HAEPG: 자동 다중 홉 익스플로잇 생성 프레임워크

Zixuan Zhao1,2,3,4, Yan Wang1,2,3,4 및 Xiaorui Gong1,2,3,4(B)

[1] 중국과학원 사이버보안대학, 중국 베이징 네트워
[2] 크 평가 기술 핵심 연구소 중국 베이징 네트워크 보안 및 보호 기술 베이징 핵심 연구소
삼
베이징, 중국 정보
[4] 공학 연구소, 중국 과학 아카데미, 베이징, 중국
{zhaozixuan,wangyan,gongxiaorui}@iie.ac.cn

추상적인. 힙 취약점에 대한 자동 익스플로잇 생성은
오픈 챌린지. 현재 연구에는 힙에 대한 민감한 포인터가 필요합니다.
제어 흐름을 가로채고 기능이 제한된 취약점에는 거의 주의를 기울이지 않습니다. 본 논문에서는
자동 익스플로잇인 HAEPG를 제안한다.
악용을 안내하기 위해 알려진 악용 기술을 활용할 수 있는 프레임워크
세대. 심볼릭 기반의 HAEPG 프로토타입을 구현했습니다.
실행 엔진 S2E [15] 를 위해 4가지 익스플로잇 기술을 제공했습니다.
그것은 사전 지식으로. HAEPG는 충돌하는 입력, 프로그램 및 이전
지식을 입력으로 사용하고 힙 할당기의 내부 기능을 남용하여 기능이 제한된 취약점에 대한 익스
플로잇을 생성합니다.

우리는 24개의 CTF 프로그램으로 HAEPG를 평가했으며 그 결과는
HAEPG는 21개의 취약점 유형에 대해 정확하게 추론할 수 있습니다.
(87.5%), 그리고 16(66.7%) 동안 포탄을 생성하는 익스플로잇을 생성합니다.
그들의. 모든 익스플로잇은 NX [25] 및 Full RELRO [28] 를 우회할 수 있습니다.
보안 메커니즘.

키워드: 자동 익스플로잇 생성 · 힙 취약점 ·
상징적 실행

1. 소개

AEG(Automated Exploit Generation)는 다음 분야에서 중요한 방법이 되고 있습니다.
취약성 중심 공격 및 방어. 소프트웨어 공급업체는 이를 사용하여 평가합니다.
보안 취약점의 심각도를 보다 신속하게 파악하고 적절한 할당
중요한 취약점을 수정하기 위한 리소스. 방어자는 합성 익스플로잇에서 배운다
침입 탐지 시스템 규칙을 생성하고 잠재적인 공격을 차단합니다.

대부분의 AEG 솔루션 [12,13,20,23,26] 은 일반적으로 스택 관련 또는
현대 시스템에서는 드문 형식 문자열 취약점 [2]. 인해
힙 할당기 기능의 복잡성으로 인해 기존 솔루션 중 몇 개만 생성할 수 있습니다.
힙 기반 취약점에 대한 익스플로잇. 이러한 솔루션에는 다양한 접근 방식이 있습니다.

C. Maurice et al. (Eds.): DIMVA 2020, LNCS 12223, pp. 89–109, 2020.
https://doi.org/10.1007/978-3-030-52683-2_5

예를 들어 Revery [30] 는 취약 지점에서 파생된 경로에서 악용 가능한 상태를 탐색하기 위해 레이아웃 지향 퍼징 및 제어 흐름 스티칭 솔루션을 적용합니다. Gollum [22] 은 사용자 정의 힙 할당자를 사용하여 악용 가능한 힙 레이아웃을 생성 하고 힙 조작 문제를 해결하기 위해 선행 작업 [21] 을 기반으로 하는 퍼징 기술을 사용 합니다. SLAKE [14] 는 유용한 커널 객체를 검색하기 위해 정적-동적 하이브리드 분석을 사용하고 슬래브에서 자유 목록을 조정하여 힙 레이아웃을 조작합니다.

이러한 모든 솔루션은 민감한 포인터(예: VTable 포인터)를 손상시키고 공격자가 제어하는 메모리 쓰기 또는 간접 호출을 유도합니다. 이는 민감한 포인터의 존재가 제어 흐름을 가로채는 열쇠임을 의미합니다. 이 경우 힙 레이아웃이 잘 정렬되면 공격자는 취약점을 유발하는 단 하나의 작업으로 익스플로잇 프리미티브를 생성하고 이를 단일 홉 익스플로잇이라고 합니다. 그러나 간단한 단일 홉 기술을 사용하여 모든 취약점을 악용할 수 있는 것은 아닙니다. 예를 들어, off-by-one 오류 [11] 의 경우 명령 포인터를 임의의 값으로 덮어쓰는 것은 고사하고 취약점을 유발하는 것만으로 민감한 포인터를 완전히 제어하는 것은 불가능합니다. 이 문제를 해결하려면 다음과 같은 과제를 해결해야 합니다.

과제 1: 제한된 기능으로 힙 취약점의 위력 탐색. 제한된 기능으로 취약점을 악용하려면 공격자가 힙 레이아웃을 조작하고 힙 할당자의 내부 기능을 남용하여 여러 중간 홉을 만들고 홉의 도움으로 손상될 수 있는 메모리 범위를 확장하고 결국 임의의 메모리 쓰기 또는 간접 호출. 우리는 이러한 기술을 다중 홉 착취라고 부릅니다. 일부 솔루션 [19,32] 은 힙 할당자를 위한 이러한 기술을 발견하는 것을 목표로 합니다. 그러나 힙 기반 취약점이 있는 프로그램에 이 기술을 자동으로 적용할 수는 없습니다. 우리가 아는 한, 기존 AEG 솔루션은 이에 대해 거의 관심을 기울이지 않았습니다.

과제 2: 프로그램과 힙 할당자 간의 힙 상호 작용 모델링 다중 홉 악용을 수행하기 위해 AEG 솔루션은 특정 크기의 개체를 할당 및 할당 해제하거나 힙 개체에 특정 데이터를 쓰기 위해 입력을 작성하고 희생자 프로그램을 구동해야 합니다. 그러나 프로그램은 일반적으로 사용자가 힙 할당자와 상호 작용할 수 있는 직접적인 인터페이스를 노출하지 않습니다. 따라서 AEG 솔루션은 힙 상호 작용을 인식하고 익스플로잇을 생성하기 위해 특정 방식으로 조합해야 합니다.

우리의 솔루션. 본 논문에서는 위의 과제를 해결하기 위해 HAEPG를 제안한다.
힙 기반 취약성과 충돌 입력이 있는 프로그램이 주어지면 다중 홉 공격을 통해 임의 코드 실행을 시도합니다.

HAEPG는 힙 상호 작용으로 힙 할당자와 상호 작용하는 기계 수준 명령 및 함수 호출을 추상화합니다. 대부분의 프로그램이 기능 디스패처(예: 이벤트 처리 및 연결 처리 루프)를 사용하여 기능을 배포하고 이러한 디스패처를 구성하는 경로를 추출한다는 사실에 의존합니다.

그런 다음 HAEPG는 하이브리드 기술을 적용하여 힙 상호 작용을 찾고 분석하고 서로 다른 상호 작용과 경로 간의 종속성을 추론합니다.

그 후, HAEPG는 충돌 입력을 실행할 때 프로그램에서 런타임 정보를 수집합니다. 취약한 개체를 검사하고 메모리 손상 유형 및 손상된 데이터 크기를 분석합니다.

또한 힙 취약점에 대한 수동 다중 홉 악용 기술을 연구했습니다. 이러한 기술은 일반적으로 힙 할당자의 내부 기능을 남용하고 신중하게 제작된 힙 상호 작용 시퀀스를 통해 취약점의 기능을 향상시킵니다. 우리는 알려진 멀티홉 익스플로잇 기술을 익스플로잇 템플릿으로 추상화하는 템플릿 언어를 설계했습니다. HAEPG는 이를 사용하여 임의 실행을 달성하고 종단 간 익스플로잇을 생성합니다.

기호 실행 엔진 S2E [15] 를 기반으로 HAEPG의 프로토타입을 구축하고 glibc의 표준 할당 자인 ptmalloc [4]의 4가지 활용 기술에 대한 템플릿을 작성하고 잘 알려진 Capture The Flag(CTF)의 24개 프로그램에서 이를 평가했습니다. 대회. 결과는 HAEPG가 그 중 21개(87.5%)에 대해 취약점 유형을 정확하게 추론할 수 있고 그 중 16개(66.7%)에 대해 셸을 생성하는 익스플로잇을 생성할 수 있음을 보여줍니다.

2 동기 부여 예

이 섹션에서는 다중 홉 악용을 설명하고 AEG 솔루션이 이 예를 처리할 때 발생하는 문제를 보여주기 위한 예를 제공합니다.

취약점. 예제는 glibc의 수정되지 않은 버전이 있는 GNU/Linux 시스템에서 실행됩니다. 그림 1에서 볼 수 있듯이 프로그램에는 addItem, removeItem, editItem의 세 가지 기능이 있으며 이 기능은 객체를 할당하고 해제하는 데 사용됩니다. 개체를 수정하고 개체를 수정합니다. 22행 에 포이즌 널 바이트 오류 [16] 가 있지만 힙 객체 간의 메타 데이터만 손상시키는 반면 힙 객체의 내용은 변경되지 않습니다.

멀티 홉 익스플로잇. 그림 1 의 예 는 안전하지 않은 링크 해제 기술 [9] 을 통한 악용을 보여줍니다 . 먼저 세 개의 힙 개체 A, B, C를 할당합니다.
BSS에는 프로그램이 B객체에 접근하기 위해 사용한 포인터가 저장되어 있다. 그런 다음 상태 3과 같이 개체 A의 취약점을 트리거하여 개체 B의 크기를 줄이고 상태 4의 개체 A에 가짜 청크를 위조합니다. 상태 5에서 객체 B를 해제한 후 상태 6에서 프리미티브. 마지막으로 임의의 쓰기 프리미티브로 함수 포인터를 손상시키고 제어 흐름을 하이재킹합니다.

이러한 상태는 다음과 같이 분류할 수 있습니다.

- 초기 상태: 프로그램이 실행을 시작할 때의 상태, 예를 들어 상태 1.
- 준비 상태: 프로그램이 메모리 레이아웃을 조작할 때의 상태
  손상이 발생하기 전에 악용하기 위한 것입니다(예: 상태 2).
- 손상 상태: 취약성을 트리거할 때의 상태(예: 상태 3).
- 중간 상태: 프로그램이 초기 상태(예: 상태 4-5)에서 악용 가능한 상태에 도달하기 위해 거쳐야 하는 상태입니다.
- 익스플로잇 가능한 상태: 익스플로잇을 위한 익스플로잇 프리미티브가 있는 상태, 예:
  상태 6과 7.

```
1. addItem() 무효화{
2. 정수 크기 = read_int();
3. size_list[인덱스] = 크기;
4. heap_list[인덱스] = malloc(크기);
5. if(!힙_목록[인덱스])
6.         puts("malloc 오류");
7. 반환
8. }
9. removeItem(){
10. if(힙_목록[인덱스]){
11. 무료(힙_목록[인덱스]);
12.         힙 목록[인덱스] = 0;
13.     }
14. if(크기 목록[인덱스])
15. size_list[인덱스] = -1;
16. }
17. editItem() { 무효화
18. for(int i = 0; i < size_list[index]; i++){
19. read(0, heap_list[인덱스] + i, 1);
20. if(!힙_목록[인덱스][i])
21. 휴식
22. }
heap_list[index][size_list[index]] = NULL;
24. }
25. 무효 메인(){
26. 동안(참){
27. 인덱스 = readInst();
28. 선택 = readInst();
29. 스위치(선택){
30. 사례 1: addItem(); 부서지다;
31. 사례 2: removeItem(); 부서지다;
32. 사례 3: editItem(); 부서지다;
33.         }
34.     }
35. }
```

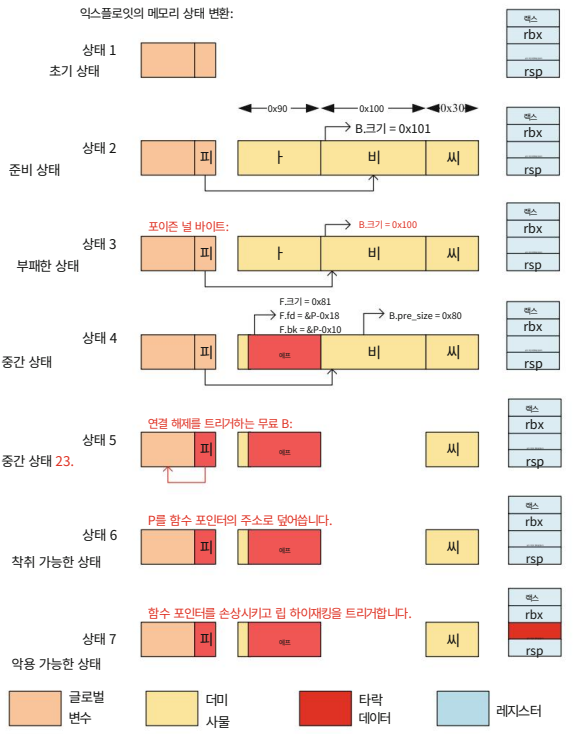

그림 1. 포이즌 널 바이트의 예

현대의 퍼징 도구 [10,33] 에서 충돌 입력을 생성하는 것은 쉽습니다.
취약점. 그러나 대부분의 AEG 솔루션은 이를 처리할 수 없었습니다.
프로그램 충돌 시 직접적인 익스플로잇 프리미티브가 없기 때문입니다. 을 위한
예를 들어 Shellphish가 개발하고 DARPA CGC [17] 에서 3위를 차지한 자동 공격 키트 프레임워크인
Mechanical Phish [26] 는
기계식 피싱 때문에 취약점이 발생하고 예제에 대한 악용이 생성되지 않습니다.
셸 코드 또는 rop-chains를 주입하기 위해 제어 가능한 포인터가 필요합니다. 해결책
Revery [30] 는 손상된 상태 3을 찾을 수 있습니다. 그러나 다음을 수행할 수 있는 기능이 없습니다.
힙 할당자의 온전성 검사를 우회하고 취약점을 강화합니다.
따라서 취약점을 악용으로 전환할 수 없습니다.

3 방법론

그림 2 는 HAEPG의 개요를 보여줍니다. 프로그램과 충돌 입력을 다음과 같이 취합니다.
입력 및 악용 안내를 위한 템플릿. HAEPG 첫 번째 모델 힙
기능 경로 및 힙 기본 요소와 대상 프로그램의 상호 작용. 그것
정적 분석을 사용하여 프로그램에서 기능 경로를 추출하고 동적으로

상호 작용하는 함수 경로의 명령 및 함수 호출을 추적합니다.
런타임 동안 힙 할당자를 사용하고 다양한 힙 상호작용의 관계를 기록합니다.

그런 다음 HAEPG는 충돌 입력으로 프로그램을 실행하여 손상 유형 및 규모를 포함하여 취약점에 대한 정보를 검색합니다.
데이터. 우리는 널리 사용되는 착취 기술을 템플릿화하기 위한 템플릿 언어를 설계했습니다. 각 템플릿에는 안내에 필요한 정보가 포함되어 있습니다.
착취. HAEPG는 다음을 확인하여 적용 가능한 익스플로잇 템플릿을 필터링합니다.
힙 할당자와 프로그램이 템플릿의 요구 사항을 충족합니다. 그것은 시도
공격 시퀀스를 생성하고 주기적으로 시퀀스를 평가하고 수정합니다.
프로그램이 악용 가능한 상태에 도달하거나 분석이 생성 시도에 대해 구성 가능한 상한을 초과할 때까지(예: 20회).

마지막으로 HAEPG는 생성된 익스플로잇 프리미티브를 사용하여 명령어를 손상시킵니다.
제어 흐름을 하나의 가젯으로 전송하기 위한 포인터 [18]. 원 가젯은 코드입니다.
인수 없이 "/bin/sh"를 호출하는 glibc 내부의 조각, 효과적으로
공격자를 위한 쉘 생성. HAEPG가 쉘 프로세스가 생성되었음을 감지하면
대상 프로그램에서 다음과 같은 경우 수집된 경로 및 데이터 제약 조건을 해결합니다.
공격 시퀀스를 실행하고 익스플로잇 입력을 생성합니다.

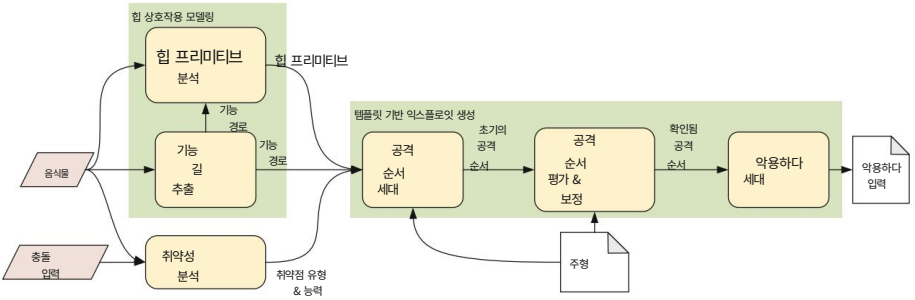

그림 2. HAEPG 개요

3.1 힙 상호작용 모델링

함수 경로 추출. 함수 디스패처는 다음과 같은 코드 구조입니다.
함수 분포를 위한 프로그램에서 널리 사용되며 일반적으로 다음과 같이 구현됩니다.
루프에 싸인 if-else 또는 switch-case 구조. 프로그램은 주기적으로 수신
루프 항목에서 명령을 실행하고 해당 기능을 실행합니다. 우리는 정의
루프 항목에서 루프 종료까지의 기본 블록 시퀀스는 함수 경로로,
다음과 같은 속성이 있습니다.

- 원자성: 함수 경로는 런타임에 분할할 수 없습니다. 프로그램은 할 수 없습니다
  함수 경로의 절반을 실행한 다음 다른 함수 경로로 점프합니다. 그만큼
  함수 디스패처의 구조는 각 함수 경로가
  다른 사람을 실행하기 전에 완전히 실행되어야 합니다.

– 재진입: 기능 경로의 입구와 출구가 동일합니다. 함수 경로의 출구에 도달하면 프로그램은 입구로 돌아가 명령에 따라 다음 함수 경로를 선택합니다. 따라서 프로그램은 적절한 명령을 통해 기능 경로를 여러 번 실행할 수 있습니다.

함수 경로를 추출하기 위해 HAEPG는 먼저 대상 프로그램의 제어 흐름 그래프를 생성하고 힙 할당 또는 할당 해제의 함수 호출을 포함하는 루프를 후보 함수 디스패처로 표시합니다. 함수 디스패처의 루프 본문이 switch-case 문인 경우 HAEPG는 여러 후속 항목이 있는 기본 블록을 검색하고 각 후속 항목의 경로를 함수 경로로 추출합니다.

그렇지 않으면 루프 본문이 중첩된 if-else 문의 시퀀스가 될 수 있습니다. HAEPG는 루프 본문의 끝에서 시작하여 루프를 역방향으로 순회하고 순회하는 동안 각 기본 블록의 선례 번호를 확인합니다. 많은 수(예: 5개 이상)가 있는 첫 번째 기본 블록은 모든 디스패치된 함수의 병합 지점으로 표시되며 루프 항목에서 이 블록으로의 모든 경로를 함수 경로로 추출합니다.

힙 기본 분석. 힙 프리미티브는 프로그램과 힙 할당자 간의 상호 작용을 모델링합니다. 다음 세 가지 유형과 해당 속성을 구분합니다.[1]

배당	할당 해제	편집하다
크기: 할당 크기	addr: 주소	base: 편집 주소의 베이스
addr: 반환된 주소	출시된	offset: 편집 주소 데이터의 오프셋: 쓸 데이터
힙 할당자에 의해		

일반적으로 힙 프리미티브는 다음과 같은 프로그램 기계 명령어입니다. (1) malloc, calloc, free 등과 같은 힙 할당자와 상호 작용하는 함수 호출 (2) read, fgets와 같은 인수로 힙 포인터를 사용하는 함수 호출 등 (3) 힙 개체를 대상 주소로 사용하는 메모리 쓰기 명령.

힙 프리미티브를 분석하기 위해 HAEPG는 기호 실행을 사용하여 함수 경로를 실행합니다. 모든 입력 바이트를 상징하고 힙 영역과 상호 작용하는 API 호출 및 명령어를 추적합니다. HAEPG는 대상 프로그램의 API 호출과 명령만 힙 기본 형식으로 기록하고 공유 라이브러리를 무시합니다. 대상 프로그램과 공유 라이브러리를 모두 추적하면 성능 오버헤드가 증가하기 때문입니다. 힙 프리미티브의 속성이 상징적이면 HAEPG는 구체화하지 않고 범위를 풀고 기록합니다.

힙 프리미티브 간의 관계를 추론하기 위해 HAEPG는 할당에 의해 반환된 각 힙 포인터에 대해 전역적으로 고유한 taint 태그를 연결하고 tainted 포인터에서 작동하는 프리미티브를 식별합니다.

3.2 취약점 분석

AFL [33] 로 충돌 입력을 얻고 메모리 사용 위반을 감지하여 취약점의 기능을 분석합니다. HAEPG는 동적으로 실행

[1] 우리는 기본 상호 작용 유형만 모델링합니다. 힙 할당자는 이 문서의 범위를 벗어나는 다른 유형의 상호 작용(예: realloc)을 가질 수 있습니다.

충돌하는 입력 및 오염은 힙과 유사하게 할당 포인터를 반환합니다.
원시 분석. 또한 태그를 지정된 메모리로 전파합니다.
객체의 바이트 힙 객체의 상태에 대한 주석을 추가합니다.
태그: 처음에는 태그가 초기화되지 않고 개체에 쓰는 명령
객체를 해제하는 명령이 변경되는 동안 상태를 사용 중으로 변경합니다.
무료로 합니다. 또한 HAEPG는 해당 객체의 크기를 기록합니다.
태그. 명령이 힙 메모리에 액세스하면 포인터의 태그를 얻을 수 있습니다.
ptag 및 지적된 객체의 메모리 태그 mtag, 상태 및
그들의 크기. 상태를 변경하기 전에 HAEPG는
표 1 에 표시된 위반 규칙 이 트리거됩니다.

표 1. 취약점 유형, 트리거 옵션 및 위반 규칙 목록

취약점 유형 트리거 작업		위반 규칙
더블 무료	힙 청크 해제	mtag.status == 무료 또는 ptag.status == 무료
무료 후 사용	n 바이트의 데이터 저장 메모리 주소 [base + off] (베이스와 오프는 베이스 및 주소 오프셋)	mtag.status == 무료 또는 ptag.status == 무료
과다		n + 끄기 >ptag.size
포이즌 널 바이트		n + 끄기 == ptag.size + 1 데이터의 마지막 바이트는 널 바이트
오프 바이 원		n + 끄기 == ptag.size + 1

위반 규칙이 트리거되면 HAEPG는 손상된 데이터의 규모를 기록하고,
오버플로된 바이트의 범위 또는 취약한 청크의 크기와 같은.

3.3 템플릿 기반 익스플로잇 생성

이 섹션에서는 익스플로잇 생성 방법을 설명합니다. 우리는 가져
메모리 구성에 대한 사전 지식으로서의 기존 활용 기술
상태 및 악용 가능한 상태에 도달하여 HAEPG에 도달합니다. 또한 악용 기술을 HAEPG에 하드 코딩하는 대신 템플릿 언어를 개발했습니다.
착취 기술을 설명합니다. 템플릿을 동적으로 해석하고 공격 시퀀스를 구성하는 방법은 유연성과 확장성을 제공합니다.
HAEPG로.

템플릿. 악용 기술을 적용하는 절차는 프로그램에 민감합니다. 대상 프로그램이 약간만 변경되더라도 막대한 비용이 필요하기 때문입니다.
다양한 착취 전략. 따라서 우리는 착취 기술의 지속적이고 필수적인 구성 요소를 수집하고 템플릿으로 추상화하여 다음을 제공합니다.
다음 구성 요소에 대한 정보:

– Backbone Primitives Sequence: 취약점을 유발하고 힙 할당자의 내부 기능을 남용하는 데 사용되는 힙 프리미티브의 순서는 일정하게 유지됩니다. 예를 들어, fastbin 공격에서 주기적으로 힙 청크를 해제하여 이중 자유 취약성에서 임의 할당을 얻습니다 [5]. 이러한 힙 프리미티브를 백본 프리미티브 시퀀스라고 합니다.

– 레이아웃 제약: 다중 홉 활용의 중간 상태는 특정 제약 조건을 충족해야 할 수 있습니다. 예를 들어, unsafe unlink 공격은 희생자 청크가 unsortedbin 크기로 할당되어야 하고 가짜 청크가 희생자 청크에 인접해야 합니다. 이러한 제약을 레이아웃 제약이라고 합니다.

– 요구 사항: 익스플로잇 기술을 사용하려면 프로그램이 몇 가지 요구 사항을 충족해야 합니다. 예를 들어, 프로그램은 fastbin 공격을 위해 fastbin 크기의 개체를 할당하는 기능이 있어야 합니다.

이러한 구성 요소는 메모리 상태가 충족해야 하는 다중 홉 공격 및 제약 조건에서 악용 가능한 상태에 도달하기 위해 메모리 상태를 구성하는 방법을 나타냅니다. 마찬가지로 각 템플릿은 템플릿 사용 요구 사항, 백본 기본 요소 및 레이아웃 제약 조건을 포함하여 세 부분으로 구성됩니다. 우리는 섹션에서 익스플로잇 기술을 추상화하는 데 사용한 템플릿 언어를 소개합니다. 4.3.

공격 시퀀스 생성. 공격 시퀀스 생성은 제공된 템플릿과 밀접하게 연결되어 있습니다. 먼저 HAEPG는 대상 프로그램이 취약점 유형 및 glibc 버전과 같은 지정된 요구 사항을 충족하는지 확인합니다. 프로그램이 이러한 검사를 통과하면 HAEPG는 다음 단계를 위해 이 템플릿을 선택합니다. 그렇지 않으면 다른 템플릿을 시도합니다.

다음으로 HAEPG는 템플릿의 백본 프리미티브를 탐색합니다. 모든 기능 경로를 스캔하여 백본 프리미티브를 포함하는 경로를 찾고 이러한 기능 경로를 힙 프리미티브와 함께 공격 시퀀스로 결합합니다.

공격 시퀀스를 실행하기 위해 다음 두 가지 방법을 설계했습니다.

– 힙 시뮬레이터: 시뮬레이터는 대상 프로그램과 동일한 힙 할당자를 사용하는 독립 바이너리입니다(예: 프로토타입의 경우 ptmalloc). 우리는 그것을 사용하여 공격 시퀀스의 힙 프리미티브를 실행하고 대상 프로그램의 중간 상태를 시뮬레이션합니다.

– Symbolic Execution: 타겟 프로그램이 공격 시퀀스의 기능 경로를 실행하도록 하고 S2E의 상호 관련된 메모리 데이터 및 레지스터와 데이터 제약 조건으로 힙 프리미티브를 연결합니다(예: 할당 크기를 malloc/calloc의 첫 번째 인수의 데이터 제약 조건으로 변환). .

공격 순서 평가 및 수정. HAEPG는 템플릿의 레이아웃 제약 조건에 따라 공격 순서를 평가합니다. 대상 프로그램과 공격 순서를 동적으로 실행하고 런타임에 힙 레이아웃을 기록합니다. 그런 다음 각 필수 힙 객체의 주소, 실제 크기, 객체 포인터가 저장된 주소 등을 추출한다. 이 정보로 메모리 상태의 선행 백본 프리미티브가 끝나면,

HAEPG는 현재 힙 레이아웃이 힙 개체의 거리 및 상태를 포함하여 레이아웃 제약 조건을 충족하는지 확인합니다.

성능을 향상시키기 위해 HAEPG는 먼저 빠른 평가를 위해 힙 시뮬레이터를 사용합니다. 공격 시퀀스가 평가를 통과한 후에만 HAEPG는 정확한 평가를 위해 S2E를 사용합니다.

레이아웃 제약 조건이 충족되면 프로그램은 악용 가능한 상태에 들어가고 HAEPG는 악용을 생성하려고 시도합니다. 그렇지 않으면 HAEPG가 충돌하는 힙 레이아웃을 찾습니다. 그러면 HAEPG는 충돌의 원인을 추론하고 수정을 시도합니다. 일반적으로 충돌의 원인은 다음과 같습니다.

1. 힙 청크 A는 사용 가능해야 하지만 사용 중이거나 초기화되지 않았습니다. 2. 힙 청크 A와 B는 인접해야 하지만 다른 사용 중인 청크가 있습니다.
그들 사이에;
3. 힙 청크 A와 B는 인접해야 하지만 A의 인접 청크는 비어 있습니다.

첫 번째 경우 HAEPG는 공격 시퀀스에 경로를 삽입하여 청크 A를 해제합니다. 두 번째 경우 HAEPG는 경로를 삽입하여 청크 A와 B의 중간에 있는 모든 청크를 해제하려고 합니다. 청크를 해제할 수 없는 경우 HAEPG는 새로운 힙 레이아웃을 얻기 위해 할당하기 전에 청크 A와 동일한 크기의 힙 청크를 할당하면 만족스러운 힙 레이아웃으로 이어질 수 있습니다. 세 번째 경우는 일반적으로 청크 B가 잘못된 슬롯에 할당되도록 하는 여유 목록의 추가 힙 청크에 의해 발생하며 HAEPG는 B와 동일한 크기의 청크를 할당하여 추가 청크를 채웁니다. 마지막으로 HAEPG는 새로운 공격 시퀀스를 생성하고 다음을 반복합니다. 악용 가능한 메모리 상태(즉, 레이아웃 제약 조건과 일치하는 상태)로 이어지는 공격 순서를 찾을 때까지 평가 및 수정.

익스플로잇 생성. 공격 시퀀스가 평가를 통과하면 HAEPG는 기호 실행을 사용하여 공격을 실행하고 대상 프로그램의 명령을 추적하고 기호 데이터가 다음 위치 중 하나에 있는지 확인하여 익스플로잇 프리미티브를 감지합니다. (1) 메모리의 내용 및 대상 주소- 지침 및 함수 호출 작성 (2) 빈 중 하나의 헤드 포인터; (3) 간접 호출/점프의 대상 주소; (4) 프로그램 및 glibc의 함수 포인터 값(예: GOT 및 malloc 후크).

기호 데이터의 위치를 기반으로 HAEPG는 다음 익스플로잇 프리미티브 중 하나를 사용하여 익스플로잇 입력을 유도할 수 있습니다.

- 임의 실행(AX): 간접 호출의 대상 주소가 기호이거나 함수 포인터가 기호인 경우 HAEPG는 명령 포인터를 임의의 값으로 손상시킬 수 있습니다. 이 경우 HAEPG는 단일 가젯 [18] 을 대상 주소로 사용합니다. 각 단일 가젯에는 충족해야 하는 개별 메모리 및 레지스터 제약 조건이 있습니다. 따라서 HAEPG는 명령어 포인터가 손상되었을 때 관련 메모리와 레지스터를 확인하고 악용을 위한 적절한 단일 가젯을 선택합니다.

- 임의 쓰기(AW): 메모리 쓰기 명령 또는 함수 호출의 값과 대상 주소가 기호인 경우 HAEPG는 임의의 데이터를 임의의 위치에 쓸 수 있습니다. HAEPG는 이를 활용하여 GOT 또는 glibc의 함수 포인터를 덮어씁니다(Full RELRO [28] 이 활성화된 경우 glibc가 선호됨)

익스플로잇 프리미티브를 임의 실행으로 변환하는 손상된 함수 포인터를 호출하는 함수 경로를 실행합니다.

– AA(임의 할당): bin의 헤드 포인터가 기호인 경우 HAEPG는 임의의 위치에 청크를 할당할 수 있습니다. 그러나 이 프리미티브에는 bin 유형에 따라 다른 제약 조건이 있습니다. tcache의 경우 할당자는 할당된 청크의 메타 데이터를 확인하지 않으므로 프리미티브를 임의의 쓰기 프리미티브로 취급합니다. fastbin의 경우 할당자는 청크 크기의 일관성을 확인하므로 공격자는 온전성 확인을 우회하기 위해 가짜 메타 데이터를 찾거나 구성해야 합니다. 다행히도 glibc에는 온전성 검사를 우회하고 익스플로잇 프리미티브를 임의 실행2으로 변환하는 데 적합한 일부 데이터가 있습니다.

하나의 가젯으로 명령 포인터를 손상시킨 후 HAEPG는 프로세스 생성을 위해 API를 연결합니다(예: execve). HAEPG가 bash의 경로 인수로 이러한 API 호출을 감지하면 공격 시퀀스를 실행할 때 수집된 제약 조건을 해결하고 익스플로잇 입력을 생성합니다.

4 구현

4.1 정적 분석

흐름 그래프 구성을 제어합니다. Shoshitaishvili et al. [27], 간접 호출로 인해 프로그램에 대한 정확한 CFG를 복구하는 것은 매우 어렵습니다. CFG 생성은 이 문서의 기여가 아니므로 재귀 및 간접 호출이 없는 프로그램에만 중점을 둡니다. HAEPG의 프로토타입을 위한 간단한 CFG 생성 프로그램을 개발했으며, 정적 해석을 사용하여 분해에서 기본 블록의 점프 대상을 추출하여 CFG를 구성합니다.

평가 세트에는 충분합니다.

중복 기능 경로 단순화. 기능 경로는 기본 블록의 시퀀스이므로 분기는 각 분기 대상에 대해 새 기능 경로를 생성합니다. 동적 분석을 사용하여 기능 경로 간의 종속성을 추론할 때 많은 수의 기능 경로는 HAEPG의 성능 오버헤드를 증가시킵니다.

힙 상호 작용에만 초점을 맞추므로 대부분의 함수 경로는 HAEPG에 대해 중복됩니다. 힙 상호 작용이 동일하거나 유사하기 때문에 동적 분석이 반복적인 작업을 수행하게 되기 때문입니다. 또한 일부 기능 경로는 악용과 관련이 없으므로 삭제해야 합니다. 다음 방법을 사용하여 함수 경로를 단순화합니다.

– 병합: 그림 3.a 와 같이 그림 1 의 14행에서 16행까지 분기에 대한 CFG를 구성합니다 . 경로 수에 대한 분기의 영향을 줄이기 위해 경로에서 API 호출 시퀀스를 추출하고

[2] 예를 들어, 함수 포인터는 glibc에서 malloc 후크 근처에 있으며 할당자에 의해 유효한 메타 데이터로 잘못 간주될 수 있습니다. 우리는 malloc 후크를 직접 덮어쓰고 덩어리를 할당하여 명령 포인터를 가로챌 수 있습니다. –

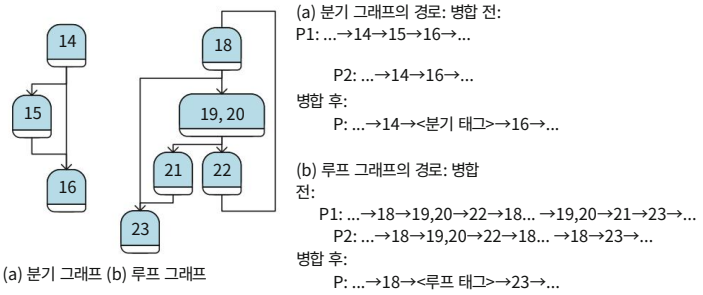

그림 3. 최적화할 두 가지 유형의 구조

API 호출 시퀀스에 힙이 포함되지 않은 경우 분기 태그가 있는 분기
상호 작용.

루프는 둘 이상의 종료를 가질 수 있으며 추가 종료는 기능 경로의 수를 증가시킵니다. 그림 1 의 func3 루프를 예로 들어 CFG가 그림 3.b 에 나와 있습니다. 루프에는 2개의 출구, 즉 21행과 23행이 있습니다. 23행은 21행의 후계자이므로 23행을 루프의 유일한 출구로 간주하고 루프를 '로 대체하여 21행 또는 21행에서 나오는 함수 경로를 병합합니다. 23을 하나로.

– 가지치기: 프로그램은 함수의 반환 값을 확인하고 함수 실행 실패를 나타내는 경우 다른 작업을 수행합니다. 이러한 작업은 일반적으로 조건부 분기로 구성되어 기능 경로의 수도 증가합니다. 함수 호출 실패를 악용하는 것은 일반적으로 힙 할당자보다 프로그램에 의존하므로 HAEPG의 범위를 벗어난 것으로 간주합니다. 우리는 해당 경로를 관련 없는 것으로 버리고 함수의 반환 값에 대한 가벼운 오염 분석을 사용하여 필터링합니다.

4.2 함수 경로의 종속성

프로그램 변수가 기호 실행에서 경로 제약 조건을 충족하지 않기 때문에 HAEPG가 함수 경로를 실행하지 못할 수 있습니다. 이 경우 HAEPG는 프로그램 변수를 올바르게 설정하는 다른 기능 경로를 먼저 실행합니다. 우리는 이러한 변수를 의존 변수라고 하며 후자의 기능 경로는 전자의 지배적인 경로입니다. 예를 들어, 기능 경로 F P1이 그림 1에서 라인 번호 27-29-30-5-7의 시퀀스이고 F P2가 27-29-31-10-11-14-15라고 가정합니다 . 가지의 줄 번호를 취하십시오). 힙 목록 [인덱스] 값이 F P2의 경로 제약 조건을 충족하지 않기 때문에 F P1을 먼저 실행하지 않고 F P2를 실행할 때 HAEPG가 10행에서 11행으로 점프하는 데 실패합니다 . 이 경우 heap list[index]는 F P2 의 의존 변수이고 F P1 은 F P2 의 지배적인 경로입니다.

–

–

HAEPG는 힙 프리미티브가 누락되는 것을 방지하기 위해 이러한 함수 경로에 대한 모든 주요 경로를 찾아야 합니다. 먼저 동적으로 함수 경로를 실행하고 다음 정보를 기록합니다. (1) 함수 경로가 원래 의존하는 값을 덮어쓰거나 업데이트하는 데 사용한 새 값; (2) 제약 조건이 충족되지 않고

목록 1. 안전하지 않은 링크 해제 템플릿

```
1 RQMT: Include(UNSORTEDBIN, hmodel.malloc_sizes) 및
2          Include(vuln.type, ["Off-by-One", "Poison-Null-Byte", "Overflow"])
        → 그리고
3          VersionLower(allocator.ver, "2.26")
4 EXEC: vul_ptr = 할당(크기 = (0x80 + RANDNUMB * 0x10 + 8), 태그 = vul.
        → vul_tag)
5          vic_ptr = 할당(크기 = (0xf0 + RANDNUMB * 0x100))
6          sep_ptr = 할당(크기 = RANDNUMB)
7 SATS: 인접(vul_ptr, vic_ptr) == True 및 인접(vic_ptr, topchunk)
        → == 거짓
8 실행: vul_data = h64(0) + h64(vul_ptr.size - 8 + 1) + h64(vul_ptr.base - 0
        → x18) + h64(vul_ptr.base - 0x10) + RANDBYTE * (vul_ptr.size - 0x28)
        → + h64(vic_ptr.size & (~8)) + "00"
9          편집(기본 = vul_ptr, 오프셋 = 0, 데이터 = vul_data)
10 SATS: vul_ptr.chunk.fd == 0 및
11          vul_ptr.chunk.bk == (vul_ptr.chunk.raw_size - 0x10) 및
12          vic_ptr.chunk.pre_size == (vic_ptr.chunk.bk - 1) 및
13          vic_ptr.chunk.raw_size & 1 == 0
14 EXEC: 할당 해제(vic_ptr)
15 EXEC: 편집(base = vul_ptr.base - 0x18, 오프셋 = 0, 데이터 = "A" * 0x20)
```

기호 실행에서 상태 종료를 유발합니다. HAEPG는 예상 값을 추론합니다.
이러한 제약 조건을 해결하여 종속 변수의
의존 변수를 올바르게 설정하십시오(즉, 지배적인 경로).

4.3 템플릿 엔진

템플릿 언어. 우리의 템플릿 언어는 요구 사항, 백본 프리미티브 및 레이아웃 제약을 통해 활용 기술을 설명합니다. 우리는 표시
RQMT, EXEC 레이블이 있는 템플릿에 포함 목록 1. 언어는 사용자에게 설명, 및 SATS,
할 기능과 개체를 제공합니다.
익스플로잇 기술 및 표 2 는 LAN의 핵심 구성 요소를 보여줍니다.
게이지.
힙의 속성을 구체화하기 위해 다음과 같은 방법을 사용합니다.
공격 시퀀스의 기본 요소:

- 직접 계산: HAEPG는 크기와 같은 템플릿과 힙 프리미티브의 범위를 기반으로 힙 프리미티브의 일부 속성을 결정합니다.
  배당. 공격 순서를 구성할 때 이러한 속성을 해결합니다.
  직접 계산이라고 합니다.
- Lazy Calculation: 힙 프리미티브의 일부 속성은
  vul ptr.chunk, vul ptr.base 및 데이터와 같이 런타임에 결정됩니다. -
  목록 1의 7행에서 편집합니다. 공격 시퀀스가
  건설된다. HAEPG는 런타임 정보를 수집하고 이러한 속성을 해결합니다.
  실행 중. 우리는 이것을 게으른 계산이라고 합니다.

Allocation 함수는 다음을 포함하는 HeapChunk 개체를 반환합니다.
멤버 변수는 메타 데이터의 필드, 사용자 페이로드 및 주소를 나타냅니다.

힙 포인터가 저장되는 위치입니다. HAEPG가 지연 계산을 사용하여 자동으로 초기화하기 때문에 사용자는 이러한 멤버 변수를 초기화할 필요가 없습니다. 목록 1 은 안전하지 않은 링크 해제 템플릿을 보여줍니다. 에 나타난 Fig. 1 의 활용을 직관적으로 설명하고 있다.

공격 시퀀스를 생성할 때 동일한 EXEC 레이블의 백본 프리미티브가 순서가 맞지 않고 HAEPG는 의존 변수의 변경을 시뮬레이션하고 의존 변수가 기능 경로의 기대치를 충족한다는 전제 하에 백본 프리미티브를 정렬합니다.

힙 시뮬레이터. 시뮬레이터의 주요 부분은 섹션에서 설명한 세 가지 힙 기본 함수를 래핑하는 루프입니다. 3.1. 힙 프리미티브를 입력으로 사용하고 해당 기능을 실행하여 대상 프로그램의 힙 상호 작용을 시뮬레이션합니다. 각 기능의 실행이 끝나면 시뮬레이터는 힙 레이아웃을 출력합니다. HAEPG는 시뮬레이터를 사용하여 대상 프로그램의 메모리 상태를 시뮬레이션합니다.

5 평가

HAEPG의 효율성을 평가하기 위해 HAEPG의 프로토타입을 구현하고 24개 프로그램 CTF 챌린지로 이를 평가했으며 모두 ctftime.org [1], pwnable.tw [7], github.com [3]에서 찾을 수 있습니다.]. 우리는 다음 기준에 따라 프로그램을 선택했습니다. (1) 프로그램에는 하나 이상의 힙 취약점이 있어야 하며 취약점이 다양합니다. (2) 점수가 높은 프로그램을 선호합니다. 일반적으로 점수가 높은 도전 과제가 더 어렵습니다. 대부분의 선택된 챌린지는 CTF 게임의 중간 점수보다 높은 점수를 가지며 그 중 4개는 익스플로잇 카테고리에서 가장 높은 점수를 받았습니다. 우리는 4가지 일반적인 힙 익스플로잇 기술에 대한 템플릿을 작성했습니다. fastbin attack [5], unsafe unlink [9], house of force [6], tcache poisoning [8]은 상당한 양의 CTF 챌린지에 적용할 수 있습니다. 그러나 일부 챌린지는 해당 glibc와 함께 제공되지 않으며 기본 glibc를 제공했습니다(tcache가 포함된 솔루션의 경우 2.27, 다른 glibc의 경우 2.24)3. 결과는 HAEPG가 대부분의 익스플로잇을 생성할 수 있음을 보여줍니다.

모든 프로그램은 Intel(R) Xeon(R) Gold 6154 CPU @ 3.00GHz*24 및 512GB RAM을 사용하여 Ubuntu18.04에서 테스트되었습니다. 모든 프로그램에 대해 NX [25] 를 활성화하고 ASLR [24] 을 비활성화했습니다. ASLR을 우회하는 것은 이 백서의 범위를 벗어난 직교 문제이기 때문에 ASLR을 비활성화했습니다.

5.1 효과

표 3 은 우리의 평가 결과를 보여줍니다. 여기에는 이름 및 CTF 대회와 같은 프로그램에 대한 세부 정보가 포함됩니다. 또한 glibc 버전, HAEPG가 식별한 취약점 유형 및 HAEPG가 익스플로잇을 생성할 수 있는지 여부를 보여줍니다.

[3] 우리는 glibc의 다른 버전에 대해 ld.so를 제공했고 모든 테스트 케이스에 대해 ld.so 및 libc.so의 절대 경로를 상대 경로로 변경했습니다. 이런 식으로 평가 시스템에 임의의 ld.so 및 libc.so를 로드할 수 있었습니다.

표 2. 템플릿 언어에서 제공하는 함수 및 변수 목록

유형 이름		설명
개체 취약		취약점 유형을 포함한 취약점 정보 및 기능(예: 오버플로된 데이터 크기)
	hmodel	함수 경로 및 그들의 힙 프리미티브
	할당자	버전 정보를 포함한 할당자
유형	Allocation에서 반환된 HeapChunk 값	
기능 포함		두 매개변수가 포함되어 있는지 확인
	할당 할당 프리미티브	
	할당 해제 프리미티브	
	편집 프리미티브 편집	
	인접한	힙 레이아웃에서 두 개의 힙 객체가 인접해 있는지 확인
	거리	두 힙 객체의 거리를 반환

그들을 위해. 결과적으로 HAEPG는 21개(87.5%) 프로그램에 대한 취약점 유형을 정확하게 추론하고 16개 프로그램에 대해 작동하는 익스플로잇을 생성합니다.
(66.7%) 입니다. 또한, 우리는 전체 RELRO [28] 를 손상시켜 우회했습니다.
GOT 대신 glibc의 함수 포인터(예: malloc 후크). –

우리는 또한 Revery [30] 와 Mechanical Phish [26] 를 비교 평가했습니다.
그리고 그들 중 누구도 이러한 프로그램에 대한 익스플로잇을 생성할 수 없습니다. 환상을 찾았습니다
이러한 도전에 대한 부패 국가. 챌린지 연결 해제 상태에 도달했습니다.
unsafe unlink로 악용될 수 있습니다(그 결과는 다음으로 표시됩니다.
표 3의 "*"). 그러나 Revery는 완전한 익스플로잇을 생성할 수 없었습니다.
퍼징을 사용하여 메모리 상태 공간을 탐색합니다. 불행히도, 이것은 충분하지 않습니다
가짜 덩어리를 위조합니다. Mechanical Phish는 이러한 문제에 대해 충돌 상태를 발견했습니다.
Driller [29]를 사용했지만 포인터가 없었기 때문에 익스플로잇을 생성할 수 없었습니다.
기호 바이트로 손상되었습니다. 따라서 Mechanical Phish는 하이재킹을 할 수 없습니다.
명령 포인터 또는 셸 코드/rop-chains 삽입.

5.2 성능

HAEPG의 성능을 평가하기 위해 HAEPG가 힙에 소비한 시간을 기록했습니다.
그림 4와 같이 상호 작용 모델링 및 익스플로잇 생성
병합 및 가지 치기, HAEPG는 기능 경로 수를 크게 줄였습니다.
그리고 대부분의 프로그램의 익스플로잇 생성을 위한 총 시간은 400초 미만입니다. 그만큼
단순화 없이 가장 많은 기능 경로를 가진 프로그램은 airCraf t입니다.
4284개의 기능 경로가 있습니다. 없는 이 프로그램의 힙 상호작용 모델링
단순화하는 데 10시간 이상이 걸렸지만 중복 기능을 제거한 후
경로 중 6개만 남기고 모델링에 344초가 소요되었습니다. 그외에 프로그램이
정교한 비트 알고리즘을 가지고 있는 note3와 같은 복잡한 알고리즘,
더 많은 시간이 필요한 복잡한 제약 조건으로 인해 많은 분석 시간이 필요합니다.
해결을 위해.

표 3. HAEPG로 평가된 CTF pwn 프로그램 목록

	–							
	– –							
	– –							
	–							
	–							

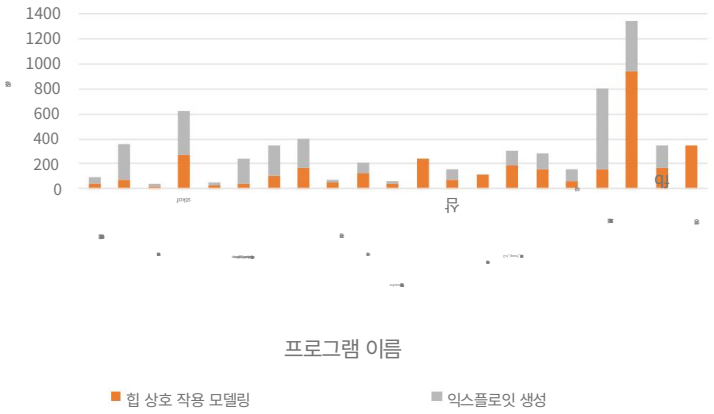

그림 4. 힙 상호작용 모델링 및 익스플로잇 생성의 시간 간격

5.3 다중 홉 악용 사례 연구

stkof를 예로 들어 HAEPG가 CTF 챌린지에 대한 익스플로잇을 자동으로 생성하는 방법을 보여줍니다. 바이너리는 각각 힙 청크를 할당, 할당 해제 및 수정하는 데 사용되는 stkof, do alloc, do dealloc 및 do edit의 세 가지 중요한 기능을 가지고 있습니다. do edit 함수의 오버플로는 공격자가 청크의 경계를 넘어 임의의 양의 데이터를 쓸 수 있도록 합니다.

취약점의 기능을 기반으로 HAEPG는 악용을 위해 안전하지 않은 링크 해제 템플릿(목록 1 참조)을 사용했습니다.

그림 5와 같이 HAEPG는 unsafe unlink 템플릿을 기반으로 먼저 Attack Sequence 1을 생성하였다. 템플릿은 취약성을 트리거할 때 vul ptr과 vic ptr이 인접해야 합니다. 그러나 glibc는 io 스트림을 초기화할 때 예기치 않게 vul ptr과 vic ptr 사이에 stdout 버퍼를 생성했습니다(HAEPG는 stkof의 힙 상호 작용만 추적하고 공유 라이브러리를 무시했기 때문에 캡처하지 않았습니다). 따라서 HAEPG는 stdout 버퍼를 먼저 해제하여 힙 레이아웃을 수정하려고 했습니다. stdout 버퍼는 glibc에 의해 생성되었으므로 이를 해제할 수 있는 함수 경로가 없습니다. 따라서 HAEPG는 새로운 힙 레이아웃을 만들려고 했습니다. 할당 프리미티브가 있는 기능 경로를 삽입하여 공격 시퀀스 2를 구성했습니다. HAEPG는 프리미티브를 사용하여 vul ptr과 동일한 크기의 청크 ph ptr을 할당했습니다. ph ptr은 vul ptr의 원래 주소에 할당되고 vul ptr과 vic ptr은 상위 주소에 강제로 할당되며 원하는 대로 서로 인접하게 됩니다. 이제 vul ptr의 오버플로로 인해 vic ptr의 메타 데이터가 손상될 수 있습니다. 마지막 할당 해제를 통해 힙 할당자는 마침내 vul ptr 내부의 가짜 청크를 연결 해제하고 임의의 기본 쓰기를 발생시켰습니다. HAEPG는 이 프리미티브를 통해 하나의 가젯 포인터로 malloc 후크를 손상시키고 do alloc의 기능 경로를 공격 시퀀스 2에 삽입하여 명령 포인터를 하이재킹했습니다. 셸이 생성되었을 때 HAEPG는 S2E에서 익스플로잇 입력을 생성했습니다.

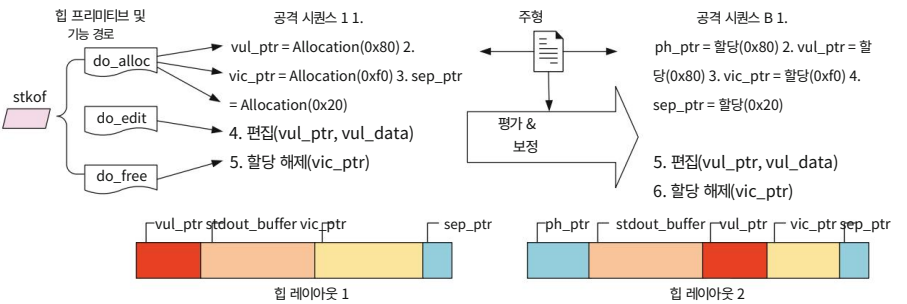

그림 5. stkof를 활용하기 위한 힙 프리미티브

5.4 실패 사례

익스플로잇 생성 실패: HAEPG가 airCraf t에 대한 fastbin의 포워드 포인터를 손상시켰습니다. 그러나 fastbin 공격에서 파생된 AA 프리미티브를 악용하려면 이 경우 크기 0x88에 유효한 가짜 메타 데이터가 필요합니다. HAEPG는 잘못된 힙 개체를 저장할 이러한 프리미티브를 찾을 수 없으므로 악용 가능한 상태를 구성할 수 없습니다. 챌린지를 악용하기 위해 공격자는 HAEPG의 능력을 넘어서는 글로벌 데이터 영역에 가짜 청크를 미리 위조해야 합니다.

HAEPG 는 프로그램이 동시에 사용할 수 있는 두 개의 힙 개체만 제공하기 때문에 기존 템플릿에는 충분하지 않기 때문에 house of atum에서 실패했습니다.
의도한 방법은 tcache 목록과 fastbin 목록을 혼동하는 것입니다.

누락 가능한 템플릿: HAEPG는 익스플로잇 생성을 위한 템플릿에 의존합니다. 즉, 기존 템플릿으로 익스플로잇 가능한 경우만 처리할 수 있으며 알려지지 않은 익스플로잇 기술을 자체적으로 사용할 수 없습니다. 예를 들어, HAEPG는 제공된 템플릿으로 악용할 수 없기 때문에 자식 tcache, iz heap lv2 및 schmaltz를 처리할 수 없습니다. 어린이 tcache, iz heap lv2를 위한 솔루션은 힙 메타 데이터를 손상시키고 통합을 트리거하여 힙 개체를 겹치는 것입니다. schmaltz와 관련하여 의도된 솔루션은 청크를 두 개의 다른 tcache 목록에 넣은 다음 AA 프리미티브를 얻기 위해 링크 포인터를 손상시키는 것입니다. _

추가 테스트를 위해 glibc 버전을 다운그레이드했습니다. HAEPG 성공은 안전하지 않은 링크 해제 및 tcache 중독 템플릿을 사용하여 iz heap lv2 및 schmaltz에 대한 익스플로잇 입력을 완전히 생성했습니다. 프로그램은 제공된 템플릿에 충분하지 않은 각 개체에 대해 쓰기 기회를 한 번만 제공하기 때문에 HAEPG는 하위 tcache를 해결할 수 없습니다.

취약점 감지 실패: HAEPG는 개체 수준에서 힙 상호 작용을 추적했기 때문에 경우에 따라 메모리 손상을 감지하지 못했습니다. 예를 들어, 챌린지 Car Market은 객체 내부에 버퍼 오버플로가 있습니다. 즉, 인접 객체가 아닌 인접 데이터 필드를 손상시킵니다. 이 경우를 처리하려면 보다 세분화된 방법이 필요합니다.

프로그램 분석 실패: HAEPG는 AES 암호화 알고리즘이 있는 SecretNoteV 2와 같은 정교한 프로그램을 분석할 수 없었고,

난독화된 프로그램인 Auir. HAEPG는 기호 실행에 의존하기 때문에 이러한 프로그램은 많은 복잡한 제약 조건과 상태를 생성하여 경로/상태 폭발을 초래했습니다.

6 관련 작업

자동 익스플로잇 생성. Mechanical Phish [26] 는 DARPA의 CGC [17] 를 위해 Shellphish 팀이 개발한 사이버 추론 시스템입니다 . Driller [29] 를 사용하여 취약점의 PoC를 찾고 Angr [27] 에서 충돌 상태를 재현 합니다.
그런 다음 입력 데이터가 충돌 지점에서 쓰기 포인터 또는 명령 포인터를 손상시키는지 확인합니다. 그렇다면 익스플로잇을 위한 쉘코드나 rop-chains를 생성할 것입니다.
결국 데이터 제약을 해결하고 익스플로잇 입력을 생성합니다.

Revery [30] 는 힙 기반 취약점에 대한 자동 익스플로잇 생성 도구입니다. 충돌 분석 및 섀도우 메모리를 사용하여 충돌하는 입력에서 메모리 손상을 감지합니다. 또한 레이아웃 지향 퍼징 기법을 사용하여 악용 가능한 지점을 검색합니다. 결국 Revery는 분기 경로와 충돌 경로를 함께 연결하여 악용을 생성합니다. 섹션에 표시된 대로. 5, Revery가 악용 가능한 지점을 탐색하는 데 사용한 퍼징 기술이 다중 홉 악용이 불가능하기 때문에 Revery는 평가 세트에서 실패했습니다.

FUZE [31] 는 커널 UAF 활용 프로세스를 자동화하는 새로운 프레임워크입니다. 커널 퍼징 및 기호 실행을 사용하여 커널 UAF 악용에 유용하고 유용한 시스템 호출을 분석 및 평가합니다.

그런 다음 동적 추적 및 SMT 솔버를 활용하여 힙 레이아웃 조작을 안내합니다. 저자는 15개의 실제 취약점을 사용하여 FUZE가 커널 UAF 악용 가능성을 확대할 수 있을 뿐만 아니라 작동 중인 악용을 다양화할 수 있음을 보여주었습니다.

SLAKE [14] 는 Linux 커널의 취약점을 악용하는 솔루션입니다. 커널 활용에 유용한 개체 및 시스템 호출을 검색하기 위해 정적-동적 하이브리드 분석을 사용합니다. 그런 다음 SLAKE는 취약성 기능을 모델링하고 해당 개체와 기능을 일치시킵니다. 취약점을 악용하기 위해 SLAKE는 취약점 유형에 따라 악용 방법을 선택하고 슬랩의 여유 목록을 조정하여 힙 레이아웃을 조작합니다.

Gollum [22] 은 인터프리터의 힙 오버플로에 대한 자동 익스플로잇 생성에 대한 첫 번째 접근 방식입니다. 사용자 지정 힙 할당자 SHAPESHIFTER를 사용하여 악용 가능한 힙 레이아웃을 생성하고 유전 알고리즘을 사용하여 대상 힙 레이아웃으로 이어질 수 있는 힙 상호 작용 시퀀스를 찾습니다. Gollum이 대상 힙 레이아웃에 도달하면 힙 오버플로를 트리거하여 희생자 개체의 함수 포인터를 손상시킵니다. HAEPG와 마찬가지로 Gollum은 악용을 생성하기 위해 단일 가젯으로 명령 포인터를 손상시킵니다.

인터프리터 또는 커널에 대한 솔루션은 언어 문법 또는 커널 시스템 호출에 대한 지식으로 힙 상태 공간을 탐색합니다. 그러나 각 응용 프로그램이 입력을 다르게 구문 분석하기 때문에 응용 프로그램에 대한 표준 입력 프로토콜은 없습니다. 우리의 방법은 프로그램 경로의 차원에서 힙 상호 작용을 모델링했으며 결과는 그 효과를 보여주었습니다.

게다가, 이러한 솔루션은 단순히 힙 레이아웃 조작 및 트리거링 취약성으로 덮어쓸 수 있는 민감한 포인터가 있다고 가정합니다.
이전에 논의한 바와 같이 전제가 항상 일부 취약점 및 애플리케이션에 적용되는 것은 아니므로 악용이 더 어려워집니다. 우리 방법은 이 문제를 효과적으로 해결할 수 있으며 이것이 다른 방법에 비해 우리 방법의 주요 이점입니다.

또한 기존의 악용 기술을 HAEPG로 인코딩하는 대신 이를 추상화하는 템플릿 언어를 개발했습니다. 사용자는 HAEPG의 내부 세부 정보를 알 필요 없이 템플릿을 작성할 수 있습니다. 위에서 언급한 솔루션에는 그러한 인터페이스가 없습니다.

힙 악용 기법 발견. Heaphopper [19] 는 메모리 손상이 있는 경우 힙 구현의 악용 가능성을 분석하기 위한 자동화된 접근 방식입니다. 힙 구현의 바이너리 라이브러리, 트랜잭션 목록(예: malloc 및 free), 공격자가 수행할 수 있는 최대 트랜잭션 수, 보안 속성 목록을 입력으로 사용합니다. Heaphopper는 트랜잭션의 순열을 열거하여 트랜잭션 목록을 생성하고 C 파일과 컴파일된 프로그램을 생성합니다. 그런 다음 이러한 프로그램을 실행하고 메모리 상태를 추적합니다. 프로그램이 보안 속성을 위반하는 상태로 이어지는 경우 Heaphopper는 해당 프로그램의 C 파일을 출력으로 사용합니다.

ARCHEAP [32] 는 새로운 힙 익스플로잇 기술을 발견하기 위해 기호 실행보다 퍼징을 사용합니다. 힙 작업 및 공격 기능을 변형 및 합성하여 테스트 케이스를 생성하고 생성된 테스트 케이스를 잠재적으로 임의 쓰기 또는 중첩 청크와 같은 익스플로잇 프리미티브를 구성하는 데 사용할 수 있는지 확인합니다. 그 결과 ARCHEAP는 ptmalloc에서 이전에 알려지지 않은 5가지 익스플로잇 프리미티브를 발견했고 jemalloc, tcmalloc 및 사용자 지정 힙 할당자에 대한 몇 가지 익스플로잇 기술을 발견했습니다.

위의 솔루션은 응용 프로그램보다 익스플로잇 기술 검색에 중점을 두므로 일반적으로 힙 할당자의 보안 평가에 사용됩니다. 우리의 솔루션은 알려진 익스플로잇 기술을 사용하여 익스플로잇을 생성할 수 있지만 알려지지 않은 익스플로잇 기술을 사용할 수는 없습니다.

7 토론

HAEPG는 힙 기반 취약점에 대한 다중 홉 악용 프로세스를 자동화하는 데 전념합니다. 그러나 다음과 같은 제한 사항이 있습니다.

- HAEPG는 정교한 프로그램이나 실제 프로그램을 분석할 수 없습니다. 첫째, HAEPG가 CFG를 추출하는 데 사용한 정적 분석은 간접 호출/점프 처리에 좋지 않습니다. 둘째, 기호 실행의 단점으로 인해 HAEPG를 작은 프로그램에만 적용할 수 있습니다. 셋째, 실제 프로그램의 상당수는 다중 스레딩 또는 다중 처리를 사용하므로 HAEPG에서 사용하는 프로그램 분석 기술에 추가적인 문제가 발생합니다.

- HAEPG는 전체 메모리 상태 공간을 탐색하지 않고 기존 템플릿을 기반으로 특정 메모리 상태만 검색하기 때문에 근본적으로 불완전합니다. 즉, HAEPG는 템플릿이 제공되지 않는 익스플로잇 기술로 익스플로잇을 생성할 수 없습니다.

– HAEPG는 ptmalloc용으로 구현되었으며 현재로서는 다른 할당자를 사용하는 프로그램에 대한 익스플로잇을 생성할 수 없습니다. HAEPG를 다른 힙 할당자에 적용하려면 힙 개체를 구문 분석하는 코드, 힙 레이아웃 조작 전략, 익스플로잇 프리미티브를 감지하고 활용하는 코드를 변경해야 합니다.

앞으로의 과제로 남겨둡니다.

8 결론

본 논문에서는 하이브리드 기술을 사용하여 힙 상호작용 모델을 구축하고 멀티홉 익스플로잇을 탐색하는 힙 취약점에 대한 자동 익스플로잇 생성 솔루션 HAEPG를 제안했습니다. HAEPG는 힙 할당자의 내부 기능을 남용하는 복잡한 악용을 생성하고 이전에는 수동으로만 완료할 수 있었던 취약점의 기능을 단계별로 향상시킬 수 있습니다. CTF 챌린지로 HAEPG를 평가한 결과 HAEPG의 효과가 나타났습니다. 결국 우리는 HAEPG가 자동화된 익스플로잇 생성의 최첨단을 개선하고 현장에서 남아 있는 문제를 해결하기 위한 유용한 빌딩 블록을 제공한다고 믿습니다.

참고문헌

1. 씨티타임. https://ctftime.org/. 액세스 2020 2. Cve 세부 정
보. https://www.cvedetails.com/. 액세스 2019 3. Github. https://
github.com/. 2020년에 액세스함 4. gnu c 라이브러리(glibc). https://
www.gnu.org/software/libc/. Accessed 2019 5. 힙 악용 - fastbin 공격. https://0x00sec.org/t/heap-
exploitation fastbin-attack/3627. Accessed 2019 6. malloc maleficarum glibc malloc 악용 기술. https://
dl. packetstormsecurity.net/papers/attack/MallocMaleficarum.txt. 액세스 2020

7. Pwnable.tw. https://pwnable.tw/. 액세스 2020 8. tcache
poisoning.c. https://github.com/shellphish/how2heap/blob/master/glibc 2.26/tcache 중독.c. 2019년 –
에 액세스함 9. 익스플로잇을 연결 해제합니다. https://heap-exploitation.dhavalkapil.com/attacks/unlink
익스플로잇. –
HTML. 액세스 2019
10. Honggfuzz: 흥미로운 분석 옵션이 있는 사용하기 쉬운 범용 fuzzer(2017). https://github.com/google/
honggfuzz. Accessed 2020 11. Cwe-193: Off-by-one 오류(2019). https://cwe.mitre.org/data/
definitions/193.
HTML. 액세스 2019
12. Avgerinos, T., Cha, SK, Hao, BLT, Brumley, D.: AEG: 자동 익스플로잇 생성. In: 제18회 네트워크 및 분산 시스템 보안 심포지엄(2011)
13. Cha, SK, Avgerinos, T., Rebert, A., Brumley, D.: 이진 코드의 혼란을 일으키십시오. 2012년 보안 및 개인 정보 보호에 관한 IEEE 심포지엄. IEEE(2012)
14. Chen, Y., Xing, X.: Slake: Linux 커널의 취약점을 악용하기 위한 슬랩 조작 촉진. In: 2019 ACM SIGSAC 컴퓨터 및 통신 보안 컨퍼런스. ACM (2019)
15. Chipounov, V., Kuznetsov, V., Candea, G.: S2e: 소프트웨어 시스템의 생체 내 다중 경로 분석을 위한 플랫폼. In: ACM SIGARCH 컴퓨터 아키텍처 뉴스, vol. 39. ACM (2011)

16. cwe.mitre.org: Cwe-626: 널 바이트 상호 작용 오류(포이즌 널 바이트). https://cwe.mitre.org/data/definitions/626.html. 액세스 2019
17. DARPA: 사이버 그랜드 챌린지. https://www.cybergrandchallenge.com/. 액세스 2019
18. david942j@217: [프로젝트] glibc의 단일 가젯. https://david942j.blogspot.com/2017/02/project-one-gadget-in-glibc.html. 액세스 2019
19. Eckert, M., Bianchi, A., Wang, R., Shoshitaishvili, Y., Kruegel, C., Vigna, G.: Heaphopper: 힙 구현 보안에 제한된 모델 검사를 가져옵니다. 2018년 제27회 USENIX 보안 심포지엄(USENIX 보안 18)
20. Heelan, S.: 소프트웨어에 대한 제어 흐름 하이재킹 익스플로잇의 자동 생성 취약점. 박사 옥스퍼드 대학교 논문(2009)
21. Heelan, S., Melham, T., Kroening, D.: 악용을 위한 자동 힙 레이아웃 조작. 2018년 제27회 USENIX 보안 심포지엄(USENIX 보안 18)
22. Heelan, S., Melham, T., Kroening, D.: Gollum: 인터프리터의 힙 오버플로에 대한 모듈식 및 그레이박스 익스플로잇 생성. In: 2019 ACM SIGSAC 컴퓨터 및 통신 보안 컨퍼런스. ACM (2019)
23. Huang, SK, Huang, MH, Huang, PY, Lai, CW, Lu, HL, Leong, WM: Crax: 공격을 상징적 연속으로 모델링하여 자동 익스플로잇 생성을 위한 소프트웨어 충돌 분석. In: 2012년 IEEE 6차 소프트웨어 보안 및 신뢰성 국제 회의. IEEE(2012)
24. PaX-Team: Pax 주소 공간 레이아웃 무작위화. https://pax.grsecurity.net/문서/aslr.txt. 액세스 2019
25. PaX-Team: Pax 비실행 페이지 디자인 및 구현. https://팍스.grsecurity.net/docs/noexec.txt. 액세스 2019
26. Shoshitaishvili, Y., et al.: 기계식 피싱: 탄력적인 자율 해킹. 에: 2018 IEEE 보안 및 개인 정보 보호 심포지엄(2018)
27. Shoshitaishvili, Y., et al.: Sok: (state of) war of war: 이진 분석의 공격 기술. In: 2016 IEEE Symposium on Security and Privacy. IEEE(2016)
28. Sidhpurwala, H.: 재배치 읽기 전용(relro)을 사용하여 엘프 바이너리 강화. https://www.redhat.com/en/blog/hardening-elf-binaries-using-relocation-read only-relro. Accessed 2019 29. Stephens, N., et al.: Driller: 선택적 기호 실행을 통한 퍼징 증대. In: 제23회 네트워크 및 분산 시스템 보안 심포지엄, vol. 16 (2016)
30. Wang, Y., et al.: Revery: 개념 증명에서 악용 가능으로. In: 2018 ACM SIGSAC 컴퓨터 및 통신 보안 컨퍼런스. ACM (2018)
31. Wu, W., Chen, Y., Xu, J., Xing, X., Gong, X., Zou, W.: Fuze: 커널 사용 후 자유 취약점에 대한 익스플로잇 생성을 촉진하는 방향으로. 2018년 제27회 USENIX 보안 심포지엄(USENIX 보안 18)
32. Yun, I., Kapil, D., Kim, T.: 새로운 힙 익스플로잇 프리미티브를 체계적으로 발견하는 자동 기술. arXiv 사전 인쇄 arXiv:1903.00503 (2019)
33. Zalewski, M.: 미국 퍼지 룹(2017). https://lcamtuf.coredump.cx/afl/