

Praktyczna analiza plików binarnych : zbuduj własne narzędzia na platformie Linux do instrumentacji binarnej, analizy i deasemblacji / Dennis Andriessse. – Wydanie 1. – Warszawa, © 2020

Spis treści

SŁOWO WSTĘPNE	XVII
PRZEDMOWA	XXI
PODZIĘKOWANIA	XXIII
WSTĘP 1	
Czym jest analiza binarna i dlaczego jej potrzebujesz?	2
Dlaczego analiza binarna stanowi wyzwanie?	3
Kto powinien przeczytać tę książkę?	4
Co jest w tej książce?	4
Jak korzystać z tej książki?	6
Model programowy procesora	6
Składnia asemblera	6
Format binarny i platforma systemowa	7
Przykładowy kod i maszyna wirtualna	7
CZĘŚĆ I FORMATY BINARNE	1
1	
ANATOMIA PLIKU BINARNEGO	11
1.1. Proces kompilacji	12
1.1.1. Faza preprocesowania	12
1.1.2. Faza kompilacji	14
1.1.3. Faza asemblacji	16
1.1.4. Faza konsolidacji (linkowania)	16
1.2. Symbole i okrojone pliki binarne	18
1.2.1. Podgląd symboli w pliku	18
1.2.2. Kolejny plik binarny przechodzi na ciemną stronę: „okrajanie” pliku	20
1.3. Deasemblacja pliku binarnego	20
1.3.1. Zagłądanie do wnętrza pliku obiektowego	20
1.3.2. Badamy kompletny wykonywalny plik binarny	22
1.4. Ładowanie i wykonywanie pliku binarnego	26
1.5. Podsumowanie	28
2	
FORMAT ELF	29

2.1. Nagłówek pliku	31
2.1.1. Tablica e_dent	31
2.1.2. Pola e_type, e_machine i e_version	33
2.1.3. Pole e_entry	33
2.1.4. Pola e_phoff i e_shoff	34
2.1.5. Pole e_flags	34
2.1.6. Pole e_ehsize	34
2.1.7. Pole e_entsize i e_num	34
2.1.8. Pole e_shstrndx	35
2.2. Nagłówki sekcji	35
2.2.1. Pole sh_name	36
2.2.2. Pole sh_type	37
2.2.3. Pole sh_flags	37
2.2.4. Pola sh_addr, sh_offset oraz sh_size	38
2.2.5. Pole sh_link	38
2.2.6. Pole sh_info	38
2.2.7. Pole sh_addralign	38
2.2.8. Pole sh_entsize	38
2.3. Sekcje	38
2.3.1. Sekcje .init i .fini	40
2.3.2. Sekcja .text	40
2.3.3. Sekcje .bss, .data i .rodata	41
2.3.4. Wiązanie leniwe symboli i sekcje .pit, .got oraz .got.plt	42
2.3.5. Sekcje .rei* i .rela*	45
2.3.6. Sekcja .dynamic	46
2.3.7. Sekcje .init_array i .fini_array	48
2.3.8. Sekcje .shstrtab, .symtab, .strtab, .dynsym i .dynstr	48
2.4. Nagłówki programów	49
2.4.1. Pole p_type	50
2.4.2. Pole p_flags	51
2.4.3. Pola p_offset, p_vaddr, p_paddr, p_filesz i p_memsz	51
2.4.4. Pole p_align	51
2.5. Podsumowanie	52
3	
FORMAT PE: KRÓTKIE WPROWADZENIE	53
3.1. Nagłówek MS-DOS i stub MS-DOS	54
3.2. Sygnatura, nagłówek i nagłówek opcjonalny pliku PE	54
3.2.1. Sygnatura pliku PE	57
3.2.2. Nagłówek pliku PE	57
3.2.3. Nagłówek opcjonalny pliku PE	58
3.3. Tablica nagłówków sekcji	58
3.4. Sekcje	59
3.4.1. Sekcje .edata i .idata	59
3.4.2. Wyrównywanie w sekcjach kodu PE	60
3.5. Podsumowanie	61

4

TWORZENIE PROGRAMU ŁADUJĄCEGO PLIKI BINARNE PRZY UŻYCIU LIBBFD	63
4.1. Czym jest libbfd?	64
4.2. Prosty interfejs do ładowania plików binarnych	64
4.2.1. Klasa Binary	67
4.2.2. Klasa Section	67
4.2.3. Klasa Symbol	67
4.3. Implementacja loadera plików binarnych	67
4.3.1. Inicjalizacja libbfd i otwieranie pliku binarnego	69
4.3.2. Parsowanie podstawowych właściwości pliku binarnego	71
4.3.3. Ładowanie symboli	73
4.3.4. Ładowanie sekcji	76
4.4. Testowanie loadera binarnego	78
4.5. Podsumowanie	81

CZĘŚĆ II PODSTAWY ANALIZY BINARNEJ

5

PODSTAWOWA ANALIZA BINARNA W LINUXIE	85
5.1. Rozwiązywanie kryzysów tożsamości przy użyciu file	86
5.2. Użycie ldd do badania zależności	89
5.3. Oglądanie zawartości pliku w xxd	90
5.4. Parsowanie ekstrahowanego ELF za pomocą readelf	92
5.5. Parsowanie symboli za pomocą nm	94
5.6. Szukanie wskazówek za pomocą strings	97
5.7. Siedzenie wywołań systemowych i bibliotecznych za pomocą strace i ltrace	100
5.8. Badanie zachowań na poziomie instrukcji przy użyciu objdump	104
5.9. Zrzut dynamicznego bufora łańcucha za pomocą gdb	106
5.10. Podsumowanie	108

6

DEASEMBLACJA I PODSTAWY ANALIZY BINARNEJ	109
6.1. Deasemblacja statyczna	110
6.1.1. Deasemblacja liniowa	111
6.1.2. Deasemblacja rekurencyjna	112
6.2. Deasemblacja dynamiczna	116
6.2.1. Przykład: śledzenie wykonywania pliku binarnego za pomocą gdb	116
6.2.2. Strategie pokrycia kodu	119
6.3. Strukturyzacja deasemblowanego kodu i danych	123
6.3.1. Strukturyzowanie kodu	123
6.3.2. Strukturyzacja danych	130
6.3.3. Dekompilacja	132
6.3.4. Reprezentacje pośrednie	133

6.4. Podstawowe metody analizy	135
6.4.1. Własności analizy binarnej	135
6.4.2. Analiza przepływu sterowania	139
6.4.3. Analiza przepływu danych	141
6.5. Wpływ ustawień kompilatora na deasemblację	145
6.6. Podsumowanie	146

7

PROSTE METODY WSTRZYKIWANIA KODU DLA ELF	147
7.1. Najprostsze modyfikacje binarne przy użyciu edycji hex	147
7.1.1. Obserwowanie błędu off-by-one w akcji	148
7.1.2. Naprawienie błędu off-by-one	151
7.2. Modyfikacja zachowania biblioteki współdzielonej przy użyciu LD_PRELOAD	154
7.2.1. Zagrożenie przepełnieniem sterty	155
7.2.2. Wykrywanie przepełnienia sterty	157
7.3. Wstrzykiwanie sekcji kodu	160
7.3.1. Wstrzykiwanie sekcji ELF: przegląd wysokiego poziomu	161
7.3.2. Użycie elfinject do wstrzykiwania sekcji ELF	163
7.4. Wywołanie wstrzykniętego kodu	166
7.4.1. Modyfikacja adresu startowego	167
7.4.2. Przejęcie konstruktorów i destruktorów	170
7.4.3. Przejęcie wpisów w GOT	173
7.4.4. Przejęcie wpisów PLT	176
7.4.5. Przekierowanie wywołań bezpośrednich i pośrednich	177
7.5. Podsumowanie	178

CZĘŚĆ III ZAAWANSOWANA ANALIZA BINARNA

8

DOSTOSOWYWANIE DEASEMBLACJI DO POTRZEB	181
8.1. Dlaczego pisać własny program do deasemblacji	182
8.1.1. Sprawa dla dostosowania deasemblacji: zaciemniony kod	182
8.1.2. Inne powody, by napisać dostosowany deassembler	185
8.2. Wprowadzenie do Capstone'a	186
8.2.1. Instalacja Capstone'a	186
8.2.2. Deasemblacja liniowa z Capstone'a	187
8.2.3. Eksploracja API C Capstone'a	192
8.2.4. Deasemblacja rekurencyjna z Capstone	193
8.3. Implementacja skanera gadżetów ROP	202
8.3.1. Wprowadzenie do programowania zorientowanego na powrót (ROP)	202
8.3.2. Wykrywanie gadżetów ROP	204
8.4. Podsumowanie	210

9

INSTRUMENTACJA BINARNA	213
9.1. Czym jest instrumentacja binarna?	214
9.1.1. API instrumentacji binarnej	214
9.1.2. Statyczna kontra dynamiczna instrumentacja binarna	215
9.2. Statyczna instrumentacja binarna	216
9.2.1. Podejście int 3	216
9.2.2. Podejście z użyciem trampolin	218
9.3. Dynamiczna instrumentacja binarna	223
9.3.1. Architektura systemu DBI	223
9.3.2. Wprowadzenie do Pin	225
9.4. Profilowanie za pomocą Pin	226
9.4.1. Struktury danych profilera i kod instalacyjny	226
9.4.2. Parsowanie symboli funkcji	229
9.4.3. Instrumentacja bloków podstawowych	230
9.4.4. Instrumentacja instrukcji przepływu sterowania	232
9.4.5. Liczenie instrukcji, transferów sterowania i wywołań systemowych	235
9.4.6. Testowanie profilera	236
9.5. Automatyczne rozpakowywanie plików binarnych z Pin	240
9.5.1. Wprowadzenie do programów pakujących	240
9.5.2. Struktury danych i kod instalacyjny programu rozpakowującego	242
9.5.3. Instrumentowanie zapisów do pamięci	244
9.5.4. Instrumentowanie instrukcji przepływu sterowania	245
9.5.5. Śledzenie zapisów do pamięci	245
9.5.6. Wykrywanie pierwotnego punktu wejścia i rzucanie rozpakowanego pliku binarnego	247
9.5.7. Testowanie programu rozpakowującego	248
9.6. Podsumowanie	252

10

ZASADY DYNAMICZNEJ ANALIZY TAINT	255
10.1. Czym jest DTA?	256
10.2. DTA w trzech krokach: źródła taint, taint sinks i propagacja taint	256
10.2.1. Definiowanie źródeł taint	256
10.2.2. Definiowanie taint sinks	257
10.2.3. Siedzenie propagacji taint	257
10.3. Użycie DTA do wykrycia błędu Heartbleed	258
10.3.1. Krótki przegląd podatności na Heartbleed	258
10.3.2. Wykrywanie Heartbleed przez taint	259
10.4. Czynniki w projektowaniu DTA: ziarnistość taint, kolory taint i zasady propagacji taint	261
10.4.1. Ziarnistość taint	261
10.4.2. Kolory taint	262
10.4.3. Zasady propagacji taint	263
10.4.4. Nadmierne i niedostateczne pokrycie taint	264
10.4.5. Zależności sterowania	264

10.4.6. Shadow memory	265
10.5. Podsumowanie	267
11	
PRAKTYCZNA DYNAMICZNA ANALIZA TAINT Z LIBDFT	269
11.1. Wstęp do libdft	269
11.1.1. Organizacja wewnętrzna libdft	270
11.1.2. Zasady taint	272
11.2. Wykorzystanie DTA do wykrywania zdalnego przechwycenia sterowania	273
11.2.1. Sprawdzanie informacji o taint	276
11.2.2. Źródła taint: taintowanie otrzymywanych bajtów	277
11.2.3. Taint sinks: sprawdzanie argumentów execve	279
11.2.4. Wykrywanie próby przejęcia przepływu sterowania	280
11.3. Obejście DTA za pomocą przepływów niejawnych	285
11.4. Detektor eksfiltracji danych oparty na DTA	286
11.4.1. Źródła taint: śledzenie taint dla otwartych plików	289
11.4.2. Taint sinks: monitorowanie przesyłów sieciowych w poszukiwaniu eksfiltracji danych	292
11.4.3. Wykrywanie próby eksfiltracji danych	293
11.5. Podsumowanie	296
12	
PODSTAWY WYKONYWANIA SYMBOLICZNEGO	297
12.1. Przegląd wykonywania symbolicznego	298
12.1.1. Wykonywanie symboliczne kontra konkretne	298
12.1.2. Rodzaje i ograniczenia wykonywania symbolicznego	302
12.1.3. Zwiększanie skalowalności wykonywania symbolicznego	307
12.2. Rozwiązywanie ograniczeń za pomocą Z3	309
12.2.1. Dowodzenie osiągalności instrukcji	309
12.2.2. Dowodzenie nieosiągalności instrukcji	312
12.2.3. Dowodzenie poprawności formuły	313
12.2.4. Upraszczenie wyrażeń	314
12.2.5. Modelowanie ograniczeń dla kodu maszynowego za pomocą wektorów bitowych	315
12.2.6. Rozwiązywanie nieprzejrystego kodu warunkowego za pomocą wektorów bitowych	317
12.3. Podsumowanie	318
13	
PRAKTYCZNE WYKONYWANIE SYMBOLICZNE Z TRITONEM	321
13.1. Wprowadzenie do Tritona	322
13.2. Utrzymywanie stanu symbolicznego przy użyciu drzew składniowych (AST)	323
13.3. Slicing wsteczny z Tritonem	325
13.3.1. Pliki nagłówkowe i konfigurowanie Tritona	328

13.3.2. Plik konfiguracji symbolicznej	328
13.3.3. Emulowanie instrukcji	329
13.3.4. Konfigurowanie architektury Tritona	331
13.3.5. Obliczanie „wycinka wstecznego”	332
13.4. Wykorzystanie Tritona do zwiększenia pokrycia kodu	333
13.4.1. Tworzenie zmiennych symbolicznych	335
13.4.2. Znajdowanie modelu dla nowej ścieżki	336
13.4.3. Testowanie narzędzia pokrycia kodu	339
13.5. Automatyczna eksploatacja podatności	342
13.5.1. Podatny program	342
13.5.2. Znajdowanie adresu podatnego miejsca wywołania	346
13.5.3. Budowanie generatora exploita	348
13.5.4. Uzyskiwanie dostępu do powłoki administratora	354
13.6. Podsumowanie	356

CZĘŚĆ IV DODATKI A

A

BŁYSKAWICZNY KURS ASEMBLERA X86	361
A.1. Zarys programu w języku asemblera	362
A.1.1. Instrukcje języka asemblera, dyrektywy, etykiety i komentarze	362
A.1.2. Rozdzielenie kodu od danych	363
A.1.3. Składnia AT&T i Intela	364
A.2. Struktura instrukcji x86	364
A.2.1. Reprezentacja instrukcji x86 na poziomie języka asemblera	364
A.2.2. Instrukcje x86 na poziomie kodu maszynowego	364
A.2.3. Operandy rejestru	365
A.2.4. Operandy pamięci	367
A.2.5. Operandy bezpośrednie	368
A.3. Częste instrukcje x86	368
A.3.1. Porównywanie operandów i ustawianie flag stanu	370
A.3.2. Implementacja wywołań systemowych	370
A.3.3. Implementacja skoków warunkowych	370
A.3.4. Ładowanie adresów pamięci	371
A.4. Częste konstrukcje kodu w języku asemblera	371
A.4.1. Stos	371
A.4.2. Wywołania funkcji i ramki funkcji	372
A.4.3. Instrukcje warunkowe	376
A.4.4. Pętle	378

B

IMPLEMENTACJA NADPISYWANIA PT_NOTE PRZY UŻYCIU LIBELF	379
B.1. Wymagane pliki nagłówkowe	380
B.2. Struktury danych używane w elfinject	380
B.3. Inicjalizacja libelf	381

B.4. Uzyskiwanie nagłówka pliku wykonywalnego	385
B.5. Znajdowanie segmentu PT_NOTE	386
B.6. Wstrzykiwanie bajtów kodu	387
B.7. Dopasowanie adresu ładowania dla wstrzykiwanej sekcji	388
B.8. Nadpisanie nagłówka sekcji .note.ABI-tag	388
B.9. Ustawienie nazwy dla wstrzykniętej sekcji	393
B.10. Nadpisanie nagłówka programu PT_NOTE	395
B.11. Modyfikacja adresu startowego	397

C

SPIS NARZĘDZI ANALIZY BINARNEJ 399

C.1. Deasemblerzy	399
C.2. Debuggery	401
C.3. Platformy deasemblerów	401
C.4. Platformy analizy binarnej	402

D

DALSZA LEKTURA 403

D.1. Standardy i źródła	403
D.2. Artykuły i raporty techniczne	404
D.3. Książki	406

INDEKS 407