연구보고서	**10. 10. 31.** 작성

취약점 분석을 위한 퍼징(Fuzzing)

부경대학교 CERT-IS

김연재, 최종학, 신명재
이인회, 이재호, 최자운, 박지환

* 본 문서는 대학정보보호동아리연합회(http://www.kucis.org)의 지원을 얻어 작성된 문서입니다.

Table of Content

1. 서론

1.1. Fuzzing 이란?

영단어 “fuzzing” 을 사전에서 찾으면 다음과 같다.

- n., (pl. ~) U
  - 잔털, 솜털, (섬유 등의) 보풀
  - the fuzz on a peach 복숭아 의 잔털
  - 《구어》 고수머리;상고머리
  - 흐림《사진 등》
  - 《미·속어》 마약
- vi., vt.
  - 보풀이 나다[나게 하다], 부드럽게 되다[만들다], 훨훨 날아 흩어지다, 어물거리다, 애매하게 하다;《미·속어》 술취하다

Wisconsin-Madison 대학의 연구 프로젝트에서 처음 사용되었으며, 정보보안에서의 퍼징은 사전에 정의되어 있지 않다. 영단어 “fuzzing”으로부터 애플리케이션에 입력된 데이터가 보풀을 일으키어 어떠한 특정 상태를 만든다는 의미로 해석할 수 있다. 그래서 일반적으로 비정상적인 데이터를 애플리케이션에 전달하여 에러를 유도하는 방법을 퍼징이라 말한다.

비슷한 의미의 용어로 Blackbox Testing, Taint Analysis, BVA(Boundary Value Analysis) 등이 있다. 이 중에서 퍼징을 가장 잘 설명하는 용어는 Taint Analysis이다. 이 분석 방법은 오염된 데이터가 애플리케이션에 입력되어서 애플리케이션에서의 처리 단계별로 데이터들을 오염시킨다는 개념이다.

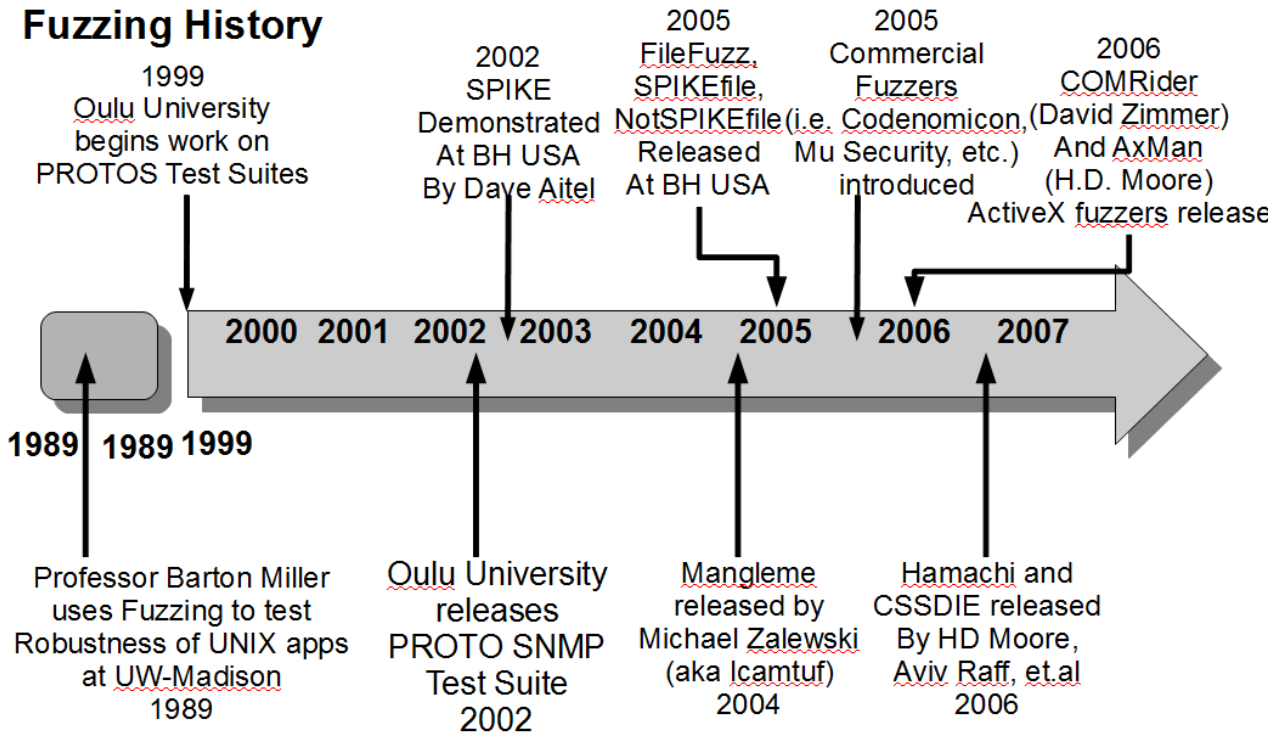

그림 1: 퍼징의 역사

1.2. 연구의 배경 및 목적

웹과 스마트폰의 보급이 활발해지면서 수많은 웹/응용 애플리케이션들이 범람하게 되었고 사용자들은 어떠한 의심도 없이 애플리케이션들을 사용하고 있다. 그래서 네트워크/시스템 해킹이 주를 이루던 과거와는 달리, 최근의 정보보안은 웹/응용 애플리케이션의 취약점을 파고드는 공격이 대세가 되었다. 그러나 많은 애플리케이션 개발자들은 자신이 만든 프로그램에 어떠한 취약점이 있는지를 완전히 파악하지 못하고 있어서 사용자들은 여러 위험에 노출되었거나 이미 피해를 입고 있다.

다음의 [그림 2]는 트위터와 국내 보안커뮤니티인 버그트럭과 보안인, 그리고 개발자 커뮤니티인 all of software에서 “퍼징을 이용하여 보안 취약점을 점검해본 적이 있으신가요?”라는 질문으로 설문을 한 결과이다.

그림 2: 퍼징에 대한 설문결과 (http://hisjournal.net/blog/340)

설문 결과에서 퍼징을 이용하여 취약점을 점검해본 적이 없다고 응답한 비율이 전체의 79%를 차지하고 있다. 퍼징은 소스 검증과 더불어 취약점을 찾는 강력한 기법 중의 하나이지만, 아직 국내에서는 많이 이용되는 것 같지는 않다.

개발자 입장에서는 퍼징을 이용하여 개발중인 애플리케이션의 취약점을 미리 찾아 예방하는 게 아주 중요하다. 이로소 제품의 신뢰도를 잃지 않을 수 있기 때문이다. 그리고 퍼징에 대하여 연구함으로써 시스템의 메모리 구조와 각종 파일들의 포맷 구조, 애플리케이션의 구동 과정 등을 상세히 알 수 있다. 그래서 우리는 웹, 네트워크, 파일 포맷, 메모리에 대한 퍼징을 연구하고 가능하면 직접 퍼징툴을 만들어서 취약점을 점검하는 것을 목표로 삼았다.

2. Web Fuzzing

2.1. 웹 퍼징이란 무엇인가?

퍼징은 공격대상 시스템의 보안 취약점을 탐지하기 위하여 목표 애플리케이션에 대한 다양한 입력 값을 전송하는 결함 주입에 근거한다. 이 기술을 사용할 때 목표 애플리케이션의 입력값 검증을 우회하며 동시에 보안 취약점을 탐지할 수 있도록 입력 값을 변경하는 것이 중요하다. 퍼징 기술은 SQL injection, Cross Site Scripting(XSS) 등과 같은 보안 취약점의 유형을 탐지한다.

2.2. 공격의 유형

2.2.a. XSS (Cross Site Script)

XSS는 다른 사용자의 정보를 추출하기 위해 사용되는 공격 기법이다. 간단히 예를 들면 게시판이나 검색 부분, 즉 사용자의 입력을 받아들이는 부분에 스크립트 코드를 필터링하지 않음으로써 공격자가 스크립트 코드를 실행할 수 있게 되는 것이다.

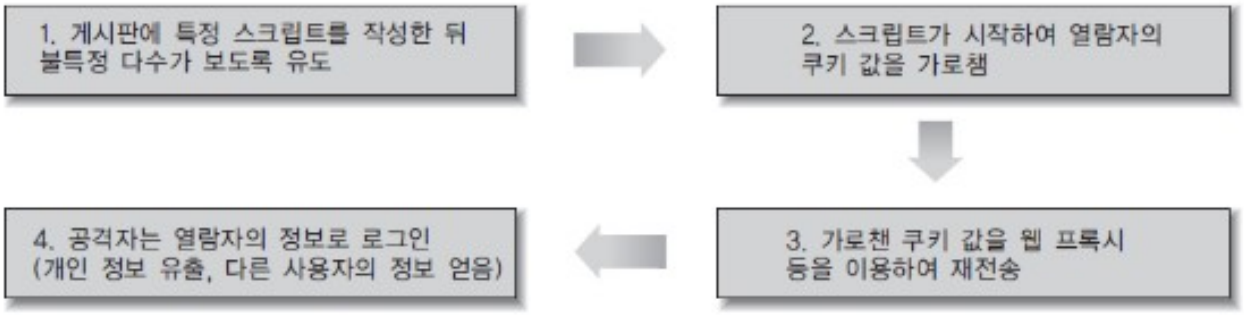

그림 3: XSS 공격 기법

[그림 3]은 실제 XSS 취약점을 이용하여 다른 사용자의 쿠키 값을 훔쳐 그 사용자로 로그인하는 과정을 순차적으로 표현한 것이다.

2.2.b. Injection SQL, parameter

SQL Injection은 웹페이지를 통해 입력된 파라미터 값을 이용하여 쿼리를 재구성 하는 방법이다. 많은 웹 페이지들은 사용자나 프로그램이 생성한 파라미터 값을 이용해 쿼리를 만들고 실행하는데, 이때 정상적인 값이 아닌 파라미터 값이 입력될 때 비정상적인 쿼리가 실행되며, 따라서 원하지 않는 결과가 나올 수 있다.

```
String param1 = request.getParameter ("user_id");
String param2 = request.getParameter ("user_pass");

rs = stmt.excuteQuery ("SELECT count(*) FROM user_t WHERE userid = '"+param1+"' AND userpw =
'"+param2+"'");
rs.next ();
if (rs.getInt (1) == 1)
{
        // 로그인 성공
}
else
{
        //로그인 실패
}
```

예제 1: SQL Injection에 취약한 로그인 인증 코드

[예제 1]에서 입력된 데이터가 ' or 1=1 'or 1=1 이라면 , 1=1 은 무조건 참이므로 SQL 문은 항상 참이다.

2.2.c. Directory Traversal

경로를 거슬러 올라가서 관리자가 유저에게 보일 생각이 없는 파일이나 디렉터리를 열람하거나 실행시키는 공격이다.

[그림 4]처럼 httpd.conf 파일에서 “Indexes” 옵션을 제거하지 않고 디렉토리 리스팅을 허용할 경우, 사용자에게 노출되서는 안 될 파일들이 노출된다.

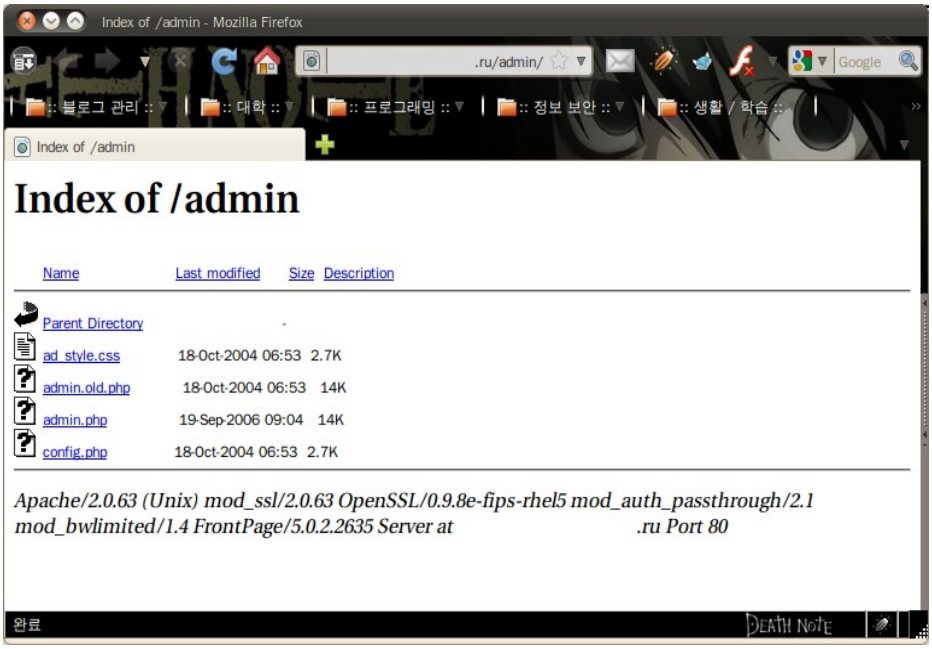

그림 4: Directory Traversal

2.2.d. 서버 인증 우회

인증 설정 중에 Limit 속성으로 메소드를 제한 할 수 있는데, 일반적으로 GET 메소드와 POST 메소드만을 설정한다. 이럴 경우 인증이 GET, POST 메소드만 적용이 되므로 그 이외의 메소드를 사용하면 인증을 우회 할 수 있게 된다.

메소드 종류

- OPTIONS - 서버에서 어떤 메소드를 지원하는지 볼 수 있다.
- HEAD - 웹 서버의 HEADER을 보여준다.
- PUT - 지정된 URL에 전달된 자료를 저장한다.
- DELETE - 지정된 리소스를 삭제한다.
- TRACE - 클라이언트가 송신한 request를 그대로 리턴 한다.

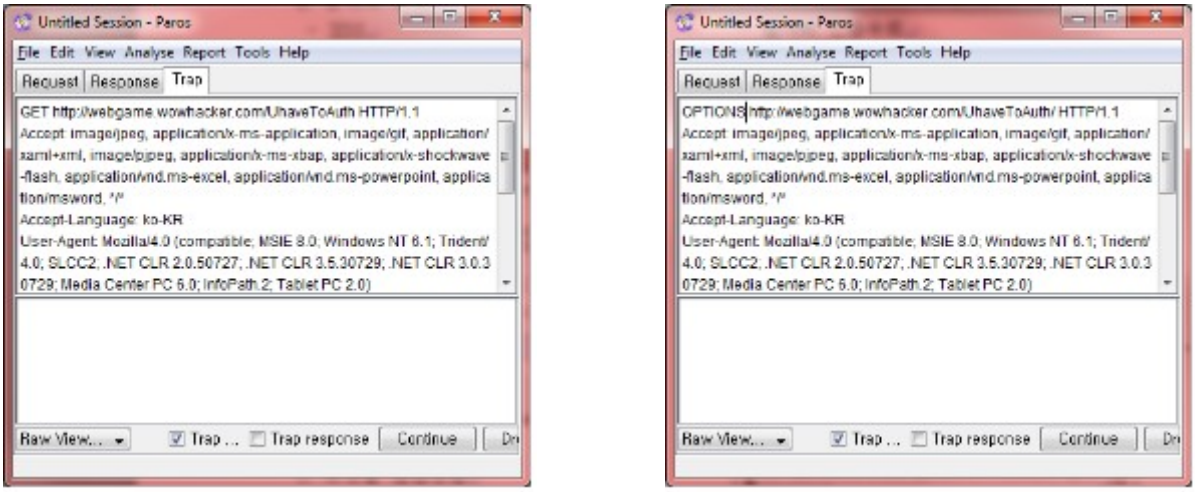

그림 5: GET -> OPTIONS 메소드로 로그인 인증 우회

2.3. 유형별 퍼징 방법

2.3.a. XSS, Injection SQL, parameter

(1) 해당서버의 80번 포트에 접속하여 form 태그를 parsing하여 전송할 파라미터의 이름(Parameter_name)과 목적지 페이지(DEST), 전송방법(GET/POST)을 찾아낸다.

```
<FORM method=[GET/POST] action=[DEST]>
<input type=text name=[Parameter_name]>
</FORM>
```

예제 2: 페이지 내의 FORM 태그

(2) GET 방식의 경우는 아래와 같은 형식으로 패킷을 생성한 뒤 목적지 페이지를 GET 방식으로 요청한다. 각 파라미터의 값은 *'sql* 이나 *">XSS* 와 같은 에러가 발생할만한 데이터를 넣는다.

```
GET DEST?[Parameter_name]='sql /HTTP/1.1
HOST : DEST
```

예제 3: GET 방식 요청

(3) POST 방식은 미리 생성한 아래와 같은 형식으로 패킷을 생성하며, [BODY] 부분의 길이를 구하여 Content-Length에 명시하고 POST 방식을 써서 웹서버에 DEST 페이지를 요청한다. 각 파라미터의 값은 GET 방식처럼 *'sql* 이나 *">XSS* 와 같은 에러가 발생할만한 데이터를 넣는다.

```
<HEADER>
POST DEST
HOST : DEST
Content-Length : 00
<BODY>
[Parmeter_name]='sql ......
```

예제 4: POST 방식 요청

(4) 요청 후 전송되어 온 패킷을 분석하여 HTTP 헤더가 "200 OK"를 포함하면 적절한 에러 핸들링이 된 것이고 400, 403, 500 에러를 포함하면 취약점이 있다고 판단한다. 혹은 응답받은 페이지 내에 "XSS" 와 같은 삽입한 데이터가 포함되면 XSS 취약점이 있다고 판단한다.

2.3.b. Directory Traversal

요청 홈페이지에서 보내온 데이터를 분석하여 [표 1]의 문자열을 포함하면 취약점이 있다고 판단한다. 혹은 Google 에서 해당 홈페이지의 URL을 intitle 옵션으로 [표 1]의 문자열을 검색하여 판단할 수 있다.

웹 서 버	디렉토리 노출 패턴
IIS	Parent Directory,
Tomcat	Directory Listing
Apache	Directory Listing
기 타	Index of/

표 1: 웹서버별 디렉토리 노출 패턴

2.3.c. 서버 인증 우회

(1) POST, GET 이외의 메소드(PUT, OPTIONS, HEAD 등)로 해당 페이지를 요청해 본다.

(2) 요청 후 전송되어 온 패킷을 분석하여 HTTP 헤더가 “200 OK”를 포함하면 해당 메소드로 요청시 인증 우회가 가능할 것으로 판단한다.

3. Network Protocol Fuzzing

3.1. 네트워크 프로토콜 퍼징이란 무엇인가?

네트워크 프로토콜 퍼징은 서버의 데몬을 대상으로 하여 조작된 패킷을 전송하는 것을 말한다. 서버의 데몬은 소켓을 통하여 클라이언트와 통신을 하는 어플리케이션이다. 클라이언트에서 서버로 보낸 메시지는 데몬에서 받아서 파싱을 한 후, 연산 처리가 된다. 그리고 필요에 따라서 데몬은 다시 클라이언트에게 메시지를 보내기도 한다.

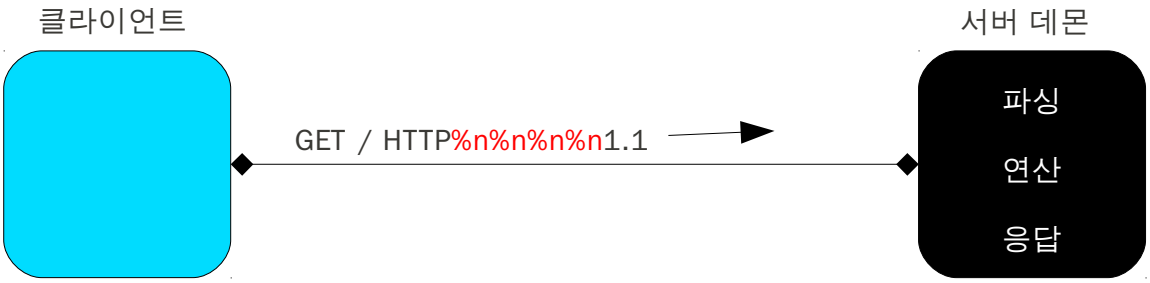

그림 6: 클라이언트와 서버 데몬

데몬 역시 사람이 만든 어플리케이션인데다, 클라이언트로부터 받은 메시지를 파싱하는 과정, 즉 사용자의 입력이라는 과정을 거치므로 취약점이 존재할 수 있다. 그렇기 때문에 네트워크 프로토콜 퍼저는 클라이언트의 입장에서 메시지를 조작하여 소켓을 통해 서버로 전송한다.

3.2. 네이트온 프로토콜

3.2.a. Notification Server 인증

네이트온은 Update Server, Dispatch Server, Notification Server 와 Session Server 이렇게 4개의 서버를 통해 TCP 통신이 이루어진다. Update Server 는 네이트온 메신저가 실행될 때 처음 접속되는 서버로, 메신저의 버전 확인과 업데이트를 처리한다. Dispatch Server 는 네이트온의 서버에 과부하가 걸리는 것을 방지하기 위해 사용자들을 분산시키는 역할을 한다. 실제로 네이트온에 접속하기 위해서는 Notification Server(이하 NS) 를 통해야 한다.

NS 를 통해 인증을 받기 위해서는 먼저 ssh 티켓을 받아야 한다. "*https://nsl.nate.com/client/login.do?id=[id]&pwd=[passwd]*" 으로 접속하면 응답메시지로 받을 수 있다.

메 시 지	설 명
LSIN [트렌젝션 번호] [id] [ticket] [기타 정보들]\r\n	사용자 인증
GLST [트렌젝션 번호] [그룹 번호]\r\n	그룹 리스트 요청
LIST [트렌젝션 번호]\r\n	친구 리스트 요청
INFY [트렌젝션 번호] [친구 id] [친구 상태]\r\n	접속중인 친구 알림
PING [트렌젝션 번호]\r\n	연결상태 확인

표 2: 인증에 필요한 메시지들

[표 2] 는 인증을 위해 서버와 클라이언트 간에 주고받는 메시지들이다. 이들 메시지는 TCP 소켓으로 통신이 이루어진다.

LSIN 은 실제 사용자 인증을 위한 메시지이다. 파라미터로 트렌젝션 번호, 사용자 계정의 ID, 방금 전 받은 ssh 티켓, 인코딩 방식 정보, 클라이언트의 빌드넘버, 문자열 인코딩 방식 정보 등이 넘겨진다. 인증이 제대로 이루어지면 아래와 같은 응답 메시지를 받을 수 있다.

LSIN [트렌젝션 번호] [클라이언트 ID] [실명] [대화명] [휴대폰번호] [unknown] [SMS 인증코드] [미니홈 ID] [unknown] [unknown] [unknown] [unknown] [사용자 계정 ID] [email] [unknown] [통서비스 사용여부] [unknown] [국가] [unknown] [unknown] [unknown]\r\n

예제 5: NS에 접속하기 위한 LSIN 메시지

GLST 는 그룹 리스트를 요청하는 메시지이고, LIST 는 친구 리스트를 요청하는 메시지이다. 사실상 그룹 리스트는 필요 없고, 유저 리스트를 받아야 실제로 인증이 이루어진다. 인증이 이루어지면, INFY 응답 메시지를 받을 수 있다.

인증이 완료되면 서버는 클라이언트에게 10초 주기로 PING 메시지를 보낸다. 서버가 PING 메시지를 8번 보내는 동안 클라이언트에서 응답이 없으면, 서버가 강제로 소켓을 닫아서 연결이 끊어지게 되어 있다.

3.2.b. 쪽지 전송

메 시 지	설 명
CMSG [트렌젝션 번호] N [쪽지의 길이]\r\n [친구의 계정 ID]\r\n IMSG\r\n title:[쪽지의 제목]\r\n from:[사용자의 계정 ID]\r\n ref:[친구의 계정 ID]\r\n data:[현재 연도, 날짜, 시간]\r\n session_id:\r\n uuid:\r\n contenttype:\r\n length:[쪽지 내용의 길이]\r\n font-name:\r\n font-style:\r\n font-size:\r\n font-color:\r\n ref-uuid:\r\n cookie:\r\n event:NW\r\n \r\n [쪽지의 내용]\r\n	쪽지 전송

표 3: 쪽지 전송에 필요한 CMSG 메시지

쪽지를 전송하기 위해서는 [표 3] 과 같이 CMSG 메시지 NS 소켓으로 보낸다. 쪽지의 전송이 성공적으로 이루어지면, 아래와 같은 응답 메시지를 받을 수 있다.

CMSG [트렌젝션 번호]\r\n

예제 6: 쪽지 전송이 성공적일 때의 응답 메시지

3.2.c. Session Server 인증

그룹/유저 리스트를 받아오거나 쪽지를 보내는 것은 NS에서 처리되지만, 대화와 파일 전송은 Session Server(이하 SS)에서 통신이 이루어진다. SS에 접속하려면 NS에서 RESS 메시지를 전송해야 한다.

RESS [트렌젝션 번호]\r\n

예제 7: NS에 접속하기 위한 RESS 메시지

[예제 7] 과 같이 RESS 메시지를 NS로 보내면 서버로부터 RESS 메시지를 응답받는다.

메 시 지	설 명
RESS [트렌젝션 번호] [ip] [port] [ticket]\r\n	사용자 인증

표 4: 서버로부터 응답받은 RESS 메시지

서버로부터 응답받은 RESS 메시지에는 [표 4] 처럼 SS의 IP와 Port, 대화에 필요한 티켓이 있다.

3.2.d. 대화

SS는 사용자와 사용자 사이에 세션을 연결해주는 역할을 한다. 다른 사용자와 대화(Chatting)를 하기 위해서는 이 SS에 접속하고 상대방을 초대해서 세션을 연결해야 한다.

메 시 지	설 명
ENTR [트렌젝션 번호] [account] [nickname] [ticket] [기타 정보들]\r\n	대화세션 입장
CTOC [트렌젝션 번호] [target] [unknown] [payload 길이]\r\n	데이터 전송
INVT [account] [ip] [port] [ticket]\r\n	대화세션 초청
MESG [트렌젝션 번호] [state] [메시지]\r\n	메시지 전달
QUIT [트렌젝션 번호]\r\n	대화세션 종료

표 5: 대화에 필요한 메시지들

먼저, 사용자는 ENTR 메시지로 대화세션에 입장을 해야 한다. 이 때 필요한 것이 RESS 메시지로부터 받은 티켓이다. 대화세션에 입장했으면 상대방을 초청하기 위해서 CTOC 메시지로 상대방에게 INVT 데이터를 전송한다. CTOC 메시지는 상대방에게 데이터를 전송하는 메시지로 데이터의 길이를 알아야 한다. INVT 는 메시지가 아니라 데이터이다. 그래서 트렌잭션 번호는 필요 없다. INVT 데이터에도 RESS 메시지로부터 받은 티켓이 필요하다.

상대방과 대화세션이 연결되면 MESG 메시지로 메시지를 전달할 수 있고, QUIT 메시지로 대화세션을 닫을 수도 있다.

3.3. 퍼징 방법

3.3.a. 메시지 조작

메시지의 조작, 즉 패킷의 조작은 여러 가지 방법으로 이루어진다.

(1) 단순히 메시지의 순서를 무작위로 섞거나 null 문자를 삽입하거나 % 인코딩을 이용하기도 한다. 이러한 방법은 흔히 TCP 나 UDP 윗단의 애플리케이션 계층의 프로토콜을 퍼징할 때 이용된다. 실상, 대부분의 서버 데몬들은 어플리케이션 층에서 동작하므로 이러한 방법들만으로도 퍼징의 효과를 볼 수 있다.

TCP, UDP 소켓을 이용하여 퍼징을 할 때는 퍼징의 대상으로 삼을 서버의 프로토콜에 대한 이해 정도가 가장 중요하다. 프로토콜의 이해가 없으면 퍼징은 무의미하기 때문이다. 예를 들어서 서버의 데몬은 “MESG” 라는 메시지만을 파싱하여 연산 처리를 한다고 가정을 한다면, 퍼저가 아무리 수많은 종류의 패킷을 보내더라도 “MESG” 라는 단어가 포함되어 있지 않아서 데몬은 이 패킷을 무시할 수 있기 때문이다.

(2) 더 깊이 들어가기 위해서는 로우 소켓을 이용하여 TCP, UDP 헤더를 직접 조작하거나 IP 계층까지도 조작하여 퍼징을 해야 한다. 로우 소켓을 이용하는 방법은 TCP, UDP 소켓을 이용하는 방법보다 더 까다롭고 네트워크 장치를 직접 다루기 때문에 OS의 권한 문제도 해결해야 하지만, 어플리케이션 계층의 프로토콜이 아닌 패킷 그 자체를 직접 퍼징하기 때문에 더 좋은 결과를 낼 수 있다. 하지만 이 방법은 프로토콜 퍼징이라기 보다는 네트워크 자체에 대한 퍼징으로 간주해야 한다.

3.3.b. fuzzNateon.py 구현

```
#!/usr/bin/python
# -*- coding:utf-8 -*-

import sys
```



```
import urllib
import getpass
from protoFuzz import *

ns_host = "211.234.240.140"
ns_port = 5004

print "============================================================================"
print ""
print "[+] Fuzzing NateOn"
print "[+] This script was written by 6l4ck3y3 (@6l4ck3y3)"
print ""
account = raw_input ("[+] account: ")
passwd = getpass.getpass ("[+] passwd: ")
target = raw_input ("[+] target: ")
print ""
print "============================================================================"
print ""

# Authentication for NS
u = urllib.urlopen ("https://nsl.nate.com/client/login.do?id=" + account + "&pwd=" + passwd)
ns_ticket = u.read ().split ('\r\n')[1]

ns = protoFuzz (ns_host, ns_port)
ns.sendMsg ("LSIN 0 " + account + " " + ns_ticket + " SSL")
ns.sendMsg ("LIST 0")

# Authentication for SS
ns.sendMsg ("RESS 0")
recv = ns.getMsg ().split ('\r\n')[0].split (' ')
ss_host = recv[2]
ss_port = recv[3]
ss_ticket = recv[4]
ss = protoFuzz (ss_host, int (ss_port))

# Chat
ss.sendMsg ("ENTR 0 " + account + " %00 %00 " + ss_ticket + " UTF8 P")
ns.fuzzMsg ("CTOC 0 " + target + " A 4096\r\nINVT " + account + " " + ss_host + " " + ss_port + " "
+ ss_ticket)
```

```
ss.close ()
ns.close ()
```

예제 8: fuzzNateon.py

[예제 8] 은 분석한 네이트온 프로토콜을 바탕으로 *whitespace*나 " *\r\n*" 등을 기준으로 메시지 내용을 쪼개어서 각 데이터에 에러가 발생할만한 쓰레기 데이터를 삽입하는 방법을 이용하는 예다.

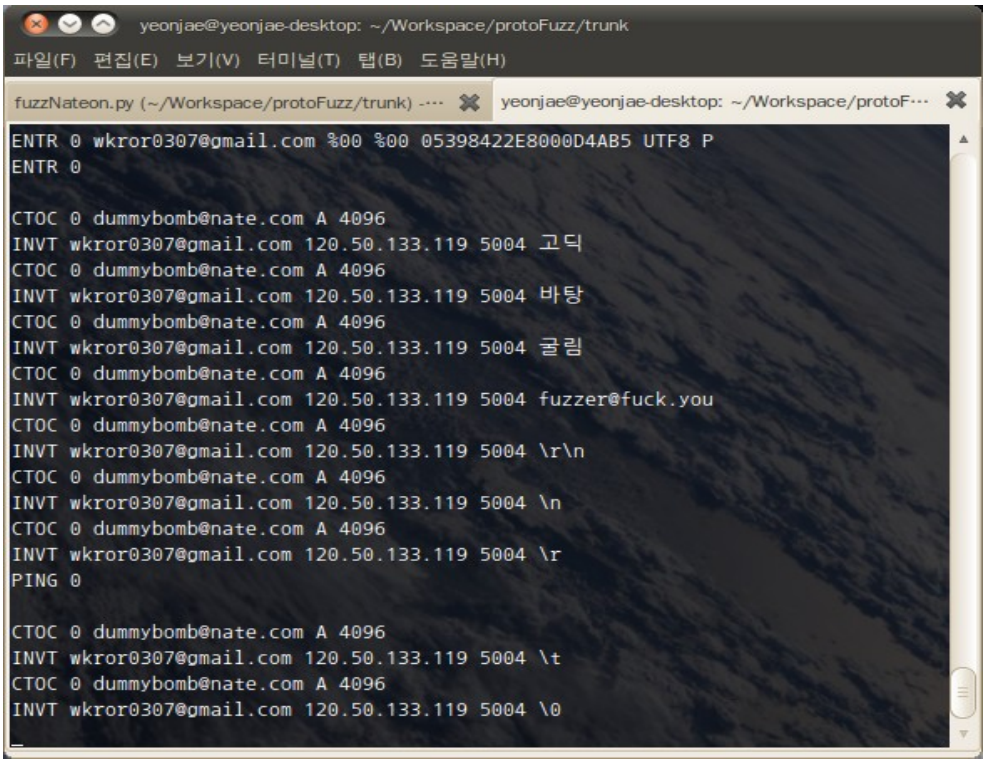

그림 7: CTOC + INVT 에 대한 퍼징 수행

[예제 8] 에서는 CTOC 메시지와 INVT 데이터 조합에 대해서만 퍼징을 시도하지만, 통신에 이용되는 모든 메시지에 대해서 fuzzMsg() 메소드를 적용할 수 있다.

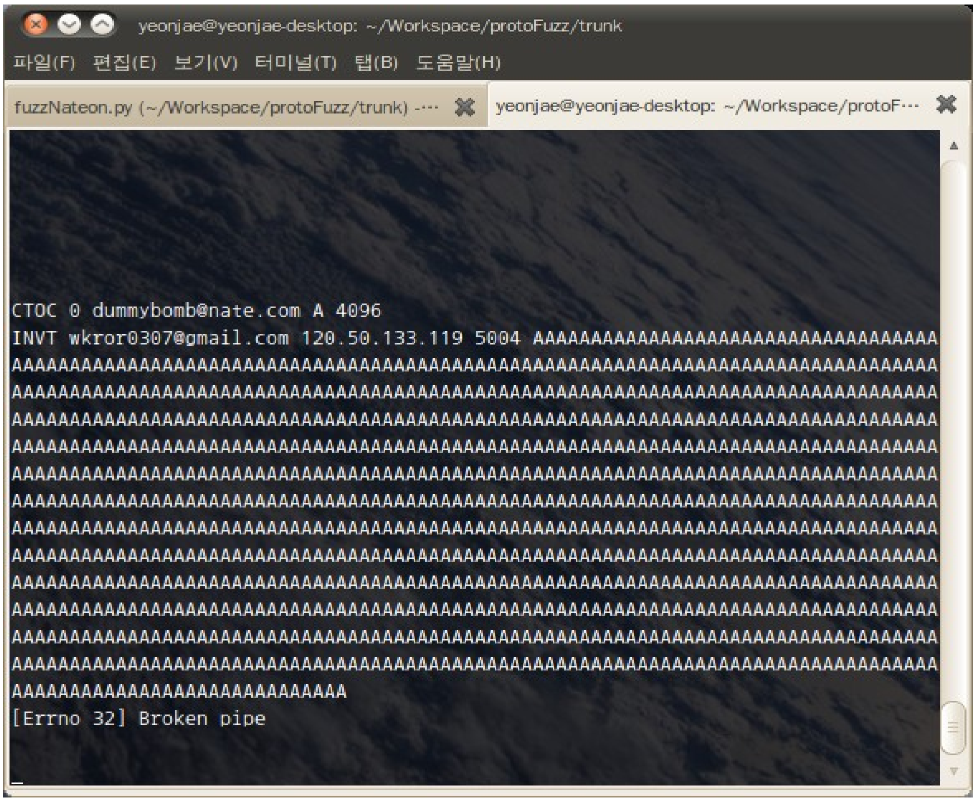

그림 8: 퍼징 중에 소켓 에러 발생

Fuzzer를 돌리면서 서버로부터 에러코드가 반환되거나 소켓에러가 발생하면 퍼징이 성공하였다고 판단한다. 그러나 그것이 바로 취약점이라고 단정할 수는 없다. 네이트온의 경우는 메시지가 부정확하다고 서버가 판단하면 자동으로 소켓을 닫아버리기 때문에 그런 경우에도 소켓에러가 발생한다.

4. File Format Fuzzing

4.1. 파일 퍼징이란 무엇인가?

문서편집기, 동영상 재생기, 압축프로그램, 토렌트 클라이언트 등등의 많은 애플리케이션들은 파일을 읽고 파싱하고 메모리에 읽어들여서 처리한다. 각 파일들은 자체의 포맷을 가지고 있고 애플리케이션은 그 포맷에 맞추어서 파싱을 한다. 이런 과정에서 애플리케이션은 취약점을 가질 수 있다. 포맷만 봐서는 정상적인 파일이지만, 그 내부에 에러를 발생할만한 쓰레기 코드가 숨을 수 있다. 파일 퍼징은 이러한 취약점을 유발할 수 있는 쓰레기 코드가 존재할 수 있는지를 검사하는 작업이다.

4.2. 퍼징 방법

4.2.a. FileFuzzer 설계

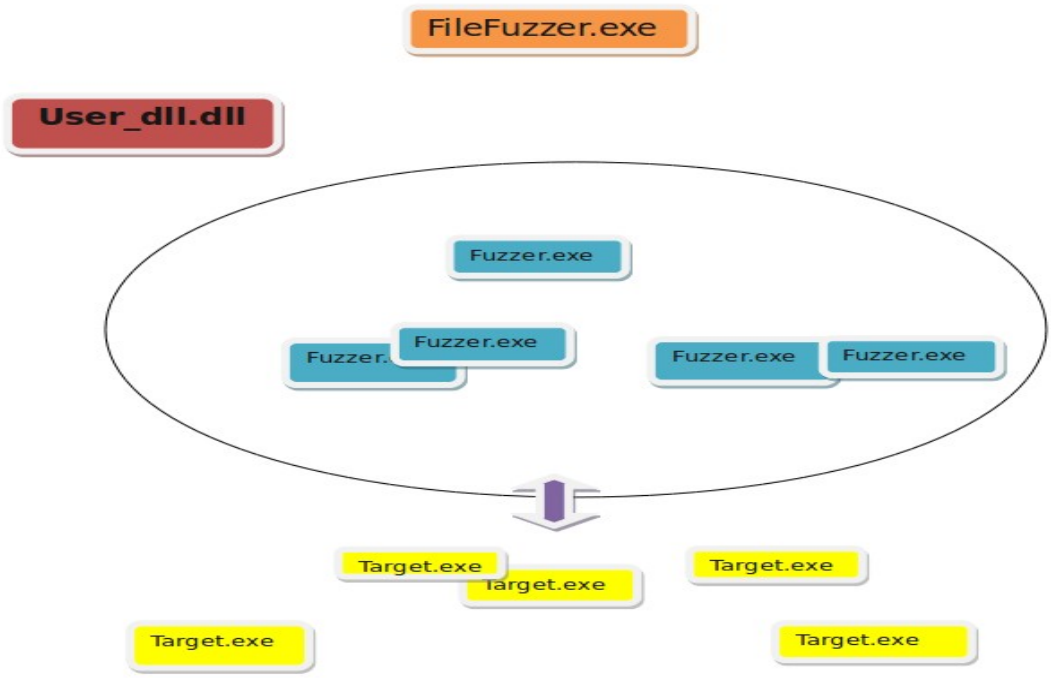

그림 9: FileFuzzer Prototype

[그림 9] 와 같이 FileFuzzer는 여러 개의 프로세스를 만들고 그 자식 프로세스들은 User_dll.dll 의 몇 가지의 함수로 파일을 변경하면서 타겟 프로세스를 디버거로 동작함으로써 예외 상황의 발생을 감지하고 예외가 발생하였을 경우 그 때의 파일과 정보를 저장한다.

4.2.b. FileFuzzer 구현

```
int main( int argc, char **argv )
{
	if( argc < 2 ) return 0;

	FILE *fs;

	fs = fopen( argv[1], "r" );

	char buf[ 64 ];

	fgets( buf, 1024, fs );
	printf( buf );

	fclose( fs );
	return 0;
}
```

예제 9: test.exe

퍼징 프로그램을 테스트하기 위해 [예제 9] 와 같은 프로그램을 설계한다. 물론 의도적으로 buf의 크기는 64byte로, 읽어들이는 크기는 1024byte로 설정하여 버퍼 오버플로우가 일어 날 수 있도록 한다.

그림 10: test.exe

파일이 정상적이라면 [그림 10] 처럼 동작을 할 것이다.

```
extern "C" __declspec(dllexport) int ChangeFile()
{
	/* at this time you can change file for attack */
	/* return value is used for time to die */
```

```
	/* if return 0, fuzzer is end other is wait return value and die */
	HANDLE hAttackFile;
	DWORD dwWrited;
	char data[] = "0123456789";

	hAttackFile = CreateFileA(	attackFilePath,
					GENERIC_READ|GENERIC_WRITE,
					FILE_SHARE_READ,
					NULL,
					OPEN_EXISTING,
					FILE_ATTRIBUTE_NORMAL|FILE_FLAG_WRITE_THROUGH,
					NULL	);

	SetFilePointer( hAttackFile, 0, 0, FILE_END );
	WriteFile( hAttackFile, data, sizeof(data)-1, &dwWrited, NULL );
	CloseHandle( hAttackFile );
	if( count– ) return (2000);
	return 0;
}
```

예제 10: Default_dll.dll

그리고 [예제 10]과 같이 dll 에 함수를 export 한다. 반환되는 값은 프로그램을 실행시키고 얼마나 대기할 것인가를 나타낸다. 따라서 2초 동안 대기하고 그 시간 안에 예외가 발생하지 않는다면 정상 실행으로 판단하고 종료시킨다. count 는 초기값이 5로 정해져있고 저 값으로 인해서 5번만 반복하고 종료한다

```
C:₩Test>FileFuzzer.exe
usage : FileFuzzer.exe execute_program pattern_dll [numOfProcess] [init_file]

		execute_program : open file program( recommand full path )
		pattern_dll		: pattern_dll is have some function
					the first is BOOL SetFile( HANDLE )
						it give you file descript are modify
					the second is BOOL ChangeFile( )
						chage the file, if changed return true
							else return false
					the third is FILE NewAttackFile( TCHAR* )
						call this function if not input init_file
						make initFile and return descripter of file
		numOfProcess	: how many process? ( default is 7 )
		init_file		: init_file

C:₩Test>FileFuzzer.exe test.exe default_dll.dll 1 test.txt
```

그림 11: FileFuzzer 실행 화면

[그림 11] 처럼 퍼저가 실행되면 info.fdb 라는 파일이 생성되고 info.fdb에 간단한 정보가 기록된다.

```
typedef struct _SAVE_INFO{
        char dllInfo[ MAX_PATH ];
        unsigned int info_size;
} SAVE_INFO, *PSAVE_INFO;

typedef struct _DEFAULT_INFO{
        unsigned int result;
        unsigned int file_size;
} DEFAULT_INFO, *PDEFAULT_INFO;

typedef struct _CUSTOM_INFO{
        int cbSize;
} CUSTOM_INFO, *PCUSTOM_INFO;
```

표 6: info.fdb 의 구조

info.fdb 의 구조는 [표 6] 과 같다. CUSTOM_INFO 값은 dll 에서 제공해야지만, 제공하지 않을 것이기 때문에 그 크기는 항상 0으로 정해져 있다고 볼 수 있다.

4.2.c. Info.fdb 분석

```
000000F0  FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE  þþþþþþþþþþþþþþþþ
00000100  FE FE FE FE 08 00 00 00 05 00 00 C0 4B 00 00 00  þþþþ.......ÀK...
00000110  22 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 21 22 30  "Hello World!!"0
00000120  31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36  1234567890123456
00000130  37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32  7890123456789012
00000140  33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38  3456789012345678
00000150  39 30 31 32 33 34 35 36 37 38 39 64 65 66 61 75  90123456789defau
```

그림 12: HexEditor 이용해서 본 info.fdb의 내용의 일부

[그림 12] 는 Filefuzzer 가 수행되고 생성된 info.fdb 의 데이터이다. 0x104 에서 dll 이름이 끝나고, 그 다음 값인 info_size의 값이 항상 8byte 라는 것을 확인할 수 있다. 그 다음은 DEFAULT_INFO 구조체이고, 첫번째 값인 result 는 0xC0000005 로서 메모리 접근 위반으로 프로그램이 죽었다는 것을 알 수 있다. 그리고 그 때의 파일 크기는 0x0000004B (= 75d) 이고, 이 파일은 DEFAULT_INFO 구조체가 끝나고 CUSTOM_INFO.cbSize 다음에 위치해 있다. 따라서 75byte 만큼의 크기라는 것을 알 수 있다.


```
"Hello World!!"01234567890123456789012345678901234567890123456789012345678901234567890123456789
```

예제 11: 에러를 발생할만한 데이터

따라서 [예제 11] 의 데이터로 파일을 만들어준 후 실행을 해보면 [그림 13] 과 같은 결과를 볼 수 있다.

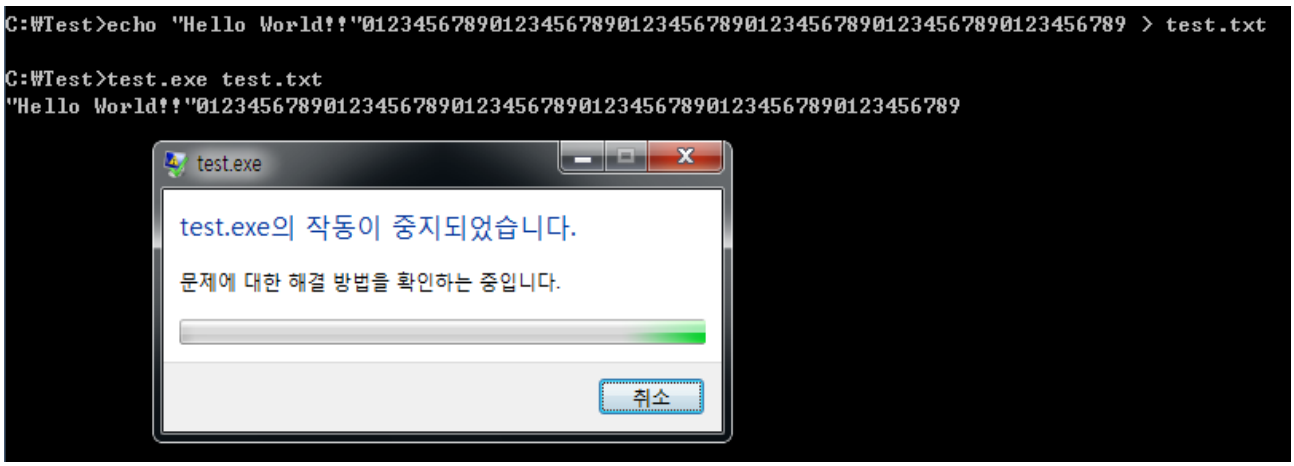

그림 13: 에러 발생 확인

5. Memory Fuzzing

5.1. 메모리 퍼징이란 무엇인가?

프로토콜이나 파일포맷과 달리 좀더 기본적인 소스 코드에 초점을 맞춘 작업이며, 보통 소스 검증과 병행되는 작업이다. Taint Analysis라고도 불리우고, 데이터 통신보다 타겟 프로그램의 메모리에 존재하는 변이되어 생성되는 데이터들에 초점을 맞추어 작업을 진행해 나가야 한다.

5.2. 메모리 구조

5.2.a. Microsoft Windows 메모리 모델

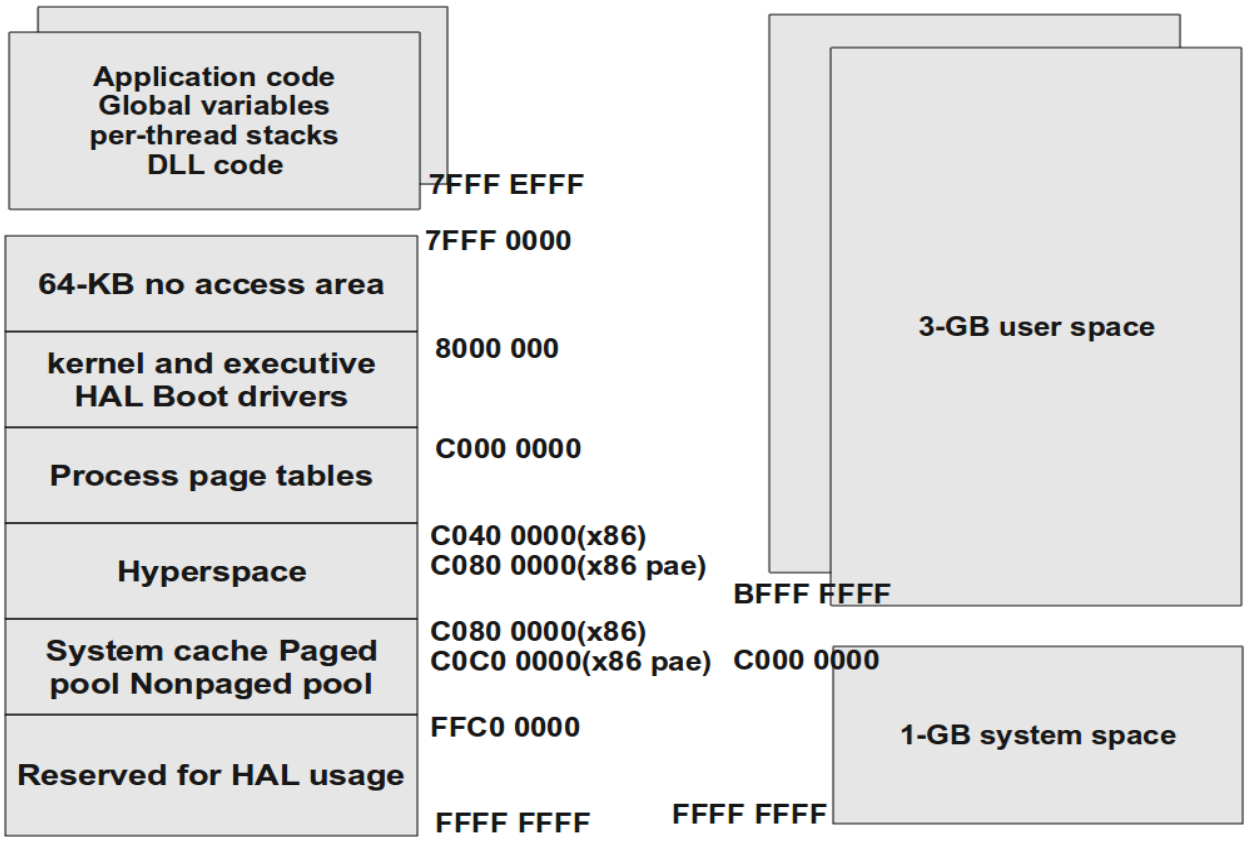

그림 14: Windows 메모리 맵

Windows 95 이래로 32-bit 플랫폼에서는 총 4GB의 주소 메모리를 제공하고 있다. 이 4GB의 메모리는 Kernel 영역과 User 영역으로 나뉘어져 있으며 일반적으로 3:1의 비율로 3GB는 User 영역, 1GB는

Kernel 영역으로 나누어 진다. 이에 대한 자세한 설정은 boot.ini에서 설정이 가능하다.

이러한 메모리에는 각각의 권한이 존재하는데, Windows 에서는 다음과 같은 권한을 제공한다.

- PAGE_EXECUTE
- PAGE_EXECUTE_READ
- PAGE_EXECUTE_READWRITE
- PAGE_NOACCESS
- PAGE_READONLY
- PAGE_READWRITE
- PAGE_GUARD

5.3. 퍼징 방법

5.3.a. 메모리 퍼징을 위한 방법들

메모리의 퍼징을 위해서 크게 두 가지의 방법론이 존재한다.

- 변조된 반복문의 삽입 (Mutation Loop Insertion)

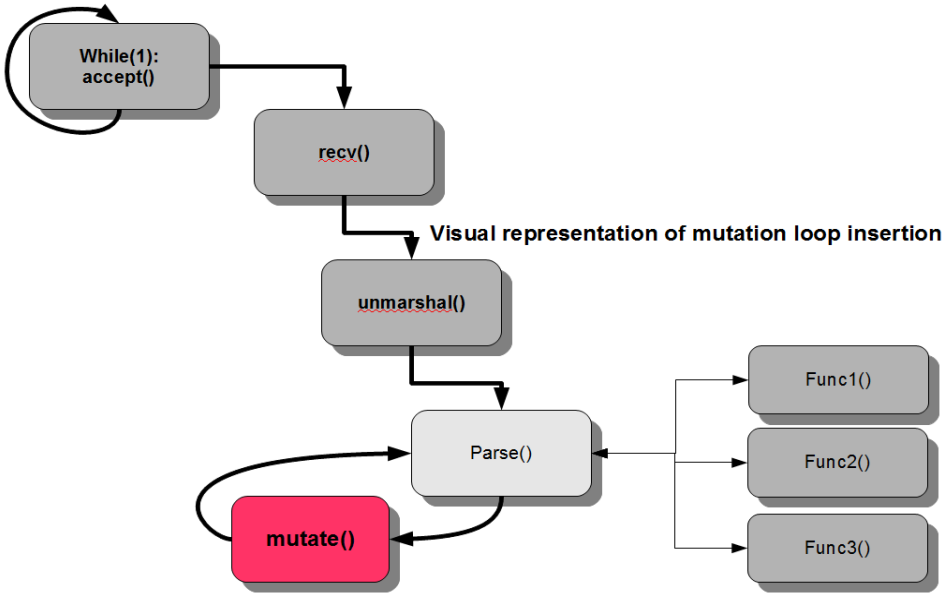

그림 15: Method: Mutation Loop Insertion

- 스냅샷과 변조된 영역의 복구 (Snapshot Restoration Mutation)

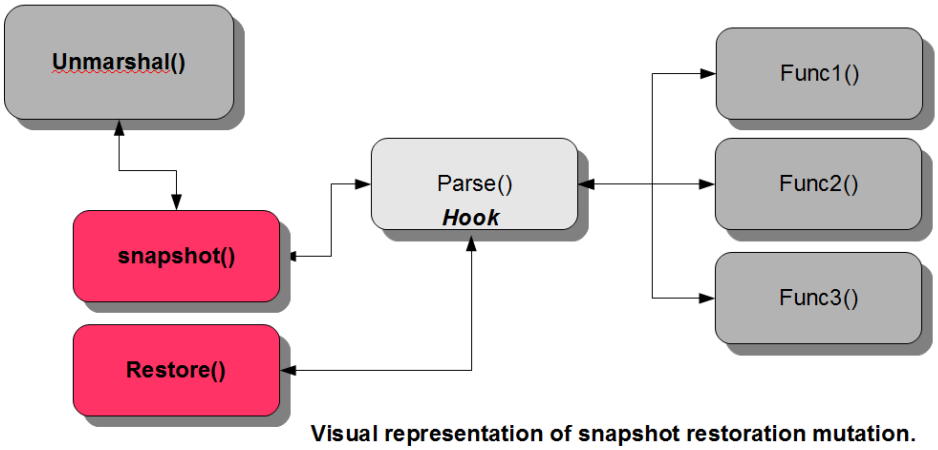

그림 16: Method: Snapshot Restoration Mutation

메모리에서의 퍼징을 위해서 다음과 같은 단계가 필요하다.

- 디버거(debugger) 변수들의 초기화
- 프로세스를 디버거에 붙임
- 후킹을 설정
- Entry point에서의 메모리 상태를 저장
- 프로세스에 존재하는 함수 인자값들의 메모리 접근을 모니터하고 주소를 저장
- 프로그램이 종료지점에 도달했을 경우 저장된 상태로 되돌림
- virtual_alloc()을 이용하여 fuzz 문자열이나 버퍼의 공간을 할당한다.
- 함수 인자값들을 fuzz data로 변경한다.
- 스택을 관찰한다.
- 기다린다....

[그림 17] 은 Fuzzing Routine을 개략적으로 나타낸 그림이다.

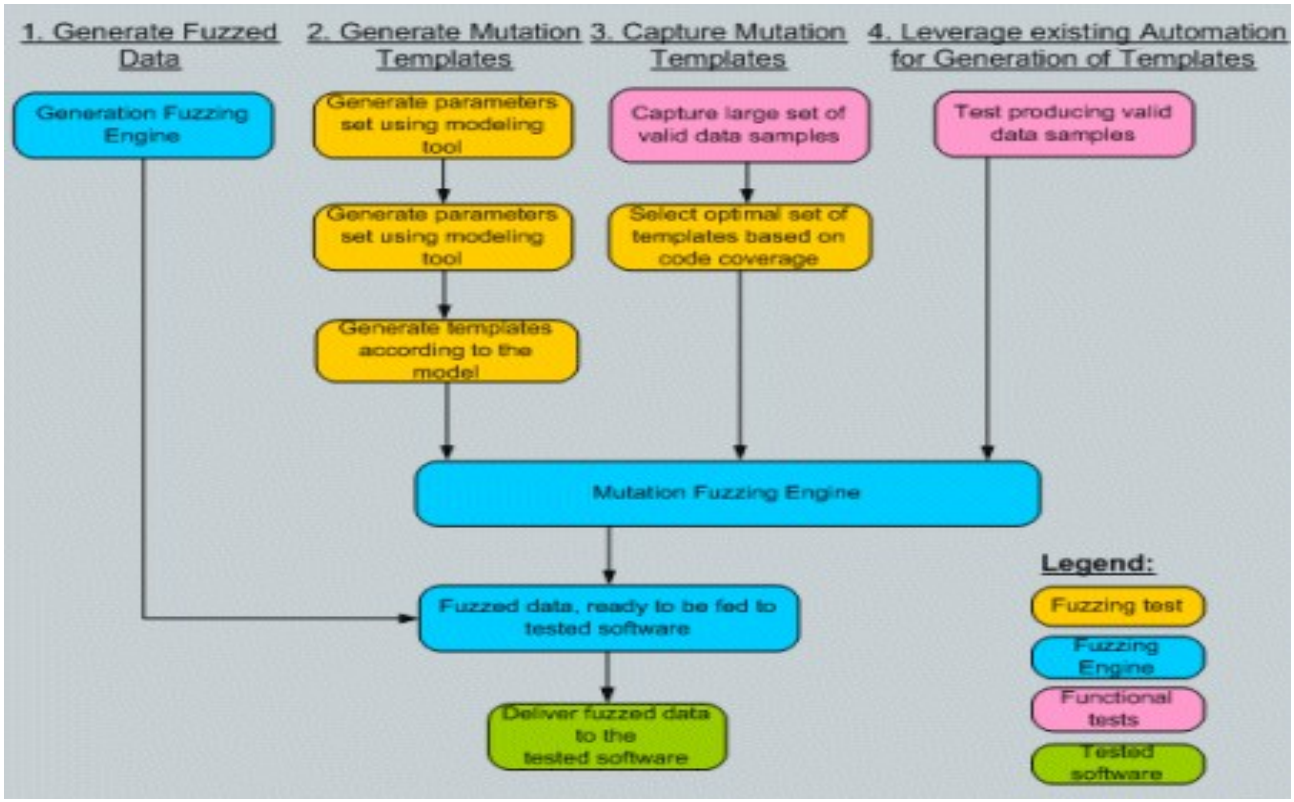

그림 17: Fuzzing Routine

5.3.b. 메모리 퍼징 도구

(1) Linux의 /dev/urandom 을 이용하여 임의 랜덤 값으로 퍼징을 수행 할 수 있다.

(2) python 을 이용한 프레임워크인 PyDbg 로 퍼징을 수행할 수 있다.

```
#!c:\python25\python.exe

from pydbg import *
from pydbg.defines import *

import time
import random

snapshot_hook       = 0x77143540
restore_hook        = 0x77143540
snapshot_taken      = False
hit_count           = 0
```



```
address                 = 0

def set_entry(pydbg):
        return 1

def handle_bp(pydbg):
        global snapshot_hook, restore_hook
        global snapshot_taken, hit_count, address

        if pydbg.first_breakpoint:
                return DBG_CONTINUE

        print "ws2_32.recv() called from thread %d @%08x" % (pydbg.dbg.dwThreadId, \
                                                              pydbg.exception_address)

        context_dump = dbg.dump_context(stack_depth=4, print_dots=False)

        print context_dump

        if pydbg.exception_address == snapshot_hook:
                hit_count += 1
                print "hit the snapshot address"
                start = time.time()
                print "taking snapshot..."
                pydbg.process_snapshot()
                end = time.time() - start
                print "snapshot took: %.03f seconds\n" % end
                if hit_count >= 1:
                        if address:
                                print "freeing last chunk"
                                print "%08x" % address
                                pydbg.virtual_free(address, 1000, MEM_DECOMMIT)
                print "allocating memory for mutated data"
                address = pydbg.virtual_alloc( None, 1000, MEM_COMMIT, PAGE_READWRITE)
                print "Allocated 1000 bytes at: %08x" % address

        return DBG_CONTINUE

def handle_av (pydbg, dbg, context):
```

```
        '''
        As we are mucking around with process state and calling potentially unknown subroutines, it is
likely that we may cause an access violation. We register this handler to provide some useful
information about the cause.
        '''

        crash_bin = utils.crash_binning.crash_binning()
        crash_bin.record_crash(dbg)

        print crash_bin.crash_synopsis()
        dbg.terminate_process()

if __name__ == '__main__':
        dbg = pydbg()
        dbg.set_callback(EXCEPTION_BREAKPOINT,handle_bp)
        dbg.set_callback(EXCEPTION_ACCESS_VIOLATION, handle_av)

        found_target = False

        for (pid, proc_name) in dbg.enumerate_processes():
                #print proc_name.lower()
                if proc_name.lower() == "nateonmain.exe":
                        found_target = True
                        print "[+] Found Target:\"%s" %proc_name.lower()
                        break

        if found_target:
                dbg.attach(pid)
                print "[+] Attached to :" + str(pid)
                dbg.bp_set(snapshot_hook)
                dbg.bp_set(restore_hook)
                print "[+] Hooks set, entering debug loop..."
                dbg.debug_event_loop()
        else:
                print "Target not found\n"
```

예제 12: PyDbg

(3) 보안 연구자들은 2007년에 드라이버의 많은 취약점들을 발견했다. 그 중 하나가 IOCTL 드라이버

핸들러의 취약점이다. 이 문제는 디바이스 드라이버의 IOCTL 핸들러 내부에 충분하지 못한 주소 공간 검사에 의해 발생한다. 이러한 취약점들을 퍼징을 통해서 점검할 수 있다. 현재까지 ioctlizer, Win32 IOCTL fuzzer 들이 공개되었다.

```
HANDLE WINAPI CreateFileW(
    LPCTSTRIpFileName,
    DWORD dwDesiredAccess, // GENERIC_READ(0x80000000) or GENERIC_WRITE(0x40000000)
    DWORD dwShareMode, // Set to 0
    LPSECURITY_ATTRIBUTES lpSecurityAttributes, // Set to NULL
    DWORD dwCreationDisposition, // OPEN_EXISTING(0x3)
    DWORD dwFlagAndAttrinutes, // Set to 0
    HANDLE hTemplateFile // Set to NULL
);
```

예제 13: CreateFileW 와 입력될 파라미터들

```
BOOL WINAPI DeviceIoControl (
    HANDLE hDevice, // CreateFileW 에서 구한 Handle
    DWORD dwIoControlCode, // IOCTL code
    LPVOID lpInBuffer, // buffer
    DWORD nInBufferSize,
    LPVOID lpOutBuffer,
    DWORD nOutBufferSize,
    LPDWORD lpByteReturned,
    LPOVERLAPPED lpOverlapped // Set to NULL
);
```

예제 14: DeviceIoControl 과 입력될 파라미터들

CreateFileW 를 이용하여 kernel32.dll의 핸들을 구하고, DeviceIoControl 를 이용하여 핸들을 IOCTL 아래로 보낸다. 해당 작업을 수행할 디바이스에게 Device Driver 제어 코드를 직접 전송한다.

6. 결론

6.1. 연구결과

퍼징은 두 가지 형태로 수행될 수 있다. 첫 째는 무작위 랜덤으로 값을 애플리케이션에 입력하는 방법이다. 가장 확실한 방법이지만, 그 만큼 비효율적인 방법이기도 하다. 두 번째는 포맷에 맞는 데이터를 부분씩 수정하여 애플리케이션에 입력하는 방법이다. 이 방법은 앞의 방법보다 효율적이지만, 놓칠 수 있는 패턴이 존재할 수 있고, 우선적으로 포맷에 대한 분석이 수행되어야 한다.

애플리케이션에 에러가 발생할만한 데이터가 입력되면 데이터 처리 중에 에러가 발생한다. 애플리케이션은 시스템에 에러메시지를 보낼 것이고, Fuzzer는 이 에러메시지를 얻어서 퍼징의 성공 여부를 판단한다. 퍼징의 자세한 결과를 확인하기 위해서는 시스템의 메모리를 덤핑하여서 분석해볼 수 있다. 만약 애플리케이션이 서버와 클라이언트로 구성된다면, 클라이언트 측에 Fuzzer를, 서버 측에 Analyser를 두어서 퍼징을 수행하는 환경을 구축할 수도 있을 것이다.

6.2. 기대 효과 및 의견

서론에서 밝혔듯이 국내에서의 퍼징의 활용도는 저조한 편이었다. 그래서인지 퍼징에 대한 자료가 국내에 너무 부족하였다. 대부분의 자료들은 영문 문서이고, 공개된 퍼징툴들 역시 외국에서 개발된 것들이 대부분이었다. 그래서 이 문서가 퍼징(Fuzzing)에 처음 입문하는 이들에게 자그마한 도움이 되기를 바란다.

그리고 원래 우리 동아리가 퍼징을 주제로 프로젝트를 시작한 것이 아니었다. 처음에는 “CUDA를 이용한 병렬화와 암호공격”이 주제였다. 그러나 CUDA를 이용하기 위한 환경을 구축할 수 없어서 난관에 봉착하였고 KUCIS 측에 지원이 가능한가에 대하여 문의를 하였지만, 하드웨어에 대한 지원이 어려운 형편이었다. 그래서 늦게나마 “취약점을 찾기 위한 fuzzing 기술의 구현”이라는 주제로 변경하게 되었다. 앞으로는 하드웨어에 대한 지원이 확충되어서 이후 다른 동아리에서도 작업환경 때문에 프로젝트 주제가 변경되는 일이 없어졌으면 한다.

7. Reference

- Windows Internals 5th Edithon - *Russinovich, Solomon*
- Gray hat Python -*Justin Seitz*
- Fuzzing, Brute Force Vulnerability Discovery - *Michael Sutton, Adam Greene, Pedram Amini*
- Attacking Antivirus - *Feng Xue, 번역 Vangelis*
- 열혈강의 웹 헵킹과 방어 - *최경철*
- Taint analysis for vulnerability discovery - *passket*
- Fuzzing for software vulnerability discovery - *Toby Clarke*
- Building Secure Software using Fuzzing and Static Code Analysis - *Anna-Maija Juuso*
- In Memory Fuzzing :: http://thatsbroken.com/?p=282 - *jRichards*
- Automated Penetration Testing with White-Box Fuzzing :: http://msdn.microsoft.com/en-us/library/cc162782.aspx - *John Neystadt*
- MSDN Library :: http://msdn.microsoft.com/en-us/library