Tutorial - Beginner's Guide to Fuzzing

Part 1 : Simple Fuzzing with zzuf

원문 번역 - 6. 27, 2018 willwayy 이정모

이 튜토리얼의 목표는 퍼징이 정말 쉽다는걸 알려주기 위함이다. 많은 무료 소프트웨어들은 오늘날 퍼징을 통해 쉽게 해당 프로그램에 대한 버그를 찾을 수 있다는 점에 골머리를 앓고 있다. 이것은 바뀌어야하고 우리는 대부분의 프로젝트 개발과정에 퍼징을 해야한다고 생각한다. 퍼징은 Target 응용프로그램의 Input에 잘못된 형식이 들어가거나 개발자가 예상치 못한 행동을 해서 원하지 않는 동작을 한다 (e.g. Crashes). 우리는 일반적으로 유효한 입력을 받아 무작위로 오류를 추가한다.

좋은 fuzzing 툴은 다양한 포맷의 파일 형식에 대한 퍼서를 제공한다. `ImageMagick` 을 예로 들어 보자. 이것은 수많은 파일 형식으로 이미지를 처리하는 명령어 기반 도구다.

일단 우리는 어떻게 이걸 퍼징할건가?? 우리는 일단 먼저 입력 샘플을 생성해야한다. 일반적으로 작은 파일을 사용하는 것이 좋다. 먼저 크기가 작은 여러 형식의 간단한 이미지를 만들자 (e.g. 3x3 pixel PNG). 우리는 이 파일들을 example.png 라고 할 것이다. 자 이제 우리는 이 PNG 파일을 다양한 포맷의 파일로 변환하는 작업을 할 것이다. 이 경우 `ImageMagick` 를 사용하거나 더 정확한 `ImageMagick` 의 일부인 변환도구를 사용하여 예제 파일을 만들 수 있다.

```
convert example.png example.gif
convert example.png example.xwd
convert example.png example.tga
```

위에 나와있는 것 말고도 원하는만큼 많이 만들어보자 (*convert -list format* 이 명령어는 지원되는 모든 포맷을 보여준다). 이제 우리는 예제 파일의 형식이 잘못된 버전이 필요하다. 여기서 우리는 zzuf 툴을 사용할 것이다. 간단한 fuzzing 도구이며 대부분의 Linux 배포판에서 사용할 수 있다.

zzuf 를 사용하는 방법은 여러가지가 있는데 예를 들어 example.* 입력파일에서 많은 수의 잘못된 파일을 만드는데 사용할 수 있다. 간단한 bash loop를 사용하면 더 쉽게 만들 수 있다 :)

```
for i in {1000..3000}; do for f in example.*; do zzuf -r 0.01 -s $i < "$f" > "$i-$f";
done; done
```

> -r 옵션은 파일에서 원하는 변경사항의 양을 의미한다.
>
> 여기서 0.01은 파일의 1 %가 무작위로 변경됨을 의미한다.

> -s 옵션은 seed 이다.

모든 예제 파일에 대해 변형된 *[number]-example.[extension]* 로 지어진 파일 2000개를 생성했고, 모든 다른 **s** 값에 대해 우리는 다른 결과를 얻는다. 물론 여러가지 숫자를 적용할 수 있지만 경험상 2000 이 가장 합리적인 숫자였습니다.

다음 단계 : 우리는 `ImageMagick` 에 잘못된 형식의 파일을 제공하고 싶다! 그러기 위해서 우리는 파일을 가지고 어떠한 일을 하는 명령어가 필요하다. 또한 우리는 *convert* 도 사용할 수 있다. 실제로 버그 발생 확률을 높이기 위해 *convert* 에 변형된 이미지의 크기를 조정하도록 하자. 그리고 output 을 나중에 검사 할 수 있는 로그파일로 리다이텍트한다.

```
LC_ALL=C; LANG=C; for f in *-example.*; do timeout 3 convert -resize 2 "$f"
/tmp/test.png; echo $f; done &> fuzzing.log
```

> *LC_ALL=C; LANG=C;* 는 확실하게 output 이 영어로 설정되게끔 하기 위함이다 (왜냐하면 error 메시지를 grep 위함).
>
> *timeout* 명령어는 단일 파일이 너무 오래 걸릴 때 중지하도록 하기 위함이다 (왜냐하면 무한루프 버그를 놓칠 수 있다).
>
> 해결방법을 제시할 수 있지만 여기는 튜토리얼이므로 생략하겠다. 또한 변환 할 때마다 현재 파일의 이름을 출력한다.
>
> 이것을 분명히 해야한다 - 나중에 어떤 파일을 찾으면 crash가 났는지를.

이것은 꽤 오랜 시간이 걸릴 수 있으며 출력 파일을 얼마나 많이 사용하는지에 따라 출력 파일이 상당히 커질 수 있기 때문에 몇 기가 바이트 공간이 있어야 한다.

이제 우리가 무언가를 찾았는지 확인해야한다. 우리는 로그파일에서 **"Segmentation faults"** 부분을 살펴볼 것이다.

```
grep -C2 "Segmentation fault" fuzzing.log
```

> 만약 어떠한 crashes 를 발견하면 그걸 봐야하며 또한 crash 를 일으킨 파일의 파일 이름을 확인해야한다.

Using zzuf directly

앞서 해본 내용은 입력 파일 형식이 많은 도구에 유용하지만 가장 효율적인 방법은 아닙니다. zzuf 자체적으로 퍼징하고 싶은 도구를 실행할 수 있다. zzuf는 작업을 병렬 처리 할 수 있으며 테스트를 마친 소프트웨어의 응답을 감지할 수도 있다.

아까 해본 ImageMagick 을 예로 들자면 아까와 다른 방식으로 할 수 있다. objdump 를 사용해볼건데, 이 프로그램은 디버깅을 하는 도구이다. objdump 를 사용하기 위해서는 인자값으로 실행파일이 필요하다. 우리는 windows EXE 파일을 사용할 것이며 trivial EXE from our file formats archive 여기서 얻을 수 있다. 자 그럼 zzuf 위에서 objdump 와 우리가 구한 실행파일을 실행시켜보자.

```
zzuf -s 0:1000000 -c -C 0 -q -T 3 objdump -x win9x.exe
```

> -s 옵션은 백만 개의 시드 값을 시도 할 것을 의미합니다.
>
> -c 옵션은 zzuf 가 명령 행에서 주어진 파일만을 퍼징해야 함을 의미한다. 그렇지 않으면 도구가 이미 설정 파일이나 기타 다른 것들로부터 오류 메시지를 던질 것이기 때문에 유용하다. 그래서 그것들은 우리의 Fuzz input 을 갖지 않을 것이다.
>
> -C 0 옵션은 zzuf 가 첫 번째 crash가 발견 된 후에 멈추지 않아야 함을 의미한다.
>
> -q 옵션은 Fuzz 명령의 출력이 나오지 않게 한다.
>
> -T 3 옵션은 제한 시간을 3 초로 설정하여 zzuf 가 무한 루프로 빠지면 멈추지 않도록 한다.

다음과 같은 출력이 표시된다 :

```
zzuf[s=215,r=0.004]: signal 11 (SIGSEGV)
```

> 매개 변수 -s 215와 -r 0.004를 가진 segfault를 발견하는 것을 의미한다.

이제 이 잘못된 형식의 파일을 다시 만든다.

```
zzuf -r 0.004 -s 215 < win9x.exe > crash.exe
```

Analyzing and Reporting

이제 응용 프로그램에서 crash를 일으키는 Fuzz 파일이 생겼다. 이것들을 어플리케이션 작성자에게 보낼 수 있다!

참고 : 여기에 언급 된 예제들은 이미 보고가 된 상태이며 작성자(원문)는 이를 수정하는 중이다.

우리는 또한 그것들에 대해 좀 더 분석 할 수 있다. 편리한 도구 중 valgrind 가 있는데 *valgrind -q* 을 실행시켜 분석해보자.

```
valgrind -q objdump -x crash.exe
```

> -q 옵션은 불필요한 출력을 억제한다.

그러면 이런식의 출력을 볼 수 있다 :

```
==22449== Process terminating with default action of signal 11 (SIGSEGV)
==22449==  Access not within mapped region at address 0x7715FF3
==22449==    at 0x4E7FAC0: bfd_getl16 (libbfd.c:570)
==22449==    by 0x4EE356D: pe_print_idata (peigen.c:1328)
==22449==    by 0x4EE356D: _bfd_pe_print_private_bfd_data_common (peigen.c:2160)
==22449==    by 0x4EDE1F8: pe_print_private_bfd_data (peicode.h:335)
==22449==    by 0x408504: dump_bfd_private_header (objdump.c:2643)
==22449==    by 0x408504: dump_bfd (objdump.c:3214)
==22449==    by 0x408AA7: display_object_bfd (objdump.c:3313)
==22449==    by 0x408AA7: display_any_bfd (objdump.c:3387)
==22449==    by 0x40AB22: display_file (objdump.c:3408)
==22449==    by 0x405249: main (objdump.c:3690)
```

참고 * - 이러한 문제를 업스트림 개발자에게 보고 할 때 좀 더 많은 정보를 보내는 것이 좋다. Fuzzing 을 수행한 소프트웨어에 디버깅 심볼이 활성화되어 있지 않으면 출력이 상세하지가 않다.

Fuzzing 을 하기 위한 소프트웨어를 컴파일하는 경우 컴파일러 플래그에 -ggdb를 포함시켜 더 많은 디버깅 정보를 얻을 수 있다. Fuzzing 을 위해 소프트웨어를 컴파일하는 경우 가능한 경우 공유 라이브러리를 비활성화하는 것이 좋다. 그러면 퍼징된 소프트웨어가 대신 시스템 라이브러리를 사용하는 문제가 발생하지 않는다. configure 스크립트 가있는 소프트웨어의 경우 다음과 같이 작동한다.

```
CFLAGS="-ggdb" CXXFLAGS="-ggdb" ./configure --disable-shared
```

이것은 Fuzzing Tutorial Part 1 이다. 또한 여기서 나온 zzuf official zzuf Tutorial 도 참고해보면 좋을 것 같다.

Part 2 에서는 Address Sanitizer 를 이용해 버그를 찾아볼것이며

Part 3 에서는 더 똑똑한 Fuzzing 도구인 fuzzy lop 를 소개할 것이다.

원문 번역 - 6. 27, 2018 willwayy 이정모