소프트웨어 취약점의 종류와 탐색 방법

김남규* · 김현호** · 이훈재**

*동서대학교

Browser fuzzing and analysis using known vulnerability

Nam-gue Kim* · Hyun Ho Kim** · Hoon-Jae Lee**

*Dept. of Information and Communication Engineering, Dongseo University

**Dept. of Ubiquitous IT Graduate School of Dongseo University

**Dept. of Computer Engineering, Dongseo University

e-mail : knq512412@gmail.com, feei_@naver.com, hjlee@dongseo.ac.kr

요 약

인터넷 기술이 보편화되면서 웹브라우저에서 뉴스, 쇼핑, 검색 등 다양한 활동이 이뤄지고 있다. 그 규모가 방대해지고 정보보안에 사고의 규모가 증가하고 이에 대한 피해가 증가함에 따라 인터넷 사용에 대한 안전성이 강조되고 있다. 일반적으로 사용하는 소프트웨어인 IE 브라우저는 ASLR, Isolated Heap 등 다양한 보호 기법 등 수많은 취약점을 지속적으로 패치 하고 있으나 취약점은 끊임없이 나오고 있다. 따라서 보안 사고를 예방하려면 해당 취약점이 악용되기 이전에 찾아내서 제거해야한다. 이에 본 논문은 취약점 발견에 사용되는 기법인 퍼징에 대해 소개하며, 이에 관련된 자동화 기술에 대해 기술한다.

ABSTRACT

Internet technology is universal, news from the Web browser, shopping, search, etc., various activities have been carried out. Its size becomes large, increasing the scale of information security incidents, as damage to this increases the safety for the use of the Internet is emphasized. IE browser is ASLR, such as Isolated Heap, but has been continually patch a number of vulnerabilities, such as various protection measures, this vulnerability, have come up constantly. And, therefore, in order to prevent security incidents, it is necessary to be removed to find before that is used to exploit this vulnerability. Therefore, in this paper, we introduce the purge is a technique that is used in the discovery of the vulnerability, we describe the automation technology related thereto. And utilizing the known vulnerabilities, and try to show any of the typical procedures for the analysis of the vulnerability.

키워드

IE, Internet Explorer, Browser, Fuzzing

Ⅰ. 서 론

인터넷 기술이 보편화되면서 웹브라우저에서 뉴스, 쇼핑, 검색 등 다양한 활동이 이뤄지고 있다. IE, firefox, chrome, opera, thor 등 다양한 브라우저들을 사용자들에게 제시되고 있다. 수많은 브라우저 중 하나인 IE는 높은 점유율을 차지하지만 매해 수많은 취약점이 발견되고 있다. 만일, 다수가 사용하는 브라우저에 취약점이 발견된다면 사용자는 악의적인 해커로부터 개인정보를 침해당할 수 있는 위험이 생긴다. 본 논문에서는 브라우저에서 발생할 수 있는 취약점을 소개하고, 그러한 취약점을 효율적으로 찾을 수 있는 자동화 기법을 기술한다.

Ⅱ. 취약점 종류

해당 장에서는 취약점에 대해 정의하고 소프트웨어에서 발생 빈도가 높은 취약점 종류에 대해서 기술한다.

2.1 CVE(Common Vulnerabilities and Exposures)

취약점 정보가 많아지고, 여러 취약점 분석 기관 및 벤더들에 의해 중복된 취약점 정보들이 생성되면서 취약점 정보의 중복으로 인한 혼선을 최소화하기 위해 CVE[1]를 구축하였다. MITRE (미연방정부의 정보보안연구 기관)에서 CVE를 개

발하여 관리하고 있으며, 아래와 같은 정보를 담아 취약점 식별정보를 고유하게 제공하고 있다.

- CVE 식별자 번호 (즉, “CVE-2012-4792”)
- 보안 취약점에 대한 간략한 설명
- 관련된 모든 레퍼런스(즉, 취약성 보고서, 대응 방안)

2.2 Stack BOF (Buffer Over Flow)

버퍼 오버플로우는 소프트웨어 개발자가 버퍼를 생성했는데, 해당 버퍼의 용량을 초과하여 페이로드가 삽입되었을 경우 초과한 용량만큼 의도치 않은 메모리 참조를 가지게 되어 특정 행위를 유발한다.

오버플로우를 유발하는 함수들은 다음과 같다.

```
gets(char *s)
scanf(const char *format, ...)
strcpy(char *dest, const char *src)
strcat(char *dest, const char *src)
```

그림 1. 버퍼 오버플로우에 취약한 함수들

[그림 1]의 버퍼 오버플로우에 취약한 함수 중 strcpy()를 사용하여 버퍼 오버플로우에 취약한 서버를 만들고 클라이언트 측에서 버퍼의 크기를 넘는 페이로드를 보낸다.

```
void pr(char *str)
{
    char buf[500] = " ";
    strcpy(buf, str);
}
/*--중략--*/
while (bytesRecv == SOCKET_ERROR)
{
    // 클라이언트로 부터 최대 5000바이트 크기의 데이터를 받음
    bytesRecv = recv(clientSocket, Message, 5000, 0);
    if (bytesRecv == 0 || bytesRecv == WSAECONNRESET)
    {
        printf("\nConnection Closed.\n");
        break;
    }
}
// 받은 데이터를 함수 pr로 넘겨줌
pr(Message);
```

그림 2. 버퍼 오버플로우 예시

[그림 2]에서 최대 5000바이트의 메시지를 클라이언트로부터 받는데 서버는 500바이트의 크기밖에 수신하지 못한다. [그림 3]의 스택 구조를 보면 ESP가 buffer의 시작 주소에 위치하며 이후 EBP-EIP 순으로 레지스터가 저장되어 있다. 만일 오버플로우가 된다면 EBP-EIP-ptr to buffer 순으로 넘친 페이로드의 값이 씌워질 것이다.

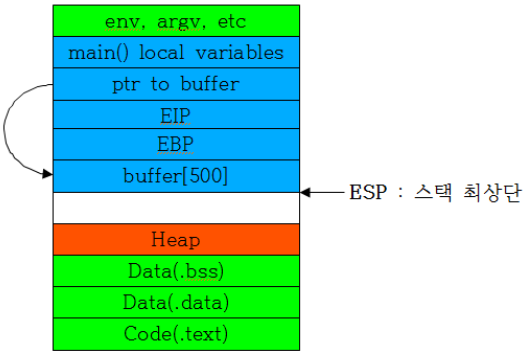

그림 3. 스택 구조

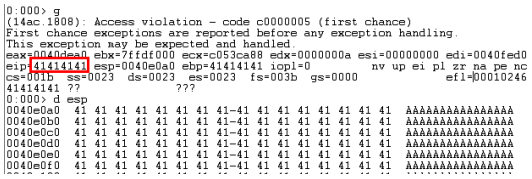

그림 4. 결과

[그림 4]를 보면 버퍼 오버플로우로 인한 태로, 수신할 수 있는 버퍼의 최대값(500byte)를 넘어서 eip와 ebp 모두 공격자가 지정한 임의의 (‘A’)로 채워졌다. 즉, EIP를 클라이언트 측에서 원하는 값으로 변경할 수 있다는 뜻이다. EIP를 컨트롤할 수 있다면 특정 명령어(jmp esp, pop-pop-ret)를 활용하여 특정 동작을 유도할 수 있다.

2.3 UAF (Use After Free)

브라우저에서 많이 발견되는 Use-After-Free 취약점은 객체가 해제된 후에도 여전히 존재하는 객체를 참조할 때 발생하는 취약점이다. 해제된 메모리 위치에 공격자가 다시 원하는 값을 쓸 수 있고, 그 값을 참조할 경우, 공격자가 원하는 방향으로 프로그램의 흐름을 제어할 수 있다.

[그림 5]의 코드를 보면 구조체를 선언한 후 맨 위부터 3개의 포인터 변수에 default heap 할당을 한다. 이후 차례대로 문자열을 입력받아 동적 할당한 포인터 변수가 가리키는 메모리 공간에 문자열을 복사를 하고 a1.BBB를 할당 해제한다. 새롭게 동적할당한 a1.DDD가 가리키는 메모리 공간에 문자열을 복사하고 이 메모리 공간이 할당되어진 a1.BBB와 어떻게 연관되어있는지 확인한다.

```
typedef struct _Alpha
{
    char * AAA;
    char * BBB;
    char * CCC;
    char * DDD;
}Alpha;

void main()
{
    Alpha a1;
    char temp[1024];
    a1.AAA = (char*)malloc(strlen(temp) +1);
    a1.BBB = (char*)malloc(strlen(temp) +1);
    a1.CCC = (char*)malloc(strlen(temp) +1);

    gets_s(temp);
    strcpy(a1.AAA, temp);
    gets_s(temp);
    strcpy(a1.BBB, temp);
    gets_s(temp);
    strcpy(a1.CCC, temp);

    free(a1.BBB); // 해제

    a1.DDD = (char*)malloc(strlen(temp) +1); // 동적 할당
    gets_s(temp);
    strcpy(a1.DDD, temp);

    scanf("%s", a1.DDD);
    printf("%s", a1.BBB); // 해제됐음에도 불구하고 a1.DDD의 값이 나온다.(재사용)
}
```

그림 5. UAF 요약 코드

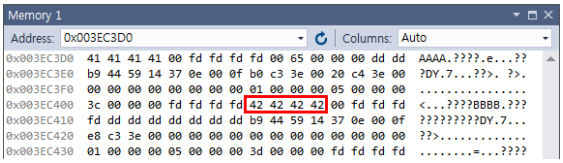

그림 6. 해제할 메모리

[그림 6]에서 임의로 받은 문자열 "BBBB"가 a1.BBB가 default heap에 할당받는 주소로 저장되어 있다.

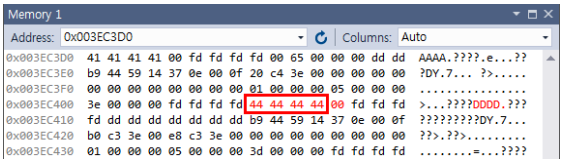

그림 7. (default) heap 재사용

[그림 7]은 free()로 a1.BBB를 해제한 후 a1.DDD를 default heap에 할당하고 "DDDD"를 저장하였더니 a1.AAA와 a1.CCC 사이에 저장된다. 바꿔 말하면 a1.BBB가 있었던 자리에 저장된다.

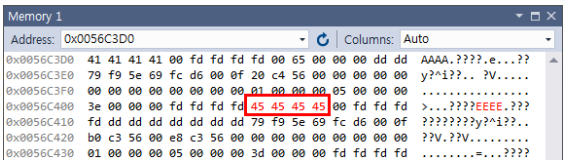

그림 8. overwrite

[그림 8]에서 a1.DDD에 "EEEE"를 쓴다. 당연하게도 a1.AAA와 a1.CCC 사이에 쓰여진다.

그림 9 Exploit 출력

[그림 9]에서 맨 마지막 "two(a1.BBB) : EEEE"는 a1.BBB를 출력한 결과 a1.DDD에 복사된 "EEEE"가 출력된다는 것을 확인한 것이다. 새로운 변수를 힙에 할당할 때, 그 변수가 가리키는 메모리 주소가 a1.BBB가 있었던 메모리 주소에 위치한다면 free()를 했음에도 불구하고, a1.BBB로 a1.DDD를 참조할 수 있다는게 증명되었다.

여기까지 스택과 힙을 기반으로 한 취약점 종류를 알아봤다.

III. 취약점 찾기

1장에서 대표적인 취약점들에 대해서 설명하였고 본 장에선 이러한 취약점들을 발견할 수 있는 방법론에 대해서 소개한다.

취약점 찾기에는 여러 가지 방법이 있다. 소프트웨어의 총체적인 권한을 가진 기업 또는 개인에게 허락을 받고 프로그램 작성 코드를 직접 열람하여 안전한 프로그래밍이 되어 있는지 확인하는 화이트 리스트 기반의 코드 분석, 코드 열람까지는 허용을 하지 않지만 보안 테스트까지는 허용하여 소프트웨어의 외부에서 페이로드를 넣어 버그를 유발하여 취약점을 찾는 블랙리스트 기반으로 나눌 수 있다. 이중 효율적이고 대부분의 취약점을 발견하는 방법인 블랙리스트 기반의 퍼징은 변형된 값을 넣어 프로그램을 실행시킨 후 오류가 발생한다면 취약점이 있다고 판단하고 수정할 수 있게 하는 도구다. 모든 외부 입력은 비신뢰성 값으로 간주하며, 이들의 데이터 흐름 통하여 취약한 메모리 영역의 추적이 가능하다.[2] 이러한 개념을 Taint Analysis라고 한다. 이번 장에선 자동화 기술의 종류, 그리고 그 중 하나의 원리를 소개한다.

3.1 자동화 기술

변형된 값을 넣어 랜덤으로 프로그램의 실행시키는 것이 퍼저라면 어떤 변형된 값을 넣을지 고민해야한다.

3.1.1 Instrumentation

해당 소프트웨어의 점검에 유용할 만한 정보들을 저장하고 변수를 입력하여 그에 대응하는 분기문을 거쳐서 취약점을 판별한다. 변수의 값을 주는 것이기에 해당 경로 이외의 경로로는 실행하지 않아서 실행되지 않은 경로의 취약점은 놓칠 수 있다.

3.1.2 Symbolic Execution

변수가 아닌 미지수를 넣어서 미지수에 들어갈 수 있는 변수를 다 넣어 취약점을 탐색한다. path condition에 맞춰준다고 하는데, 만약 분기문을 들어가는 중 한 곳에서 실행 경로가 겹쳐도 합쳐지는 것이 아니라 각각 실행된다.

3.1.3 Abstract Interpretation

변수가 가지는 값, 연산자, 실행 경로를 요약하여 어림잡아 전체적으로 분석하는 방법으로 정확도는 희생하지만 Symbolic Execution보다 빠른 속도를 가진다.

3.1.4 etc

앞서 나열한 3가지 방법 이외에 Symbolic 스타일의 Instrumentation을 가지는 Concolic testing, Taint analysis와 Abstract Interpretation을 합치는 등의 주제가 있다.

3.2 Symbolic Execution에 대한 설명

자동화 기법 중 하나인 Symbolic Execution에 대하여 설명한다. 3.1.2에서 기술하였듯이 분기의 모든 경로를 점검한다. 경로를 모두 지나가기 때문에 루프에 빠지기 쉽다. 이를 염두에 두고 루프에 빠지지 않게끔 퍼저를 구현하는게 제작자의 몫이다.

```
y = read()
y = 2 * y
if (y == 12)
    fail()
print("OK")
```

그림 10. Symbolic Execution 예시[3]

[그림 10]에서 read로 임의의 값을 가져와서 y가 12일 경우 fail이 되는 코드다.

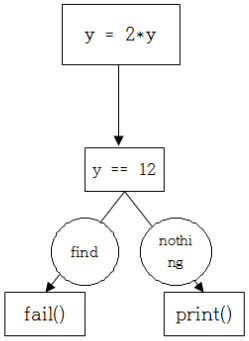

그림 11. Symbolic Execution 흐름도

[그림 11] 흐름도를 보면 y에 미지수를 대입하여 fail, print로 나눠지는 y==12 분기문에서 양쪽 모두 다 실행시킨다. 지속적으로 다른 변수가 대입되어 실행되다보면 y에 6이 대입되고 원하던 fail()에 도달한다.

간략하게 퍼징의 개념 및 방법에 대해서 살펴봤다.

Ⅳ. 결 론

기본적인 취약점의 종류와 퍼징, 그리고 취약점 분석의 자동화 기술에 대한 개념을 소개하고 그 중 하나인 Symbolic Execution에 대한 프로세스를 보았다. 이를 토대로 취약점 분석을 위한 절차를 이해하고 직접 자동화 분석 툴을 구현하여 다양한 취약점을 분석하고 찾아 사전에 해킹 공격에 대비한다.

감사의 글: 이 논문은 2015년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(과제번호: 2011-0023076). 또한 부산광역시에서 지원하는 BB21 과제에서 지원받았음

참고문헌

[1] 김지홍, 김휘강, “시스템 취약점 분석을 통한 침투 경로 예측 자동화 기법“, 정보보호학회, 22호 5권, 2012.10

[2] teamcrak, “Taint Analysis 연구분석보고서”, http://teamcrak.tistory.com/328, 2011.01.17

[3] wikipedia, “Symbolic execution”, https://en.wikipedia.org/wiki/Symbolic_execution