

Spis treści

Wstęp 15

1. Funkcje, struktury, klasy i obiekty na niskim poziomie 19

1.1. Wywoływanie funkcji w językach (bardzo) niskiego poziomu 21

1.1.1. CALL, RET i konwencje wywołań 21

1.1.2. Konwencje wywołań x86 24

1.1.3. Konwencje wywołań x86-64 27

1.2. Struktury 28

1.2.1. „Zgadywanie” wielkości i ułożenia elementów struktury w pamięci ... 30

1.2.2. Rozpoznawanie budowy struktur lokalnych i globalnych 30

1.2.3. Rozpoznawanie budowy struktur dynamicznie alokowanych 32

1.3. Klasy, obiekty, dziedziczenie i tablice wirtualne 34

1.3.1. Prosta klasa a struktura 34

1.3.2. Obiekty = struktury + funkcje + thiscall 38

1.3.3. Wszystko zostaje w rodzinie, czyli dziedziczenie 38

1.4. Podsumowanie 41

2. Środowisko uruchomieniowe na systemach GNU/Linux 43

2.1. Wstęp 45

2.2. Pliki wykonywalne ELF 45

2.2.1. Identyfikacja systemu i architektury docelowej 46

2.2.2. Segmenty 49

2.2.3. Segment PT_LOAD 51

2.2.4. Segment PT_DYNAMIC 53

2.2.5. Sekcja .dynamic 53

2.2.5.1. Deklaracja bibliotek zależnych 57

2.2.5.2. Wczesna inicjalizacja programu 59

2.3. Środowisko uruchomieniowe 62

2.3.1. Kod PIC 62

2.3.2. Tablice GOT i PLT 65

2.3.3. Program ładujący ld.so 73

2.3.3.1.	Zmienne środowiskowe	73
2.3.3.2.	LD_LIBRARY_PATH	74
2.3.3.3.	LD_PRELOAD	75
2.3.3.4.	LD_AUDIT	77
2.3.4.	Zrzucanie pamięci procesów	81
2.3.4.1.	System plików /proc	81
2.3.4.2.	Pliki specjalne w /proc/pid	82
2.3.4.3.	Pliki specjalne maps i mem.	83
2.3.4.4.	VDSO	85
2.3.4.5.	Wektory inicjalizacyjne	86
2.3.5.	Wstrzykiwanie kodu	88
2.3.5.1.	Wersja ptrace(2)	95
2.3.6.	Samomodyfikujący się kod	96
2.4.	Podsumowanie	98
	Bibliografia	99
3.	Mechanizmy ochrony aplikacji	101
3.1.	Wstęp	103
3.2.	Przepełnienie bufora na stosie	104
3.3.	Procedury obsługi wyjątków	108
3.4.	Zapobieganie wykonaniu danych	112
3.5.	Losowość układu przestrzeni adresowej	114
3.6.	Dodatkowe materiały	116
3.7.	Podsumowanie	116
	Bibliografia	116
4.	Metody przeciwdziałania odbudowie kodu aplikacji z pamięci procesu	119
4.1.	Wstęp	121
4.1.1.	Struktura rozdziału	121
4.2.	Packery, szyfratory i protektory plików PE	122
4.3.	Emulowanie zabezpieczeń	126
4.4.	Ochrona przed odbudową kodu programu zapisanego z pamięci procesu	128
4.5.	Nanomity	130
4.6.	Skradziony punkt rozpoczęcia programu	135
4.7.	Przekierowanie i obfuskacja importowanych funkcji	142
4.9.	Podsumowanie	147
	Bibliografia	148
5.	NET internals – format & RE	151
5.1.	Wstęp	153
5.2.	Format pliku .NET	153
5.2.1.	JIT, czyli drugi stopień kompilacji programu	153
5.2.2.	Język pośredni CIL	154

5.2.2.1.	Przykład 1: add()	155
5.2.2.2.	Przykład 2: stringWriteTest()	155
5.2.2.3.	Przykład 3: programFlowTest()	157
5.2.3.	Dedykowane kontenery metadanych.	161
5.2.4.	Wysokopoziomowa struktura programu zachowana w pliku wykonywalnym.	161
5.2.5.	Token	164
5.2.6.	Funkcje natywne	165
5.2.7.	AOT	165
5.3.	Inżynieria wsteczna	166
5.3.1.	Analiza statyczna	166
5.3.2.	Dekompilacja	166
5.3.3.	Rekompilacja i odtwarzanie działania programu	170
5.3.4.	Modyfikacja istniejących metod	170
5.3.5.	Debug	174
5.3.5.1.	dnSpy	174
5.3.5.2.	Wtyczki do WinDbg	176
5.4.	Metody protekcji plików .NET	176
5.4.1.	Nadpisywanie nazw nadanych przez użytkownika	176
5.4.2.	Szyfrowanie stałych tekstowych	176
5.4.3.	Utrudnianie dekompilacji	178
5.4.4.	Ukrywanie kodu programu	178
5.4.5.	Deobfuskacja	179
5.5.	Podsumowanie	180
	Bibliografia	180
6.	Python – obfuskacja i inżynieria wsteczna	181
6.1.	Wstęp	183
6.2.	Obfuskacja a model wykonania	183
6.3.	Obfuskacja źródeł	185
6.4.	Pliki .pyc i .pyo	188
6.5.	Bundlery i kompilator	190
6.5.1.	Py2exe	190
6.5.2.	cx_Freeze	193
6.5.3.	PyInstaller	194
6.5.4.	Nuitka	198
6.5.5.	Inne bundlery i kompilatory	205
6.6.	Obiekty code i function	205
6.7.	Kod bajtowy i gdzie go szukać	213
6.8.	Prosta obfuskacja kodu bajtowego	216
6.9.	Samomodyfikujący się kod bajtowy	222
6.10.	Podsumowanie	224
	Bibliografia	224

7.	Malware w owczej skórze, czyli o wstrzykiwaniu kodu w inne procesy	225
7.1.	Wstęp	227
7.2.	Przegląd technik wstrzykiwania	228
7.2.1.	Schemat działania	228
7.2.2.	Przygotowania przed implementacją	228
7.2.3.	Wybieranie celu	229
7.2.3.1.	Wstrzykiwanie kodu do istniejącego procesu	229
7.2.3.2.	Wstrzykiwanie kodu do nowo uruchomionego procesu ...	230
7.2.4.	Wpisywanie kodu do zdalnego procesu	231
7.2.5.	Metody przekierowania do wstrzykniętego kodu	233
7.2.5.1.	Uruchomienie dodanego kodu w nowym wątku	233
7.2.5.2.	Dodawanie do istniejącego wątku (przy użyciu NtQueueApcThread)	234
7.2.5.3.	Nadpisywanie punktu wejścia procesu	235
7.2.5.4.	Nadpisywanie kontekstu procesu	238
7.2.5.5.	Dodawanie do okienka Tray	239
7.2.5.6.	PowerLoader	241
7.3.	Tworzenie wstrzykiwalnego kodu	242
7.3.1.	Podstawy shellcodu	242
7.3.2.	Warunki wstrzykiwania plików PE	244
7.3.2.1.	Wstępne przygotowanie pliku PE	245
7.3.3.	Samodzielne ładowanie pliku PE	246
7.3.3.1.	Konwersja surowego obrazu na wirtualny	246
7.3.3.2.	Pobieranie adresów tabel	248
7.3.3.3.	Rozwiązywanie relokacji	250
7.3.3.4.	Rozwiązywanie importów	253
7.3.3.5.	Przekierowanie wykonania do dodanego pliku PE	261
7.3.3.6.	Plik PE o cechach shellcodu	261
7.3.4.	Debugowanie wstrzykniętego kodu	262
7.4.	Metody wstrzykiwania plików PE	264
7.4.1.	Klasyczny DLL Injection	265
7.4.1.1.	Demonstracja	265
7.4.1.2.	Implementacja	266
7.4.2.	RunPE (Process Hollowing)	267
7.4.2.1.	Demonstracja	267
7.4.2.2.	Implementacja	269
7.4.3.	ChimeraPE	273
7.4.3.1.	Demonstracja	274
7.4.3.2.	Implementacja	275
7.4.4.	Reflective DLL injection	277
7.4.4.1.	Demonstracja	277
7.4.4.2.	Implementacja	278
7.4.5.	Wstrzyknięcia DLL na poziomie systemu operacyjnego	282

7.4.5.1. AppInit_DLLs	283
7.4.5.2. Shim + InjectDll Tag.	283
7.5. Podsumowanie	285
Bibliografia	286
8. ELFie brudne sztuczki	287
8.1. Wstęp	289
8.2. ELFie struktury	289
8.3. Rozdwojenie jaźni	292
8.3.1. Niewidzialny kod	292
8.4. Misja: przechwycić resolver	299
8.5. Klasyka wiecznie żywa.	302
8.7. DYNAMICzne zmiany	307
8.7.1. Co dwie tablice to nie jedna	307
8.7.2. Jeden bajt by nimi rządzić.	309
8.8. Podsumowanie	311
Bibliografia	311
9. Łamanie zaawansowanych technik przekierowania API.	313
9.1. Wstęp	315
9.2. Format PE – podstawy	315
9.3. Tabela importów	319
9.4. Tabela eksportów	321
9.5. Proste metody obfuskacji IAT	322
9.6. Przekierowania API z przepisywaniem kodu	323
9.7. Zróbmy to lepiej!	325
9.8. Weryfikacja hipotezy pod debuggerem	326
9.9. Automatyczne przepisywanie plików DLL	326
9.10. Pierwsza próba i pierwsze problemy	331
9.11. Modyfikacja NTDLL	333
9.12. Ostatnie poprawki	335
9.13. Efekt końcowy	336
9.14. Podsumowanie	337
Bibliografia	337
10. Śledzenie ścieżki wykonania procesu w systemie Linux	339
10.1. Wstęp	341
10.2. Metody programowe	342
10.2.1. Instrumentacja kodu źródłowego	342
10.2.2. Instrumentacja programu w ramach kompilacji	344
10.2.3. Instrumentacja kodu binarnego	347
10.2.4. Śledzenie wywołań systemowych	349
10.2.5. Śledzenie wywołań funkcji bibliotecznych	350
10.2.6. Emulacja instrukcji	351

10.2.7. Nagrywanie i odtwarzanie przebiegu ścieżki wykonania	353
10.3. Metody programowo-sprzętowe	355
10.3.1. Próbkowanie instrukcji	355
10.3.2. Rejestry debuggera	356
10.3.3. Wykonanie krokowe	359
10.3.4. BTF (Branch Trap Flag)	362
10.3.5. LBR (Intel Last Branch Record)	363
10.3.6. BTS (Intel Branch Trace Store)	365
10.3.7. IPT (Intel Processor Trace)	367
10.4. Metody sprzętowe	370
10.5. Porównanie wydajności	371
10.6. Podsumowanie	372
11. Ciasteczko zagłady – studium przypadku eksploatacji routera	373
11.1. Wstęp	375
11.2. Konsola	375
11.3. Architektura MIPS	379
11.4. Ciasteczko i ból brzucha	381
11.5. Co w trawie piszczy	383
11.5.1. Funkcja sub_80244E88	386
11.5.2. Funkcja sub_802CF6E8	389
11.5.3. Funkcja sub_802457D0	394
11.5.4. Argumenty dla funkcji sub_8024281C	396
11.5.5. Funkcja sub_8024281C	398
11.6. Wykorzystanie podatności	400
11.6.1. Adres docelowy	401
11.7. Podsumowanie	404
Bibliografia	406
12. W pogoni za flagą – eksploatacja na systemach Windows i Linux	407
12.1. Wstęp	409
12.2. Lokalne testowanie rozwiązań	410
12.3. Multipurpose Calculation Machine (średnio trudne)	411
12.4. Memory (łatwe)	424
12.5. Crypto Machine (średnio trudne)	433
12.6. Quarantine (bardzo trudne)	444
12.7. Night Sky (bardzo trudne)	461
12.8. Bubblegum (bardzo trudne)	480
12.9. Antipasto (łatwe)	501
12.10. Entree (średnio trudne)	511
12.11. Podsumowanie	518
Bibliografia	518
Indeks	521