 값이 'tommy'라는 것을 통해 tommy라는 사용자의 정보를 보여 주는 것이라고 유추할 수 있다. 만약 공격자가 id의 값인 'tommy' 대신 다른 사용자 계정의 이름을 입력한다면 애플리케이션이 어떻게 동작하게 될까? 이러한 접근을 통해 URL 파라미터를 분석하고 목록화하는 것은 추후 취약성의 분석 및 공격 시도에 있어서 반드시 필요할 뿐만 아니라 웹 크래킹을 위한 중요한 정보가 될 수 있다. 전통적인 시스템 크래킹 기법 중 하나인 오버플로 공격과 웹 크래킹의 여러 공격 기법은 입력 값의 부적절한 조작이라는 점에서 일맥상통한다.

파라미터 분석과 목록화를 막기 위해서는 <이름>=<값> 파라미터 형태에서 값 뿐 아니라 이름 또한 노출 되지 않도록 파라미터 전체를 암호화 하는 것이 더 효과적이다. 파라미터 전체를 암호화 하면 이름과 값을 모두 숨길 수 있을 뿐더러 파라미터의 개수조차 감출 수 있다. URL 파라미터를 암호화 하는 일은 개념적으로 간단하다. 응답문서 내에서 링크를 찾아서 URL 중 ?뒤의 파라미터 부분을 암호화 하고, 반대로 요청 받은 URL에서 암호화된 파라미터는 복호화 하면 된다. 문제는 이러한 암호화 및 복호화 과정을 파라미터 몇 개 정도만을 가진 웹 애플리케이션에 적용하는 것은 가능하지만, 중대형 이상의 사이트 전체 웹 애플리케이션에 적용하는 것은 결코 간단하지 않다는 점이다.

3.2 관련 연구

3.2.1 QueryCrypt

QueryCrypt[2]는 Aveda Technology사에서 판매하였던 소프트웨어로서 2006년 오픈소스로 전환되었다. QueryCrypt는 링크의 URL 파라미터를 암호화 하는 기능을 라이브러리로 묶어서 제공하기 때문에 라이브러리가 없이 암호화 하는 것보다 코딩 분량을 줄일 수 있다. 하지만 실제로 애플리케이션에 적용하려면 여전히 상당한 부분의 코드 수정이 필요하다(표 2).

표 2. QueryCrypt의 암호화를 위한 코드 수정

수정전	click me 1
수정후	<a href="testMe.jsp? <%=(QueryCrypt.getInstance().encrypt(request," p=abcabcabcabc&q=def "))%> ">click me 1

3.2.2 Spiegelberg 의 방식

2008년에 java.net에 발표된 Eric Spiegelberg의 글[1]에는 JSTL 태그와 서블릿 필터라는 새로운 기술을 활용하여, QueryCrypt처럼 라이브러리를 이용하여 코드를 작성하는 방식보다 조금 더 편리하게 기존 애플리케이션에 URL 파라미터 암호화를 적용하는 방법이 소개되었다(표 3). 이 방법의 단점은 JSTL 이라는 JSP 중에서도 일부에만 사용하는 태그방식 —현재 그리 널리 쓰이지는 않음 —을 사용하기 때문에 JSTL을 사용하지 않는 일반 JSP, 서블릿, 또는 순수 자바에서는 전혀 사용할 수가 없다는 것이다.

표 3. Spiegelberg의 방식의 암호화를 위한 코드 수정

수정전	<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> <a href="<c:url value="testMe.jsp"> <c:param name="p" value="abcabcabcabc" /> <c:param name="q" value="def" /> </c:url>">click me 1
수정후	<%@ taglib prefix="sUrl" uri="http://www.spiegs.com/jsp/jstl/secureUrl" %> <a href="<sUrl:url value="testMe.jsp" encryptionEnabled="true"> <sUrl:param name="p" value="abcabcabcabc" /> <sUrl:param name="q" value="def" /> </sUrl:url>">click me 1

3.2.3 기존 관련 연구들과의 비교(표 4)

표 4. QueryCrypt, Spiegelberg의 방식, Link-E-Param 비교

	QueryCrypt	Spiegelberg의 방식	Link-E-Param
암호화	- 소스 코드 코딩 단점: 코드 수정이 많음	- JSTL 태그 장점: 기존 JSTL을 쉽게 대체 가능 단점: JSP 중에서도 JSTL을 사용하는 경우만 적용가능	- 필터 장점: 소스 코드 수정 필요 없음 단점: 약간의 필터 설정
복호화	- 소스 코드 코딩 단점: 소스 코드 수정이 많음	- 필터 (attribute 사용) 단점: 약간의 소스 코드 수정 필요 (getParameter를 getAttribute로 변경)	- 필터 (parameter 대체) 장점: 소스 코드 수정 필요 없음 단점: 약간의 필터 설정
암호화 방식	DESede + MD5	Bcodec (Base64 변형)	Base64, DES, DESede, Blowfish, AES, RC4 중 설정으로 자유롭게 선택 가능
암호화 키 관리	- 사용자 세션단위로 키 관리 - 매번 랜덤 키를 생성하므로 링크를 이메일로 전송하거나 즐겨찾기로 방문하면 복호화 불가능	- 보안성이 없는 난독화로 구현 (키 사용하지 않음) - 암호화 방식 추가 구현 시 키 관리 방법도 함께 구현해야 함	- 하나의 마스터 키만 관리함 - 각 서버의 웹 애플리케이션 단위로 고유한 키가 자동 생성됨
링크 예	<a href="testMe.jsp?<%=(QueryCrypt.getInstance().encrypt(request,"p=abcabcabcabc&q=def"))%>">click me 1	<a href="<sUrl:url value="testMe.jsp" encryptionEnabled="true"><sUrl:param name="p" value="abcabcabcabc" /><sUrl:param name="q" value="def" /></sUrl:url>">click me 1	click me 1

Ⅳ. Link-E-Param

4.1 구조 및 암·복호화 흐름도 (그림 4)

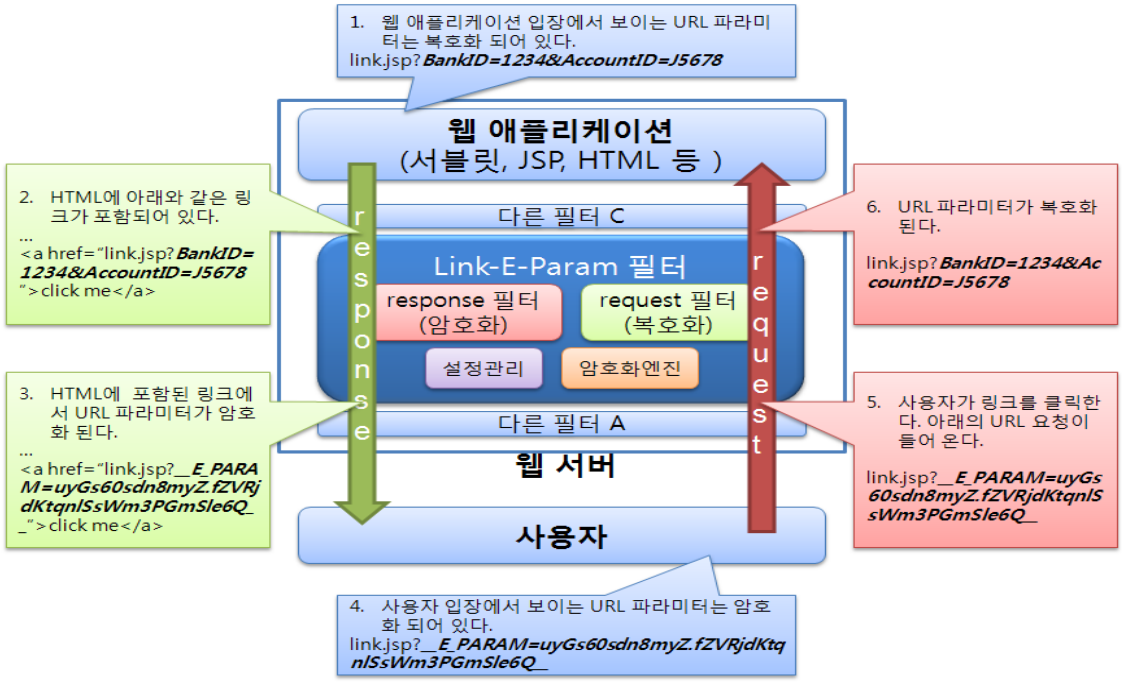


그림 4. Link-E-Param 구조 및 암·복호화 흐름도

4.2 특징 및 설계 고려사항

4.2.1 필터로 설치하여 빠른 적용

Link-E-Param은 관련 연구와 달리 필터로 암·복호화 부분을 구현하기 때문에 소스 코드 수정 없이 필터를 설치하고 약간의 설정으로 바로 작동하여 기존 웹 애플리케이션의 보안을 강화할 수 있다.

4.2.2 다양한 암호화 방식 지원

암호화 방식을 사용 환경 및 요구하는 보안 수준에 따라 다양하게 선택할 수 있도록 하였다. 자바의 SunJCE에 있는 모든 암호화 방식을 지원하는데 기본 설정 값은 AES 128 비트이다. 일부 국가의 암호화 기술 수출 제한 정책 때문에 자바에서는 암호화 키의 길이를 128 비트로 제한한다. 암호화 키로 256 비트 키를 사용하려면 JCE policy 파일을 별도로 다운로드 받아서 설치하여야 한다.

4.2.3 간단한 키 관리

Link-E-Param은 서버에 하나의 마스터 키만 설정하는데, 이 마스터 키를 실제 암호화에 직접 사용하지는 않는다. 실제 암호화를 할 때는 호스트 이름, 경로, 웹 애플리케이션 이름에 따라 달라지는 고유한 값이 암호화 키로 사용된다. 마스터 키는 HMAC의 키 역할을 함으로써, 마스터 키를 모르는 공격자는 나머지 정보 모두를 안다고 할지라도 이러한 암호화 키를 만들어낼 수 없다. 결과적으로 하나의 마스터 키만을 저장하고 있으면서도, 실제 암호화에 사용되는 키는 각각의 웹 애플리케이션별로 다르게 생성된다. 해시의 특성상 웹 애플리케이션 간에 키가 중복 사용될 가능성은 거의 없다. 또한 다른 경로를 통해 특정 웹 애플리케이션에 대한 암호화 키가 유출 되더라도 해시의 역계산 불가능성에 의해 마스터 키는 알기 어려우며, 따라서 다른 웹 애플리케이션의 암호화 키 역시 알 수 없다.

4.2.4 우수한 처리 속도

필터는 서버에 설치되어 서버 내 전체 웹 애플리케이션에 영향을 미치기 때문에 설계 단계에서부터 성능에 대한 고려를 하는 것이 필요하다. 암·복호화 알고리즘은 기존의 라이브러리를 그대로 사용하기에 이 부분에 대한 성능 개선은 본 논문에서 고려하지 않는다. 암·복호화 다음으로 성능에 많은 영향을 미치는 부분은 response 필터에서 응답 문서를 훑어서 링크를 찾는 과정이다. 웹 애플리케이션에서 나오는 응답은 HTML 이외에도 바이너리 첨부파일, 그림 파일, 자바스크립트 파일, 일반 데이터 등이 가능하다. 링크가 없는 비 HTML 에 대해서는 암호화 과정이 필요 없기 때문에, HTTP 헤더의 contentType을 확인하고 문서가 태그 시작인 '<'로 시작하는지 확인하는 등 여러 가지 검사를 통해 링크가 존재할 가능성이 있는 HTML에만 링크를 찾도록 하였다. 링크는 정규표현식을 사용하면 쉽게 찾을 수 있지만, 문자열 처리보다 성능이 우수한 바이트 단위의 비교 방법을 사용하여 찾도록 하였다.

4.3 알고리즘

4.3.1 설정 값 (표 5)

표 5. Link-E-Param에 필요한 설정

이름	설명	기본값
eParamName	암호화된 파라미터 이름	__E_PARAM
cryptType	암호화 방식	AES-128
masterKey	암호화 마스터 키	
coveragePath	링크의 암(복)호화 적용 영역 url-pattern	/*
extensions	암호화 필터 적용 확장자	htm,html,jsp,do

4.3.2 암호화 키의 생성

$$KEY = HMAC(masterKey,\ URL\ path\ part)$$

masterKey는 설정 값으로 암호화 키를 만들기 위한 마스터 키이다. URL path part는 서로 다른 경로에 대해 고유한 값의 키를 만드는 역할을 한다.

4.3.3 암호화된 파라미터 값의 구성

$$\_\_E\_PARAM = Base64Encode(checksum(1Byte) \parallel encryptedData(NBytes))$$

checksum은 평문의 각 바이트 값을 모두 더한 후 마지막 1 바이트를 취한 값이다. encryptedData = encrypt(KEY, plainParam) 와 같이 계산한다. 마지막으로 Base64Encode는 암호화된 데이터를 URL 파라미터로 전송하기 적합한 문자로 변경한다.

4.3.4 request 필터의 복호화

1. __E_PARAM (설정 값: 암호화된 파라미터 이름) 파라미터가 있는지 확인한다. 없으면 종료한다.

2. coveragePath (설정 값: 필터 적용 영역)에 해당하는 요청 URL인지 확인한다. 아니면 종료한다.
3. __E_PARAM 파라미터 값에서 Base64Decode로 checksum과 encryptedData를 구한다.
4. 설정된 암호화 방식으로 복호화 한다. plainParam = decrypt(KEY, encryptedData)
5. plainParam(복호화된 파라미터)의 checksum을 점검한다. 일치 하지 않으면 에러 처리하고 종료한다.
6. plainParam을 &로 잘라서 name=value 형태로 나누어 parameterMap에 넣는다.
7. 이후 애플리케이션에서 파라미터 값 요청 시 parameterMap에서 찾아서 제공한다.

4.3.5 response 필터의 암호화

1. 요청받은 URL이 extensions (설정 값: 적용 확장자)에 해당하는 문서인지 확인한다. 해당하지 않으면 종료한다.
2. response의 HTTP 헤더의 contentType (문서종류)이 "text/html" 으로 시작하는지 확인한다. 아니면 종료한다.
3. 첫 문자가 HTML 태그 시작인 '<'로 시작하는지 확인한다. 아니면 종료한다.
4. 링크를 찾는다. (href="LINK_URL" 또는 src="LINK_URL" 패턴을 찾는다)
5. 링크의 ?뒤 파라미터가 있는지 확인한다. 없으면 다음 링크를 찾는다.
6. 링크의 URL host부분이 일치하는지 확인한다. 일치하지 않으면 다음 링크를 찾는다.
7. 링크의 URL path부분이 coveragePath (설정 값: 필터 적용 영역)에 해당하는지 확인한다. 해당하지 않으면 다음 링크를 찾는다.
8. 링크의 URL 파라미터에 __E_PARAM 이 없는 것을 확인한다. 있다면 이미 처리된 것이므로 다음 링크를 찾는다.
9. 설정된 암호화 방식으로 암호화 한다. encryptedData = encrypt(KEY, plainParam)
10. checksum을 계산하여 앞에 붙이고 Base64Encode를 적용한다. encryptedParam = Base64Encode(checksum || encryptedData)
11. 링크의 기존 파라미터 부분을 지우고 __E_PARAM=encryptedParam 을 넣는다.
12. 남은 응답 내용에 대해 4번부터 다시 반복하여 다음 링크를 찾는다.

4.4 설치

4.4.1 jar 파일 설치

Link-E-Param의 설치는 일반적인 서블릿 필터 설치법을 따라서 진행된다. 제공된 Link-E-Param.jar 파일을 서버의 lib 폴더에 설치하고, web.xml 파일에 필터 정보를 기입한다. 이 때 init-param 태그로 설정 값을 입력할 수 있다.

4.4.2 설정 파일 예 (web.xml 파일)

```
<filter>
  <filter-name>Link-E-Param Filter</filter-name>
  <filter-class>boy0.filters.LinkEParamFilter</filter-class>
  <init-param>
    <param-name>cryptType</param-name>
    <param-value>4</param-value>
  </init-param>
  <init-param>
    <param-name>masterKey</param-name>
    <param-value>input here!! your key</param-value>
  </init-param>
  <init-param>
    <param-name>coveragePath</param-name>
    <param-value>/link_test/*</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>Link-E-Param Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

V. 성능 실험 및 분석

기존 관련 연구들과 제안 방식의 암호화 기법 사이의 상대적 성능을 실험을 통해 비교하였다. 상호 비교 시 큰 응답 시간 값을 얻기 위해 서버로는 구형 Pentium IV 2.4 GHz 컴퓨터를 사용하였다.

5.1 시나리오 1: 암호화 성능 측정 비교

이 실험의 목표는 암호화를 하지 않은 것과 기존 관련 연구의 방식 및 Link-E-Param의 각종 URL 파라미터 암호화 적용에 따른 응답 시간의 상호 차이를 비교하는 것이다. 즉, 링크의 암호화 과정이 없는 기본 방식과 비교하여 암호화 및 인코딩의 부하 정도를 판단하고자 한다. 링크를 찾고 암호화하는 성능을 위주로 비교하기 위해 링크만 100개가 있는 파일을 사용하는데, 이 파일에는 업무 처리, DB 접속, css 파일, js 파일, 그림 파일 등이 전혀 없다고 가정한다. 모두 10 명의 사용자가 동일 과정을 600회 반복(총 6,000회)한 결과의 응답 시간을 측정하여 그 평균값을 표시

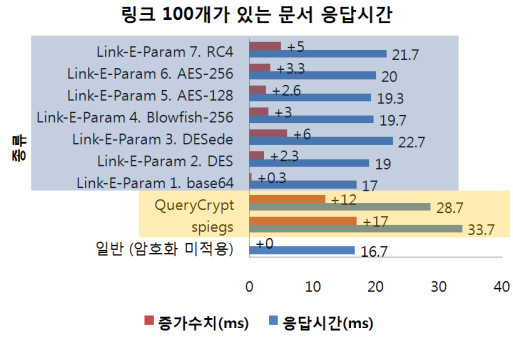


그림 5. 시나리오1의 실험 결과

하였다. 결과는 그림 5와 같다.

실험 결과를 보면 암호화를 하지 않은 기본 방식에 비해 Link-E-Param은 응답 시간의 증가 수치가 크지 않음을 알 수 있다. Base64의 경우 실제로 암호화가 아닌 가벼운 연산이기 때문에 매우 적은 시간 증가를 보인다. 다른 암호화 방식을 적용한 Link-E-Param 실험은 자바 암호화 비교 연구[6]에서 발표한 성능 비교 수치(그림 3)와 매우 유사한 결과를 보인다. 결국 Link-E-Param의 성능은 암호화 방식의 성능에 크게 영향을 받으며, 구현이 별도의 오버헤드를 초래하지 않는 수준으로 적절히 되었다고 해석할 수 있다. 암호화 성능은 하드웨어 발전 속도를 고려할 때 점점 더 사용자에게 부담이 되지 않는 수준으로 개선될 것으로 본다.

실험에 사용한 파라미터 길이는 20자~200자 사이에 분포되도록 하였다. 기존 관련 연구와 비교를 하여도 Link-E-Param은 성능 상의 강점을 보인다. Spiegelberg의 방식은 Bcodec 이라는 Base64 변형의 가벼운 인코딩을 사용하므로, 실제 암호화 과정은 없다는 점을 고려한다면 성능이 우수하다고 보기 어렵다. 성능이 좋지 못한 이유는 JSTL 태그 방식의 오버헤드로 추정된다. Spiegelberg는 아이디어 발표 정도의 글을 쓴 것이기에 성능에 대한 고려를 하지 않은 것으로 판단된다. QueryCrypt는 DESede를 사용하기 때문에 Link-E-Param의 DESede 방식과의 직접 비교가 가능한데, 실험 결과 Link-E-Param에 비해 시간이 더 많이 걸림을 알 수 있다. 이 시간 차이는 구현 시 효율성의 차이에 기인한 것으로 판단된다. URL 파라미터와 같이 비교적 적은 데이터 양에 대해 암호화를 적용할 때는, 암호화를 실행하는 시간보다 암호화 객체의 초기화 및 키 생성에 상대적으로 더 많은 시간이 소요 된다. QueryCrypt는 암호화 할 때마다 암호화 객체 초기화 및 키 생성이 반복적으로 수행되기 때문에 효율적이지 못하다.

Link-E-Param의 경우 예상했던 대로 암호화 방식에 따라 측정된 응답 시간이 차이가 남을 볼 수 있다. 일반적으로 알려진 바와 다르게 암호화 속도가 가장 빠르다는 스트림 암호화 알고리즘 RC4가 몇몇 다른 암호화 알고리즘에 비해 응답 시간이 큰 것으로 나타났는데, 그 이유는 암호화할 데이터 크기가 작아서 스트림 암호화의 성능 상 장점이 충분히 발휘되지 못한 때문으로 추정된다. 암호화 방식 간의 비교에서는 AES가 뛰어난 성능을 보였다. AES 알고리즘은 보안성에 대해서도 검증이 되었기에, Link-E-Parm에서는 128 비트 키를 사용하는 AES를 기본 암호화 방식으로 결정하였다. 필요 시 AES 암호화 키의 길이를 256 비트로 늘려 보안성을 높이는 것도 가능하다.

5.2 시나리오 2: 실사용 환경의 성능 영향 측정

이 실험의 목적은 실제 대표적인 웹 사이트 운영 환경과 유사한 상황에서 Link-E-Param을 평가하여 실제 상황에 Link-E-Param이 충분히 효율적임을 보이는 것이다. 실험을 위해 JSP 게시판 애플리케이션을 설치하고 Link-E-Param 필터를 적용하는데, 게시판 애플리케이션에는 어떠한 소스 코드 수정도 가하지 않는다. 따라서 소스 코드를 수정해야만 하는 Spiegelberg의 방식과 QueryCrypt는 실험할 수 없었다.

게시판 애플리케이션의 게시 글 목록이 나온 페이지에는 업무 처리, DB 접속, css 파일, js 파일, 그림 파일 등이 포함되어 있다. 모두 10 명의 사용자가 동일 과정을 10회 반복(총 100회)한 결과의 응답 시간을 측정하여 그 평균값을 표시하였다. 결과는 그림 6과 같다.

이 실험은 암호화 방식들 간의 성능 비교보다 실제 상황과 유사한 웹 페이지에 대해 Link-E-Param의 암호화로 인한 응답 시간의 증가가 얼마나 되는지를 추정하게 해 준다. 실험 결과 기본 설정인 AES 128 비트 암호화의 경우 Link-E-Param의 암호화로 인한 응

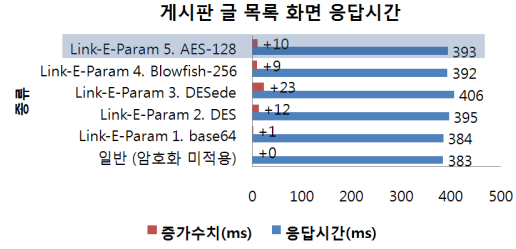


그림 6. 시나리오2의 실험 결과

답 시간의 증가가 2~3% 정도에 불과함이 입증되었다. 가장 시간이 많이 걸린 DESede의 경우에도 단 6% 수준의 응답 시간만을 보였다. 결과적으로, 실제 웹 페이지에 Link-E-Param을 적용하더라도 사용자의 입장에서는 암호화를 적용하지 않은 것에 비해서 응답 시간에 유의미한 차이를 느끼지 못할 것이라 판단한다.

Ⅵ. 결 론

URL 파라미터는 웹 애플리케이션 작동을 위해 필요하다. 일반적으로 링크로 함께 제공된 파라미터가 변경되지 않는다고 가정하고 웹 애플리케이션을 작성하지만, 실제로 URL 파라미터는 쉽게 위변조가 가능하다. 본 논문에서 제안한 Link-E-Param은 링크로 함께 제공된 파라미터가 변경되지 않는다는 가정을 현실로 만든다. 제안 방식은 공격자의 URL 분석 단계를 어렵게 만들고, 파라미터 조작을 못하게 함으로써 웹 애플리케이션에 대한 중요한 공격 루트를 차단하는 효과를 낸다. Link-E-Param은 기존 프로그램의 수정 대신 서블릿 필터로 설치하기 때문에 빠른 적용이 가능하고, 다양한 암호화 방식을 지원하며, 키 관리가 간단하면서도, 처리 속도가 빠르다는 장점이 있다. 요약하면, 제안하는 Link-E-Param이 모든 웹 기반 공격을 방어하는 것은 아니지만, 손쉽고 효율적인 처리를 통해 웹 애플리케이션 보안 수준을 크게 높인다고 볼 수 있다.

제안하는 Link-E-Param은 암·복호화 기능을 서블릿 필터로 구현하였기 때문에 현재로서는 자바 계열의 서블릿 엔진이 있는 웹 서버에서만 적용할 수 있다. 하지만, 동일한 개념을 아파치 등의 웹 서버 필터로 구현한다면 jsp, asp, php 등 다른 환경의 웹 애플리케이션에도 적용 할 수 있을 것이다.

현재 구현된 Link-E-Param은 GET 방식 링크의 URL 파라미터 암호화로 적용 범위가 한정되어 있다. 향후, 자바스크립트와 폼 전송(POST 방식)에도 적용할 수 있도록 제안 방식을 확장하는 연구가 필요하리라 판단된다.

참 고 문 헌

〔1〕 E. Spiegelberg, “Securing Your Web Application Requests”, http://today.java.net/article/2008/05/14/securing-your-web-application-requests, 2008.

〔2〕 V. Guhesan, “QueryCrypt [Encrypt Your Query Parameters]”, http://www.avedatech.com/Products/QueryCrypt/index.jsp, 2006.

〔3〕 황순일, 김광진, “웹 해킹 패턴과 대응”, 사이텍미디어, 2005.

〔4〕 이용호, 박명수, 윤준, 윤정원, “안전한 웹 서비스를 위한 웹 어플리케이션 공격 유형 및 대응 방안 분석”, 정보보호학회지, 제14권, 제4호, pp.1-9, 2004년 8월.

〔5〕 T. Berners-Lee, L. Masinter, M. McCahill, “Uniform Resource Locators (URL)”, *IETF* RFC 1738, http://www.ietf.org/rfc/rfc1738.txt, 1994.

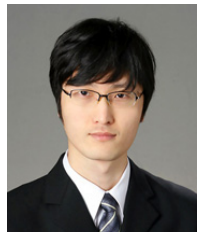
〔6〕 N. Coffey, “Comparison of encryption ciphers in Java”, http://www.javamex.com/tutorials/cryptography/ciphers.shtml

〔7〕 SANS Top Cyber Security Risks 2009, http://www.sans.org/

〔8〕 Base64, http://en.wikipedia.org/wiki/Base64

〔9〕 Transport Layer Security (TLS), http://en.wikipedia.org/wiki/Transport_Layer_Security

임 덕 병 (Deok-Byung Lim) 정회원



2004년 2월 홍익대학교 컴퓨터공학과 학사

2011년 8월 홍익대학교 컴퓨터공학과 석사

현재 ㈜비에스지파트너스 컨설턴트

<관심분야> 웹 애플리케이션

박 준 철 (Jun-Cheol Park) 종신회원

1986년 서울대 계산통계학과

1988년 KAIST 전산학과 석사

1998년 U. of Maryland, College Park, 전산학 박사

현재 홍익대학교 컴퓨터공학과 부교수

<관심분야> 네트워크, 시스템 보안