

Bug Bounty Bootcamp

*Przewodnik po tropieniu
i zgłaszaniu luk w zabezpieczeniach*



Helion 

Vickie Li



Tytuł oryginału: Bug Bounty Bootcamp: The Guide to Finding and Reporting Web Vulnerabilities

Tłumaczenie: Lech Lachowski

ISBN: 978-83-283-9411-7

Copyright © 2021 by Vickie Li. Title of English-language original: Bug Bounty Bootcamp: The Guide to Finding and Reporting Web Vulnerabilities, ISBN 9781718501546, published by No Starch Press Inc. 245 8th Street, San Francisco, California United States 94103.

The Polish-language edition Copyright © 2022 by Helion S.A. under license by No Starch Press Inc. All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/bugbou>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

PRZEDMOWA	15
WSTĘP	17
CZĘŚĆ I. BRANŻA	21
1	
WYBÓR PROGRAMU BUG BOUNTY	23
Stan branży	23
Typy zasobów	24
Serwisy i aplikacje społecznościowe	25
Ogólne aplikacje internetowe	26
Aplikacje mobilne (Android, iOS i Windows)	26
Interfejsy API	27
Kod źródłowy i programy wykonywalne	27
Sprzęt i internet rzeczy	28
Platformy bug bounty	28
Zalety	29
Wady	29
Zakresy, wypłaty i czasy reakcji	30
Zakres programu	30
Kwoty wypłat	31
Czas reakcji	32
Programy prywatne	32
Wybór właściwego programu	33
Szybkie porównanie popularnych programów	34
2	
JAK ZAPRACOWAĆ NA SUKCES?	36
Pisanie dobrych raportów	36
Krok 1. Wymyśl opisowy tytuł	37
Krok 2. Napisz przejrzyste podsumowanie	37
Krok 3. Uwzględnij ocenę dotkliwości	37

Krok 4. Podaj szczegółowe kroki do odtworzenia ataku	39
Krok 5. Przedstaw dowód koncepcji	40
Krok 6. Opisz wpływ i scenariusze ataku	40
Krok 7. Zarekomenduj możliwe środki zaradcze	41
Krok 8. Sprawdź raport	41
Dodatkowe wskazówki dotyczące pisania lepszych raportów	41
Budowanie relacji z zespołem programistycznym	43
Stany raportu	43
Radzenie sobie z konfliktami	45
Budowanie partnerstwa	45
Dlaczego nie odnosisz sukcesu?	46
Dlaczego nie znajdujesz błędów?	46
Dlaczego Twoje raporty są odrzucane?	48
Co robić, gdy utkniesz w miejscu?	50
Krok 1. Zrób sobie przerwę!	50
Krok 2. Zbuduj zestaw umiejętności	51
Krok 3. Zyskaj nową perspektywę	51
Na koniec kilka słów z doświadczenia	52

CZĘŚĆ II. ZACZYNAMY! 53

3	
JAK DZIAŁA INTERNET?	55
Model klient-serwer	55
System nazw domenowych	56
Porty internetowe	57
Żądania i odpowiedzi HTTP	58
Kontrola bezpieczeństwa w internecie	60
Kodowanie zawartości	60
Zarządzanie sesją i pliki cookie HTTP	61
Uwierzytelnianie oparte na tokenach	62
Tokeny sieciowe JSON	63
Reguła tego samego pochodzenia	66
Ucz się programować	67
4	
KONFIGURACJA ŚRODOWISKA I PRZECHWYTYWANIE RUCHU	68
Wybór systemu operacyjnego	68
Konfigurowanie narzędzi podstawowych — przeglądarka i serwer proxy	69
Otwieranie osadzonej przeglądarki	70
Konfigurowanie przeglądarki Firefox	70
Konfigurowanie Burpa	73
Korzystanie z Burpa	75
Proxy	75
Intruder	78

Repeater	80
Decoder	81
Comparer	81
Zapisywanie żądań Burpa	82
Ostatnie uwagi na temat robienia notatek	82
5	
REKONESANS PRZED HAKOWANIEM APLIKACJI INTERNETOWYCH	84
Ręczna trawersacja celu	85
Google dorking	85
Wykrywanie zakresu	89
WHOIS i odwrotne WHOIS	89
Adresy IP	89
Parsowanie certyfikatu	91
Enumeracja poddomen	92
Enumeracja usług	93
Brute forcing katalogów	94
Indeksowanie witryn	96
Hostowanie zewnętrzne	98
Rekonesans z wykorzystaniem GitHuba	101
Inne podstawowe techniki białego wywiadu	102
Fingerprinting stosu technologicznego	103
Pisanie własnych skryptów rekonesansowych	106
Podstawy pisania skryptów bashowych	106
Zapisywanie w pliku danych wyjściowych narzędzia	109
Dodanie do danych wyjściowych daty skanowania	111
Dodawanie opcji wyboru uruchamianych narzędzi	111
Uruchamianie dodatkowych narzędzi	112
Parsowanie wyników	116
Tworzenie raportu głównego	118
Skanowanie wielu domen	120
Pisanie biblioteki funkcji	125
Budowanie interaktywnych programów	126
Używanie specjalnych zmiennych i znaków	129
Planowanie automatycznego skanowania	132
Kilka słów na temat interfejsów API rekonesansu	134
Zacznij hakować!	135
Narzędzia wymienione w tym rozdziale	135
Wykrywanie zakresu	135
Biały wywiad	137
Fingerprinting stosu technologicznego	137
Automatyzacja	138

CZĘŚĆ III. LUKI W ZABEZPIECZENIACH SIECIOWYCH 139

6

CROSS-SITE SCRIPTING	141
Mechanizmy	142
Rodzaje ataków XSS	146
Zapisywany XSS	146
Ślepy XSS	147
Odbijany XSS	148
XSS oparty na modelu DOM	148
XSS własny	150
Zapobieganie	150
Tropienie błędów XSS	152
Krok 1. Szukaj możliwości wprowadzania danych	152
Krok 2. Wstawiaj ładunki	154
Krok 3. Potwierdź działanie ładunku	158
Omijanie ochrony przed atakami XSS	159
Alternatywna składnia JavaScriptu	159
Wielkość liter i kodowanie	159
Błędy logiki filtra	161
Eskalacja ataku	161
Automatyzacja tropienia błędów XSS	162
Szukanie pierwszego błędu XSS!	163

7

OTWARTE PRZEKIEROWANIA	164
Mechanizmy	164
Zapobieganie	166
Tropienie błędów open redirect	166
Krok 1. Poszukaj parametrów przekierowania	166
Krok 2. Użyj Google dorkingu, aby znaleźć dodatkowe parametry przekierowania	167
Krok 3. Przetestuj otwarte przekierowania oparte na parametrach	169
Krok 4. Przetestuj otwarte przekierowania oparte na odsyłaczach	169
Omijanie ochrony przed atakami open redirect	169
Używanie autokorekty przeglądarki	170
Eksploatacja logiki wadliwego walidatora	171
Korzystanie z adresów URL danych	172
Eksploatacja dekodowania URL	173
Łączenie technik eksploatacji	175
Eskalacja ataku	175
Szukanie pierwszego otwartego przekierowania!	177

8		
PORYWANIE KLIKNIEĆ		178
Mechanizmy		178
Zapobieganie		184
Tropienie błędów clickjackingu		186
Krok 1. Poszukaj działań zmieniających stan		186
Krok 2. Sprawdź nagłówki odpowiedzi		187
Krok 3. Potwierdź lukę w zabezpieczeniach		187
Obchodzenie zabezpieczeń		188
Eskalacja ataku		189
Uwagi na temat dostarczania ładunku clickjackingu		190
Szukanie pierwszej luki clickjackingu!		191
9		
FAŁSZOWANIE ŻĄDAŃ PRZESYŁANYCH MIĘDZY WITRYNAMI		192
Mechanizmy		192
Zapobieganie		196
Tropienie błędów CSRF		198
Krok 1. Namierz działania zmieniające stan		199
Krok 2. Poszukaj brakujących zabezpieczeń przed CSRF		199
Krok 3. Potwierdź lukę w zabezpieczeniach		200
Omijanie ochrony przed CSRF		201
Wykorzystanie clickjackingu		202
Zmiana metody żądania		202
Obejście tokenów CSRF przechowywanych na serwerze		203
Omijanie podwójnie przesyłanych tokenów CSRF		205
Omijanie kontroli nagłówka referencyjnego CSRF		207
Omijanie ochrony przed CSRF przy użyciu ataku XSS		209
Eskalacja ataku		209
Wykorzystanie luki CSRF do spowodowania wycieku informacji o użytkownikach		209
Wykorzystanie luki CSRF do utworzenia zapisywanego XSS-a własnego		210
Wykorzystanie luki CSRF do przejmowania kont użytkowników		211
Dostarczanie ładunku CSRF-a		212
Szukanie pierwszej luki CSRF!		214
10		
NIEZABEZPIECZONE BEZPOŚREDNIE ODWOŁANIA DO OBIEKTÓW		215
Mechanizmy		215
Zapobieganie		217
Tropienie błędów IDOR		218
Krok 1. Utwórz dwa konta		218
Krok 2. Odkryj oferowane funkcjonalności		219
Krok 3. Przechwytuj żądania		220
Krok 4. Zmieniaj identyfikatory		220

Omijanie ochrony przed IDOR-ami	221
Kodowane i mieszane identyfikatory	222
Wyciekające identyfikatory	223
Przełącz aplikację na tryb debugowania, nawet jeśli o niego nie prosi	223
Szukaj IDOR-ów ślepych	224
Zmień metodę żądania	225
Zmień typ żądanego pliku	225
Eskalacja ataku	226
Automatyzacja ataku	226
Szukanie pierwszego IDOR-a!	227
11	
WSTRZYKNIĘCIE SQL	228
Mechanizmy	229
Wstrzykiwanie kodu do zapytań SQL	230
Korzystanie ze wstrzyknięcia SQL drugiego rzędu	233
Zapobieganie	234
Tropienie błędów wstrzyknięcia SQL	237
Krok 1. Szukaj klasycznych wstrzyknięć SQL	238
Krok 2. Szukaj ślepych wstrzyknięć SQL	239
Krok 3. Eksfiltruj informacje za pomocą wstrzyknięć SQL	241
Krok 4. Szukaj wstrzyknięć NoSQL	242
Eskalacja ataku	244
Zdobądź wiedzę o bazie danych	244
Uzyskaj dostęp do powłoki internetowej	245
Automatyzacja wstrzyknięć SQL	246
Szukanie pierwszego wstrzyknięcia SQL!	246
12	
SYTUACJE WYŚCIGU	248
Mechanizmy	248
Kiedy sytuacja wyścigu staje się luką w zabezpieczeniach?	250
Zapobieganie	253
Tropienie sytuacji wyścigu	253
Krok 1. Znajdź funkcjonalności podatne na sytuację wyścigu	254
Krok 2. Wysyłaj żądania jednocześnie	254
Krok 3. Sprawdź wyniki	254
Krok 4. Przygotuj dowód słuszności koncepcji	255
Eskalacja sytuacji wyścigu	255
Szukanie pierwszej sytuacji wyścigu!	256

13	
FAŁSZOWANIE ŻĄDAŃ WYKONYWANYCH PO STRONIE SERWERA	257
Mechanizmy	257
Zapobieganie	259
Tropienie błędów SSRF	260
Krok 1. Znajdź funkcjonalności podatne na ataki SSRF	260
Krok 2. Przekaż potencjalnie podatnym punktom końcowym wewnętrzne adresy URL	262
Krok 3. Sprawdź wyniki	263
Omijanie zabezpieczeń przed atakami SSRF	265
Omijanie list elementów dozwolonych	265
Omijanie list elementów blokowanych	267
Eskalacja ataku	270
Wykonaj skanowanie sieci	270
Pozyskaj metadane instancji	272
Eksploituuj ślepe SSRF	274
Zaatakuj sieć	275
Szukanie pierwszego SSRF!	276
14	
NIEZABEZPIECZONA DESERIALIZACJA	277
Mechanizmy	277
PHP	278
Java	288
Zapobieganie	291
Tropienie błędów niezabezpieczonej deserializacji	292
Eskalacja ataku	293
Szukanie pierwszej niezabezpieczonej deserializacji!	293
15	
ENCJA ZEWNĘTRZNA XML-A	294
Mechanizmy	294
Zapobieganie	296
Tropienie błędów XXE	297
Krok 1. Znajdź punkty wejścia danych XML-a	297
Krok 2. Przeprowadź testy pod kątem klasycznych XXE	298
Krok 3. Przeprowadź testy pod kątem ślepych XXE	299
Krok 4. Osadź ładunki XXE w różnych typach plików	300
Krok 5. Przeprowadź testy pod kątem ataków XInclude	301
Eskalacja ataku	302
Odczytywanie plików	302
Inicjowanie ataków SSRF	303
Używanie ślepych XXE	303
Przeprowadzanie ataków DoS	306
Kilka słów o eksfiltracji danych przy użyciu XXE	307
Szukanie pierwszego błędu XXE!	309

16	
WSTRZYKNIĘCIE SZABLONU	310
Mechanizmy	310
Silniki szablonów	311
Kod wstrzyknięcia szablonu	312
Zapobieganie	315
Tropienie błędów wstrzyknięcia szablonu	315
Krok 1. Szukaj lokalizacji wprowadzania danych przez użytkownika	315
Krok 2. Wykrywaj wstrzykiwanie szablonu dzięki przesyłaniu ładunków testowych	316
Krok 3. Określ stosowany silnik szablonów	317
Eskalacja ataku	318
Szukanie dostępu do systemu za pomocą kodu Pythona	319
Ucieczka z piaskownicy przy użyciu wbudowanych funkcji Pythona	320
Przesyłanie ładunków do testowania	323
Automatyzacja wstrzyknięcia szablonu	324
Szukanie pierwszego błędu wstrzyknięcia szablonu!	324
17	
BŁĘDY LOGIKI APLIKACJI I USZKODZONA KONTROLA DOSTĘPU	326
Błędy logiki aplikacji	327
Uszkodzona kontrola dostępu	329
Udostępnione panele administracyjne	329
Luki trawersacji katalogów	330
Zapobieganie	331
Tropienie błędów logiki aplikacji i uszkodzonej kontroli dostępu	331
Krok 1. Poznaj swój cel	332
Krok 2. Przechwytyj żądania podczas przeglądania	332
Krok 3. Myśl nieszablonowo	332
Eskalacja ataku	332
Szukanie pierwszego błędu logiki aplikacji lub uszkodzonej kontroli dostępu!	333
18	
ZDALNE WYKONYWANIE KODU	334
Mechanizmy	334
Wstrzyknięcie kodu	335
Załączenie pliku	337
Zapobieganie	339
Tropienie błędów RCE	340
Krok 1. Zbierz informacje o celu	341
Krok 2. Zidentyfikuj podejrzone lokalizacje wprowadzania danych przez użytkownika	341
Krok 3. Prześlij ładunki testowe	341
Krok 4. Potwierdź lukę w zabezpieczeniach	343
Eskalacja ataku	343
Omijanie ochrony przed RCE	344
Szukanie pierwszego błędu RCE!	346

19	
BŁĘDY REGUŁY TEGO SAMEGO POCHODZENIA	347
Mechanizmy	348
Eksploatacja mechanizmu CORS	349
Eksploatacja metody postMessage()	351
Eksploatacja JSONP	353
Omijanie SOP-u przy użyciu luki XSS	355
Tropienie błędów obejścia SOP-u	355
Krok 1. Sprawdź stosowanie technik rozluźniania SOP-u	355
Krok 2. Znajdź błędną konfigurację CORS-u	356
Krok 3. Znajdź błędy postMessage	357
Krok 4. Znajdź błędy JSONP	358
Krok 5. Rozważ środki zaradcze	358
Eskalacja ataku	358
Szukanie pierwszego błędu obejścia SOP-u!	359
20	
KWESTIE BEZPIECZEŃSTWA ZWIĄZANE Z POJEDYŃCZYM LOGOWANIEM	360
Mechanizmy	361
Współdzielenie plików cookie	361
SAML	363
OAuth	366
Tropienie błędów przejęcia poddomeny	370
Krok 1. Zrób listę poddomen celu	370
Krok 2. Znajdź niezarejestrowane strony	371
Krok 3. Zarejestruj stronę	371
Monitorowanie pod kątem przejęcia poddomeny	372
Tropienie błędów SAML-a	373
Krok 1. Znajdź odpowiedź SAML-a	373
Krok 2. Analizuj pola odpowiedzi	374
Krok 3. Omiń podpis	374
Krok 4. Ponownie zakoduj komunikat	375
Tropienie kradzieży tokenów OAuth	375
Eskalacja ataku	375
Szukanie pierwszego błędu obejścia SSO!	376
21	
UJAWNIEŃ INFORMACJI	377
Mechanizmy	377
Zapobieganie	378
Tropienie błędów ujawnienia informacji	379
Krok 1. Spróbuj ataku trawersacji ścieżek	379
Krok 2. Przeszukaj Wayback Machine	380
Krok 3. Przeszukaj strony wklejania i udostępniania tekstu	382

Krok 4. Zrekonstruuuj kod źródłowy z udostępnionego katalogu .git	382
Krok 5. Znajdź informacje w publicznych plikach	386
Eskalacja ataku	387
Szukanie pierwszego błędu ujawnienia informacji!	387

CZĘŚĆ IV. TECHNIKI ZAAWANSOWANE 389

22

PRZEPROWADZANIE INSPEKCI KODU	391
Porównanie testów białej i czarnej skrzynki	392
Podejście szybkie: grep jest Twoim najlepszym przyjacielem	392
Niebezpieczne wzorce	392
Wyciekające sekrety i słabe szyfrowanie	395
Nowe poprawki i nieaktualne zależności	396
Komentarze programistów	397
Funkcjonalności debugowania, pliki konfiguracyjne i punkty końcowe	397
Podejście szczegółowe	398
Ważne funkcje	398
Dane wprowadzone przez użytkownika	398
Ćwiczenie — znajdź luki w zabezpieczeniach	401

23

HAKOWANIE APLIKACJI SYSTEMU ANDROID	404
Konfigurowanie serwera proxy dla urządzeń mobilnych	405
Omijanie przypinania certyfikatu	407
Anatomia APK	407
Narzędzia	409
Android Debug Bridge	409
Android Studio	410
Apktool	410
Frida	410
Mobile Security Framework	411
Tropienie luk w zabezpieczeniach	411

24

HAKOWANIE INTERFEJSÓW API	413
Czym są interfejsy API?	413
Interfejsy RESTful API	415
Interfejsy API SOAP	416
Interfejsy GraphQL API	417
Aplikacje skoncentrowane na API	420
Tropienie błędów API	420
Przeprowadzanie rekonesansu	421
Testowanie pod kątem uszkodzonej kontroli dostępu i wycieków informacji	423
Testowanie pod kątem problemów z ograniczaniem przepustowości	425
Testowanie pod kątem błędów technicznych	425

AUTOMATYCZNE WYKRYWANIE LUK W ZABEZPIECZENIACH ZA POMOCĄ FUZZERÓW	427
Czym jest fuzzing?	428
Jak działa fuzzer internetowy?	428
Proces fuzzingu	429
Krok 1. Określ punkty wstrzyknięcia danych	429
Krok 2. Wybierz listę ładunków	430
Krok 3. Rozpocznij fuzzing	431
Krok 4. Monitoruj wyniki	432
Fuzzing za pomocą Wfuzza	433
Enumeracja ścieżek	433
Brute forcing uwierzytelniania	434
Testowanie pod kątem typowych luk w zabezpieczeniach sieciowych	436
Więcej informacji o Wfuzzie	437
Porównanie fuzzingu i analizy statycznej	437
Pułapki fuzzingu	438
Poszerzanie zestawu narzędzi do zautomatyzowanego testowania	438
SKOROWIDZ	440

5

Rekonesans przed hakowaniem aplikacji internetowych



PIERWSZYM KROKIEM PODCZAS PRZYGOTOWYWANIA ATAKU NA DOWOLNY CEL JEST PRZEPROWADZENIE *ROZPOZNANIA*, CZYLI PO PROSTU ZEBRANIE INFORMACJI O CELU. REKONESANS JEST ISTOTNY, PONIEWAŻ W TEN sposób określa się powierzchnię ataku dla danej aplikacji. Aby zwiększyć efektywność szukania błędów, przed podjęciem decyzji o wyborze najskuteczniejszego podejścia musisz odkryć wszystkie możliwe sposoby ataku na obiekt docelowy.

Jeżeli aplikacja nie korzysta np. z PHP, nie ma powodu, by testować ją pod kątem luk w zabezpieczeniach związanych z PHP, a jeśli organizacja nie korzysta z usług AWS (ang. *Amazon Web Services*), nie należy tracić czasu na próby hakowania jej wiaderek. Zrozumienie sposobu działania obiektu docelowego pomaga zbudować solidne podstawy do znajdowania luk w zabezpieczeniach. Umiejętności zwiadowcze odróżniają hakera dobrego od nieefektywnego.

W tym rozdziale przedstawię najbardziej użyteczne dla tropiciela bug bounty techniki rekonesansu. Potem omówię podstawy pisania skryptów powłoki bash w celu automatyzacji i usprawnienia zadań zwiadowczych. **Bash** jest interpreterem powłoki dostępnym w systemach macOS i Linux. Chociaż podczas pisania tego rozdziału przyjąłam założenie, że używasz systemu Linux, wiele z wymienionych narzędzi powinno być w stanie zainstalować również w innych systemach

operacyjnych. Niektóre z narzędzi omówionych w tym rozdziale będą wymagały od Ciebie uprzedniej instalacji. Na końcu rozdziału zamieściłam linki do wszystkich opisanych narzędzi.

Przed przejściem do działania upewnij się, że możesz przeprowadzić inwazyjny rekonesans swojego celu, zanim spróbujesz jakichkolwiek technik, które wymagają aktywnej współpracy obiektu docelowego. Działania, w szczególności takie jak skanowanie portów, indeksowanie internetowe i brute forcing katalogów, mogą generować wiele niechcianego ruchu w witrynie i mogą nie być mile widziane przez daną organizację.

Ręczna trawersacja celu

Zanim przejdziesz do jakichkolwiek czynności związanych z hakowaniem, pomocne będzie ręczne przejście przez aplikację, by dowiedzieć się więcej na jej temat. Przeglądając każdą stronę i klikając każdy link, staraj się odkryć w aplikacji wszystkie funkcjonalności, do których użytkownicy mogą uzyskać dostęp. Spróbuj uzyskać dostęp do funkcjonalności, których zwykle nie używasz.

Jeśli hakujesz np. Facebooka, próbujesz utworzyć wydarzenie, zagrać w jakąś grę i skorzystać z funkcji płatności, jeżeli nigdy wcześniej tego nie robiłeś. Rejestrujesz konto na każdym poziomie uprawnień, aby odkryć wszystkie funkcjonalności aplikacji. W Slacku możesz utworzyć np. właścicieli, administratorów i członków obszaru roboczego. Tworzysz również użytkowników, którzy są członkami różnych kanałów w tym samym obszarze roboczym. W ten sposób możesz zobaczyć, jak wygląda aplikacja dla różnych użytkowników.

Powinno dać Ci to przybliżone wyobrażenie o tym, jak wygląda **powierzchnia ataku** (wszystkie punkty, w których atakujący może próbować eksploatować aplikację), gdzie znajdują się punkty wprowadzania danych i jak różni użytkownicy wchodzi z sobą w interakcje. Wtedy możesz już rozpocząć dogłębny proces rekonesansu — poznawanie technologii i struktury aplikacji.

Google dorking

Podczas tropienia błędów często trzeba zbadać szczegóły jakiejś luki w zabezpieczeniach. Jeśli eksploatujesz potencjalną podatność na ataki XSS (ang. *Cross-Site Scripting*), możesz potrzebować odnaleźć konkretny ładunek, który widziałeś na GitHubie. Zaawansowane umiejętności korzystania z silnika wyszukiwania pomogą Ci szybko i dokładnie wyszukiwać potrzebne zasoby.

Tak naprawdę zaawansowane wyszukiwanie w Google'ach to potężna technika, której hakerzy często używają do przeprowadzania rekonesansu. Nazywają to **Google dorking** lub **Google hacking**. Dla przeciętnego użytkownika Google to tylko narzędzie do tekstowego wyszukiwania obrazów, filmów i stron internetowych.

Jednak dla hakera Google może być sposobem na pozyskanie cennych informacji, takich jak ukryte portale administracyjne, odblokowane pliki haseł i wycieki kluczy uwierzytelniania.

Wyszukiwarka Google ma własny wbudowany język zapytań, który pomaga filtrować wyszukiwania. Oto kilka z najbardziej użytecznych operatorów, których można używać w dowolnej wyszukiwarce Google:

site

Instruuje wyszukiwarke Google, by wyświetliła wyniki tylko z określonej witryny. Pomoże Ci to szybko znaleźć najbardziej renomowane źródło związane z badanym tematem. Jeżeli chcesz wyszukać np. składnię funkcji `print()` Pythona, możesz ograniczyć wyniki do oficjalnej dokumentacji, stosując wyszukiwanie `print site:python.org`.

inurl

Wyszukuje strony o adresie URL zgodnym z łańcuchem znaków wyszukiwania. Jest to skuteczny sposób wyszukiwania podatnych na ataki stron w określonej witrynie. Przyjmijmy, że przeczytałeś na blogu post o tym, że istnienie w witrynie strony o nazwie `/course/jumpto.php` może wskazywać, że ta witryna internetowa jest podatna na zdalne wykonywanie kodu. Wyszukując `inurl:"/course/jumpto.php" site:example.com`, możesz sprawdzić, czy Twój cel ma tę lukę w zabezpieczeniach.

intitle

Znajduje określone łańcuchy znaków w tytule strony. Jest to przydatne, gdyż umożliwia wyszukiwanie stron zawierających określony typ zawartości. Na serwerach strony z listą plików często mają w swoich tytułach tekst *index of*. Aby wyszukać strony katalogów w określonej witrynie, możesz zastosować wyszukiwanie `intitle:"index of" site:example.com`.

link

Wyszukuje strony internetowe zawierające linki do określonego adresu URL. Możesz używać tego do wyszukiwania dokumentacji dotyczącej niejasnych technologii lub luk w zabezpieczeniach. Załóżmy, że badasz mało znaną lukę ReDoS (ang. *Regular Expression Denial-of-Service*). Z łatwością znajdziesz w internecie jej definicję, ale możesz mieć trudności ze znalezieniem przykładów. Operator `link` może pomóc wyszukać strony, które odwołują się do strony Wikipedii tej luki w zabezpieczeniach, aby znaleźć dyskusje na ten sam temat: `link:"https://en.wikipedia.org/wiki/ReDoS"`.

filetype

Wyszukuje strony z określonym rozszerzeniem pliku. Jest to niesamowite narzędzie do hakowania; hakerzy często używają go do lokalizowania na swoich stronach docelowych plików, które mogą być podatne na ataki, jak pliki dzienników i haseł.

Zapytanie `filetype:log site:example.com` wyszukuje np. w docelowej witrynie pliki dziennika mające często rozszerzenie `.log`.

Znak wieloznaczny (*)

W wyszukiwaniach możesz użyć operatora znaku wieloznacznego (*), który określa **dowolny znak lub łańcuch znaków**. Zapytanie "jak hakować * za pomocą Google'a" zwróci np. każdy łańcuch znaków zaczynający się od *jak hakować*, a kończący na *za pomocą Google'a*. Dopasuje następujące łańcuchy znaków: *jak hakować strony internetowe za pomocą Google'a*, *jak hakować aplikacje za pomocą Google'a* itd.

Cudzysłów (" ")

Umieszczenie wyszukiwanego hasła w cudzysłowie wymusza dokładne dopasowanie. Zapytanie "jak hakować" będzie szukać stron z wyrażeniem *jak hakować*. Zapytanie bez cudzysłowu również wyszuka strony zawierające słowa *jak* oraz *hakować*, ale niekoniecznie w jednym wyrażeniu.

Alternatywa (|)

Operator alternatywy jest określany za pomocą znaku pionowej kreski (|) i może być używany do wyszukiwania jednego lub drugiego hasła albo obu jednocześnie. Znak pionowej kreski musi być otoczony spacjami. Zapytanie "jak hakować" `site:(reddit.com | stackoverflow.com)` będzie np. szukać określenia *jak hakować* na stronie Reddit lub Stack Overflow, a zapytanie `(SQL Injection | SQLi)` będzie szukać stron internetowych, które wspominają o *SQL Injection* lub *SQLi*. *SQLi* jest akronimem często używanym w odniesieniu do ataków wstrzykiwania SQL i omówię je w rozdziale 11.

Minus (-)

Operator minusa (-) wyklucza niektóre wyniki wyszukiwania. Powiedzmy, że jesteś zainteresowany poznaniem stron internetowych opisujących hakowanie, ale nie tych, które omawiają hakowanie PHP. Zapytanie "jak hakować strony internetowe" `-php` wyszuka strony zawierające hasło *jak hakować strony internetowe*, ale bez określenia *php*.

Zaawansowane opcje wyszukiwarki możesz wykorzystywać również na wiele innych sposobów, które pomogą Ci zwiększyć efektywność pracy. Aby znaleźć więcej informacji na ten temat, możesz nawet wyszukać hasło operatory wyszukiwania Google'a. Operatory te mogą być bardziej przydatne, niż można by się spodziewać. Po wpisaniu poniższego hasła, możesz znaleźć np. wszystkie poddomeny firmy:

```
site:*.example.com
```

Możesz również poszukać specjalnych punktów końcowych, które mogą prowadzić do luk w zabezpieczeniach. **Kibana** to narzędzie służące do wizualizacji danych, wyświetlające dane operacyjne serwera, takie jak dzienniki serwera, komunikaty debugowania i status serwera. Zhakowanie instancji Kibany może umożliwić atakującemu zbieranie obszernych informacji na temat działania witryny. Wiele dashboardów Kibany jest uruchamianych w ścieżce *app/kibana*, dlatego poniższe zapytanie ujawni, czy obiekt docelowy ma dashboard Kibany. Jeśli tak, będziesz mógł spróbować uzyskać dostęp do tego dashboardu, by sprawdzić, czy jest on niezabezpieczony:

```
site:example.com inurl:app/kibana
```

Google umożliwia wyszukanie zasobów danej firmy hostowanych zewnętrznie w internecie, np. w wiaderkach S3 Amazon (omówię tę opcję szerzej w następnym podrozdziale):

```
site:s3.amazonaws.com nazwa_firmy
```

Powinieneś szukać specjalnych rozszerzeń, które mogą wskazywać na poufny plik. Oprócz rozszerzenia *.log*, które z reguły wskazuje pliki dziennika, szukaj rozszerzeń *.php*, *.cfm*, *.asp*, *.jsp* i *.pl*, które są często używane dla plików skryptów:

```
site:example.com ext:php
site:example.com ext:log
```

Możesz także łączyć wyszukiwane hasła w celu dokładniejszego wyszukiwania. Poniższe zapytanie wyszukuje w witrynie *example.com* pliki tekstowe zawierające słowo *hasło*:

```
site:example.com ext:txt hasło
```

Poza konstruowaniem własnych zapytań możesz zapoznać się ze stroną *Google Hacking Database* (<https://www.exploit-db.com/google-hacking-database/>) używaną przez hakerów i specjalistów ds. bezpieczeństwa do udostępniania zapytań Google'a dla wyszukiwania informacji związanych z bezpieczeństwem. Zawiera ona wiele zapytań, które mogą być pomocne podczas procesu rekonesansu. Możesz znaleźć np. zapytania, które wyszukują pliki zawierające hasła, popularne adresy URL portali administracyjnych lub strony zbudowanych przy użyciu oprogramowania podatnego na ataki.

Podczas przeprowadzania rekonesansu za pomocą wyszukiwarki Google pamiętaj, że jeśli będziesz wysyłać dużo zapytań, Google zacznie wymagać CAPTCHA dla wyszukiwań przeprowadzanych przez użytkowników z Twojej sieci. Może to być dla nich irytujące, dlatego nie polecam stosowania Google dorkingu w sieci firmowej lub współdzielonej.

Wykrywanie zakresu

Przejdźmy teraz do właściwego rekonesansu. Po pierwsze, zawsze sprawdzaj zakres celu. **Zakres** programu (znajdziesz go na stronie z warunkami udziału w programie) określa, które poddomeny, produkty i aplikacje możesz atakować. Dokładnie zweryfikuj, które z zasobów firmy znajdują się we wskazanym zakresie, aby uniknąć przekroczenia granic podczas procesów rekonesansu i hakowania. Jeśli warunki korzystania z *example.com* określają, że poza zakresem są *dev.example.com* i *test.example.com*, nie należy przeprowadzać żadnych rekonesansów ani ataków na te poddomeny.

Po zweryfikowaniu tych informacji dowiedz się, co faktycznie znajduje się w zakresie: które domeny, poddomeny i adresy IP możesz atakować oraz jakie zasoby firmy hostuje organizacja na tych komputerach.

WHOIS i odwrotne WHOIS

Gdy firmy lub osoby fizyczne rejestrują nazwę domeny, muszą przekazać rejestratorowi domeny informacje identyfikujące, takie jak adres pocztowy, numer telefonu i adres e-mailowy. Każdy może sprawdzić te informacje za pomocą polecenia `whois`, które wyszukuje dane dotyczące rejestratorów i właścicieli wszystkich znanych domen. Często można znaleźć także powiązane informacje kontaktowe, takie jak adres e-mailowy, imię i nazwisko, adres lub numer telefonu:

```
$ whois facebook.com
```

Informacje te nie zawsze są dostępne, ponieważ niektóre organizacje i osoby fizyczne korzystają z usługi zwanej **prywatnością domeny**, której zewnętrzny dostawca zastępuje informacje o użytkowniku informacjami o usłudze pośredniczącej.

Możesz przeprowadzić również **odwrotne wyszukiwanie WHOIS**, przeszukując bazę danych pod kątem nazwy organizacji, numeru telefonu lub adresu e-mailowego, by znaleźć zarejestrowane domeny. W ten sposób można znaleźć wszystkie domeny należące do tego samego właściciela. Odwrotne WHOIS jest niezwykle przydatne do wyszukiwania niejawnych lub wewnętrznych domen, które nie zostały w inny sposób ujawnione publicznie. Do przeprowadzania tego wyszukiwania możesz użyć publicznego narzędzia odwrotnego WHOIS, takiego jak `ViewDNS.info` (<https://viewdns.info/reversewhois/>). WHOIS i odwrotne WHOIS dostarczą Ci przyzwoity zbiór domen najwyższego poziomu, z którymi będziesz mógł pracować.

Adresy IP

Kolejnym sposobem odkrywania domen najwyższego poziomu należących do obiektu docelowego jest lokalizowanie adresów IP. Adres IP znanej Ci domeny znajdziesz po uruchomieniu polecenia `nslookup`. W poniższym listingu możesz zobaczyć, że *facebook.com* znajduje się pod adresem 157.240.2.35:

\$ nslookup facebook.com

Server: 192.168.0.1
Address: 192.168.0.1#53
Non-authoritative answer:
Name: facebook.com
Address: 157.240.2.35

Po znalezieniu adresu IP znanej domeny wykonaj wyszukiwanie odwrotne. **Odwrotne wyszukiwanie adresów IP** polega na wyszukiwaniu domen hostowanych na tym samym serwerze na podstawie adresu IP lub domeny. W tym celu możesz skorzystać z ViewDNS.info.

Polecenie `whois` uruchomi także dla adresu IP, a następnie sprawdź pole `NetRange` i zweryfikuj, czy cel ma dedykowany zakres IP. **Zakres IP** to blok adresów IP, które należą do tej samej organizacji. Jeżeli organizacja ma dedykowany zakres IP, każdy adres IP znajdujący się w tym zakresie należy do tej organizacji:

```
$ whois 157.240.2.35
NetRange:      157.240.0.0 - 157.240.255.255
CIDR:            157.240.0.0/16
NetName:        THEFA-3
NetHandle:      NET-157-240-0-0-1
Parent:         NET157 (NET-157-0-0-0-0)
NetType:        Direct Assignment
OriginAS:
Organization:   Facebook, Inc. (THEFA-3)
RegDate:        2015-05-14
Updated:        2015-05-14
Ref:            https://rdap.arin.net/registry/ip/157.240.0.0
OrgName:        Facebook, Inc.
OrgId:          THEFA-3
Address:        1601 Willow Rd.
City:           Menlo Park
StateProv:      CA
PostalCode:     94025
Country:        US
RegDate:        2004-08-11
Updated:        2012-04-17
Ref:            https://rdap.arin.net/registry/entity/THEFA-3
OrgAbuseHandle: OPERA82-ARIN
OrgAbuseName:   Operations
OrgAbusePhone:  +1-650-543-4800
OrgAbuseEmail:  noc@fb.com
OrgAbuseRef:    https://rdap.arin.net/registry/entity/OPERA82-ARIN
OrgTechHandle:  OPERA82-ARIN
OrgTechName:    Operations
OrgTechPhone:   +1-650-543-4800
OrgTechEmail:   noc@fb.com
OrgTechRef:     https://rdap.arin.net/registry/entity/OPERA82-ARIN
```

Adresy IP można również odkrywać, przeglądając się systemom autonomicznym, które są routowalnymi sieciami w ramach internetu. **Numer systemów autonomicznych** (ang. *Autonomous System Number* — ASN) identyfikują właścicieli tych sieci. Sprawdzając, czy dwa adresy IP mają wspólny ASN, można określić, czy adresy IP należą do tego samego właściciela.

Aby dowiedzieć się, czy firma posiada dedykowany zakres IP, uruchom kilka translacji IP na ASN w celu sprawdzenia, czy adresy IP są mapowane na jeden ASN. Jeśli wiele adresów w zakresie należy do tego samego ASN, organizacja może mieć dedykowany zakres adresów IP. Na podstawie poniższych danych wyjściowych możemy wywnioskować, że każdy adres IP w zakresie od 157.240.2.21 do 157.240.2.34 należy prawdopodobnie do Facebooka:

```
$ whois -h whois.cymru.com 157.240.2.20
AS      | IP           | AS Name
32934   | 157.240.2.20 | FACEBOOK, US
$ whois -h whois.cymru.com 157.240.2.27
AS      | IP           | AS Name
32934   | 157.240.2.27 | FACEBOOK, US
$ whois -h whois.cymru.com 157.240.2.35
AS      | IP           | AS Name
32934   | 157.240.2.35 | FACEBOOK, US
```

Flaga `-h` w poleceniu `whois` wskazuje, skąd serwer WHOIS ma pobierać informacje, a `whois.cymru.com` jest bazą danych, która tłumaczy adresy IP na numery ASN. Jeśli firma ma dedykowany zakres adresów IP i nie oznacza ich jako adresów leżących poza zakresem hakowania, możesz zaplanować atak na każdy adres IP w tym zakresie.

Parsowanie certyfikatu

Innym sposobem szukania hostów jest wykorzystanie certyfikatów SSL (ang. *Secure Sockets Layer*) używanych do szyfrowania ruchu internetowego. Pole `Subject Alternative Name` certyfikatu SSL umożliwia właścicielom certyfikatów określenie dodatkowych nazw hostów, które używają tego samego certyfikatu, można więc znaleźć nazwy hostów, parsując to pole. Aby znaleźć certyfikaty dla domeny, skorzystaj z internetowych baz danych, takich jak `crt.sh`, `Censys` i `Cert Spotter`.

Uruchamiając wyszukiwanie certyfikatów za pomocą `crt.sh` dla `facebook.com`, możemy np. znaleźć certyfikat SSL Facebooka. Jak widać, wymienionych jest wiele innych nazw domen należących do Facebooka:

```
X509v3 Subject Alternative Name:
DNS:*.facebook.com
DNS:*.facebook.net
DNS:*.fbcdn.net
DNS:*.fbsbx.com
DNS:*.messenger.com
```

DNS: facebook.com
DNS: messenger.com
DNS: *.m.facebook.com
DNS: *.xx.fbcdn.net
DNS: *.xy.fbcdn.net
DNS: *.xz.fbcdn.net

Witryna crt.sh ma również przydatne narzędzie, które dla ułatwienia parsowania umożliwia pobieranie informacji w formacie JSON, a nie HTML. Wystarczy do adresu URL żądania dodać parametr URL `output=json`: <https://crt.sh/?q=facebook.com&output=json>.

Enumeracja poddomen

Po znalezieniu możliwie jak największej liczby domen obiektu docelowego postaraj się zlokalizować dla nich jak najwięcej poddomen. Każda poddomena reprezentuje nowy kąt ataku na sieć. Najlepszym sposobem na wyliczenie poddomen jest użycie automatyzacji.

Narzędzia takie jak Sublist3r, SubBrute, Amass i Gobuster mogą automatycznie wyliczać poddomeny za pomocą różnych list słów i strategii. Sublist3r działa np. przez kwerendowanie wyszukiwarek i internetowych baz danych poddomen, podczas gdy SubBrute jest narzędziem brute force, które odgaduje możliwe poddomeny, dopóki nie znajdzie jakichś istniejących. Amass do wyszukiwania poddomen używa kombinacji transferów stref DNS, parsowania certyfikatów, wyszukiwarek i baz danych poddomen. Możesz zbudować narzędzie, które łączy w sobie wyniki działania wielu innych narzędzi, aby osiągnąć najlepsze rezultaty. Omówię to dalej w podrozdziale „Pisanie własnych skryptów rekonesansowych”.

Aby skorzystać z wielu narzędzi do wyliczania poddomen, musisz podać programowi listę słów, które mogą pojawić się w poddomenach. W internecie można znaleźć kilka dobrych list słów sporządzonych przez innych hakerów. Dość obszerna jest lista słów *SecLists* Daniela Miesslera, a znajdziesz ją na stronie <https://github.com/danielmiessler/SecLists/>. Możesz także użyć narzędzia do generowania list słów, takiego jak *Commonspeak2* (<https://github.com/assetnote/commonspeak2/>), aby wygenerować listy słów na podstawie najbardziej aktualnych danych internetowych. Ponadto w celu uzyskania najbardziej kompleksowych wyników możesz połączyć kilka list słów znalezionych w internecie lub wygenerowanych przez siebie. Oto proste polecenie usunięcia zduplikowanych elementów z zestawu dwóch list słów:

```
sort -u listas10w1.txt listas10w2.txt
```

Narzędzie wiersza poleceń `sort` sortuje wiersze plików tekstowych. Po podaniu wielu plików posortuje je i wypisze dane wyjściowe w terminalu. Opcja `-u` instruuje narzędzie `sort`, aby na posortowanej liście zwróciło tylko unikatowe elementy.

Gobuster to narzędzie wykorzystujące technikę brutalnej siły do odkrywania poddomen, katalogów i plików na docelowych serwerach WWW. Jego tryb DNS jest używany do brute forcingu poddomen. W tym trybie możesz użyć flagi `-d`, aby określić domenę do brute forcingu, oraz flagi `-w` w celu wskazania listy słów, którą chcesz zastosować:

```
gobuster dns -d domena_docelowa -w lista_słów
```

Po znalezieniu pewnej liczby poddomen możesz odkryć kolejne, identyfikując wzorce. Jeśli dla *example.com* znajdziesz dwie poddomeny o nazwach *1.example.com* i *3.example.com*, możesz się domyślać, że prawidłową poddomeną jest prawdopodobnie również *2.example.com*. Dobrym narzędziem do automatyzacji tego procesu jest Altdns (<https://github.com/infosec-au/altdns/>), które odkrywa poddomeny o nazwach będących permutacjami innych nazw poddomen.

Ponadto można znaleźć więcej poddomen, wykorzystując swoją wiedzę na temat stosu technologicznego firmy. Jeżeli wiesz już, że *example.com* używa Jenkinsa, możesz sprawdzić, czy prawidłową poddomeną jest *jenkins.example.com*.

Szukaj także poddomen dla poddomen. Po znalezieniu *dev.example.com* możesz znaleźć poddomeny takie jak *1.dev.example.com*. Poddomeny poddomen możesz odkrywać, uruchamiając narzędzia wyliczeniowe rekurencyjnie: dodaj wyniki pierwszego uruchomienia do listy znanych domen i uruchom narzędzie ponownie.

Enumeracja usług

Następnym krokiem jest wyliczenie usług hostowanych na znalezionych komputerach. Ponieważ usługi często działają na portach domyślnych, dobrym sposobem na ich wyszukiwanie jest skanowanie portów maszyny przy użyciu skanowania aktywnego lub pasywnego.

W **skanowaniu aktywnym** nawiązujesz połączenie bezpośrednio z serwerem. Aktywne narzędzia skanujące wysyłają żądania połączenia z portami maszyny docelowej, by szukać otwartych portów. Do aktywnego skanowania można używać narzędzi takich jak Nmap lub Masscan. Poniższe proste polecenie narzędzia Nmap ujawnia otwarte porty dla *scanme.nmap.org*:

```
$ nmap scanme.nmap.org
```

```
Nmap scan report for scanme.nmap.org (45.33.32.156)
```

```
Host is up (0.086s latency).
```

```
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
```

```
Not shown: 993 closed ports
```

```
PORT STATE SERVICE
```

```
22/tcp open  ssh
```

```
25/tcp filtered smtp
```

```
80/tcp open  http
```

```
135/tcp filtered msrpc
```

```
445/tcp filtered microsoft-ds
```

9929/tcp open nping-echo
31337/tcp open Elite
Nmap done: 1 IP address (1 host up) scanned in 230.83 seconds

Natomiast w **skanowaniu pasywnym** używa się zasobów zewnętrznych, aby pozyskać informacje o portach maszyny bez interakcji z serwerem. Skanowanie pasywne jest cichsze i pomaga atakującym uniknąć wykrycia. Do szukania usług na maszynie bez aktywnego skanowania możesz użyć wyszukiwarki **Shodan**, która pozwala użytkownikowi znaleźć maszyny podłączone do internetu.

Dzięki Shodanowi możesz wykryć obecność kamer internetowych, serwerów WWW, a nawet elektrowni na podstawie kryteriów takich jak nazwy hostów lub adresy IP. Jeśli uruchomisz wyszukiwanie Shodana dla adresu IP strony *scanme.nmap.org*, 45.33.32.156, otrzymasz wynik pokazany na rysunku 5.1. Widać, że wyszukiwanie daje inne rezultaty niż nasze skanowanie portów i dostarcza dodatkowych informacji o serwerze.

45.33.32.156 scanme.nmap.org

City: Fremont
Country: United States
Organization: Linode
ISP: Linode
Last Update: 2020-06-27T14:46:56.978007
Hostnames: scanme.nmap.org
ASN: AS63949

Ports
22 80 123

Services
22
tcp
ssh
OpenSSH Version: 6.6.1p1 Ubuntu-2ubuntu2.13
SSH-2.0-OpenSSH_6.6.1p1_Ubuntu-2ubuntu2.13
Key type: ssh-rsa
Key: AAAAB3NzaC1yc2EAAAADAQABAAQCA6aFootZ9mVUGFNEhM0RR1Bt:zu64X0wE1hCshw/zV1I
Xf
HKYINb09+11dn2VgJ021pxk0s+L6+EbyY0nVRURT-rMTgHL8xseB8E8kNqexs9HYZ5iq1Mx4j1tGNt
HvshfxZnbxvVUK2dasWvt8kn8J5Jag5bzWf0o4hJKM0115U1LXtLkAs2FbWliq2Ed5uKw/Nk86f1p
1TA+8cc6EAtnsVptT340/BHhAksR0kQzDowQvNBz2/BecEvoMScarf+kDfN0wK3gENt550qW9J
L0za6Y3BPL/zfUQ08nJ74R:FsV,444nT1rG19Jc2x88V786hASkUkKmhYfWb0k88yGd
FingerPrint: 28b362d14418212a1985a79d185183485134c21a61b2

Vulnerabilities
Note: the device may not be impacted by all of these issues. The vulnerabilities are implied based on the software and version.
CVE-2014-0117 The mod_proxy module in the Apache HTTP Server 2.4.x before 2.4.10, when a reverse proxy is enabled, allows remote attackers to cause a denial of service (child-process crash) via a crafted HTTP Connection header.

Rysunek 5.1. Strona wyników Shodana dla *scanme.nmap.org*

Alternatywami dla Shodana są m.in. Censys i Project Sonar. Aby uzyskać najlepsze wyniki, połącz informacje zebrane z różnych baz danych. Dzięki nim możesz również znaleźć adresy IP, certyfikaty i wersje oprogramowania obiektu docelowego.

Brute forcing katalogów

Aby odkryć jeszcze większą powierzchnię ataku na daną witrynę, możesz przeprowadzić brute forcing katalogów znalezionych serwerów WWW. Wyszukiwanie katalogów na serwerach ma dużą wartość, ponieważ dzięki nim możesz odkryć ukryte panele administracyjne, pliki konfiguracyjne, pliki haseł, przestarzałe funkcje, kopie baz danych i pliki kodu źródłowego. Brute forcing pozwala czasem na bezpośrednie przejście serwera!

Nawet jeśli nie możesz znaleźć żadnych gotowych exploitów, informacje katalogowe często mówią o strukturze i technologii aplikacji. Ścieżka zawierająca np. *phpmyadmin* zazwyczaj oznacza, że aplikacja jest zbudowana przy użyciu PHP.

Do brute forcingu katalogów możesz wykorzystać Dirsearch lub Gobuster. Narzędzia te stosują listy słów do budowania adresów URL, a następnie żądają tych adresów URL z serwera internetowego. Jeżeli serwer odpowie kodem statusu z zakresu 200, istnieje jakiś katalog lub plik. Oznacza to, że można otworzyć daną stronę i zobaczyć, co aplikacja tam hostuje. Kod statusu 404 oznacza, że katalog lub plik nie istnieje, natomiast 403 oznacza, że katalog lub plik istnieje, ale jest chroniony. Zbadaj dokładnie strony 403, aby sprawdzić, czy możesz ominąć ochronę w celu uzyskania dostępu do zawartości.

Poniżej pokazałam przykład uruchomienia polecenia `dirsearch`. Flaga `-u` określa nazwę hosta, a flaga `-e` rozszerzenie pliku, który ma zostać użyty podczas tworzenia adresów URL:

```
$ ./dirsearch.py -u scanme.nmap.org -e php
Extensions: php | HTTP method: get | Threads: 10 | Wordlist size: 6023
Error Log: /tools/dirsearch/logs/errors.log
Target: scanme.nmap.org
[12:31:11] Starting:
[12:31:13] 403 - 290B - /.htusers
[12:31:15] 301 - 316B - /.svn -> http://scanme.nmap.org/.svn/
[12:31:15] 403 - 287B - /.svn/
[12:31:15] 403 - 298B - /.svn/all-wcprops
[12:31:15] 403 - 294B - /.svn/entries
[12:31:15] 403 - 297B - /.svn/prop-base/
[12:31:15] 403 - 296B - /.svn/pristine/
[12:31:15] 403 - 291B - /.svn/tmp/
[12:31:15] 403 - 315B - /.svn/text-base/index.php.svn-base
[12:31:15] 403 - 293B - /.svn/props/
[12:31:15] 403 - 297B - /.svn/text-base/
[12:31:40] 301 - 318B - /images -> http://scanme.nmap.org/images/
[12:31:40] 200 - 7KB - /index
[12:31:40] 200 - 7KB - /index.html
[12:31:53] 403 - 295B - /server-status
[12:31:53] 403 - 296B - /server-status/
[12:31:54] 301 - 318B - /shared -> http://scanme.nmap.org/shared/
Task Completed
```

Tryb `dir` Gobustera służy do wyszukiwania dodatkowych zawartości w określonej domenie lub poddomenie. Obejmuje to ukryte katalogi i pliki. W tym trybie możesz użyć flagi `-u`, by określić domenę lub subdomenę do brute forcingu, oraz flagi `-w` celu określenia listy słów, której chcesz użyć:

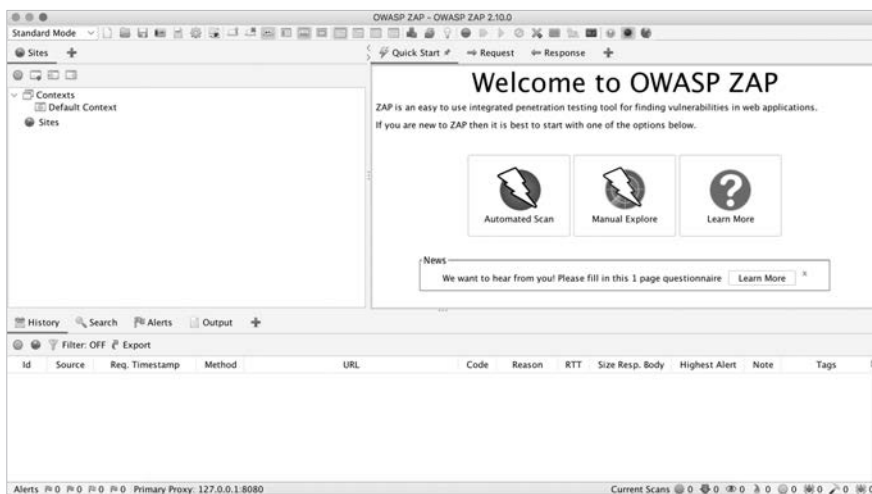
```
gobuster dir -u url_celu -w lista_slow
```

Ręczne odwiedzanie wszystkich stron znalezionych z wykorzystaniem techniki brutalnej siły może być czasochłonne. Zamiast tego możesz używać narzędzia do wykonywania zrzutów ekranu, takiego jak EyeWitness (<https://github.com/FortyNorthSecurity/EyeWitness/>) lub Snapper (<https://github.com/dxa4481/Snapper/>), aby automatycznie weryfikować, czy w poszczególnych lokalizacjach hostowana jest jakaś strona. EyeWitness przyjmuje listę adresów URL i wykonuje zrzuty ekranowe każdej strony. Możesz je szybko przejrzeć w aplikacji galerii zdjęć w celu znalezienia tych interesujących. Zwracaj uwagę na ukryte usługi, takie jak panele programistyczne lub administracyjne, strony z listami katalogów, strony analityczne i strony, które wyglądają na przestarzałe i zaniedbane. Zwykle w takich miejscach ujawniają się luki w zabezpieczeniach.

Indeksowanie witryn

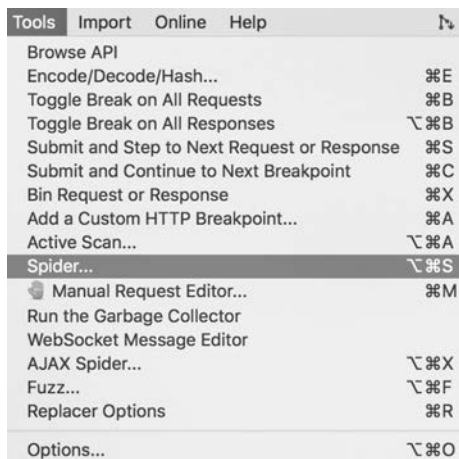
Kolejnym sposobem wykrywania katalogów i ścieżek jest **skanowanie internetu** (ang. *web spidering*), inaczej **indeksowanie stron**, czyli proces używany do identyfikacji wszystkich stron w danej witrynie. Narzędzie pająka (robota) internetowego rozpoczyna działanie od wskazanej strony. Następnie identyfikuje wszystkie adresy URL osadzone na tej stronie i odwiedza je. Przeszukując rekurencyjnie wszystkie adresy URL znajdujące się na wszystkich stronach witryny, pająk internetowy może wyszperać wiele ukrytych punktów końcowych w aplikacji.

Wbudowany pająk internetowy, z którego możesz skorzystać, oferuje narzędzie OWASP ZAP (ang. *Zed Attack Proxy*) dostępne na stronie <https://www.zaproxy.org/> (zobacz rysunek 5.2). Jest to narzędzie zabezpieczeń dystrybuowane na licencji open source, zawiera skaner, serwer proxy i wiele innych funkcjonalności. Burp Suite ma równoważne narzędzie zwane **crawlerem**, ale ja wolę pająka ZAP-a.



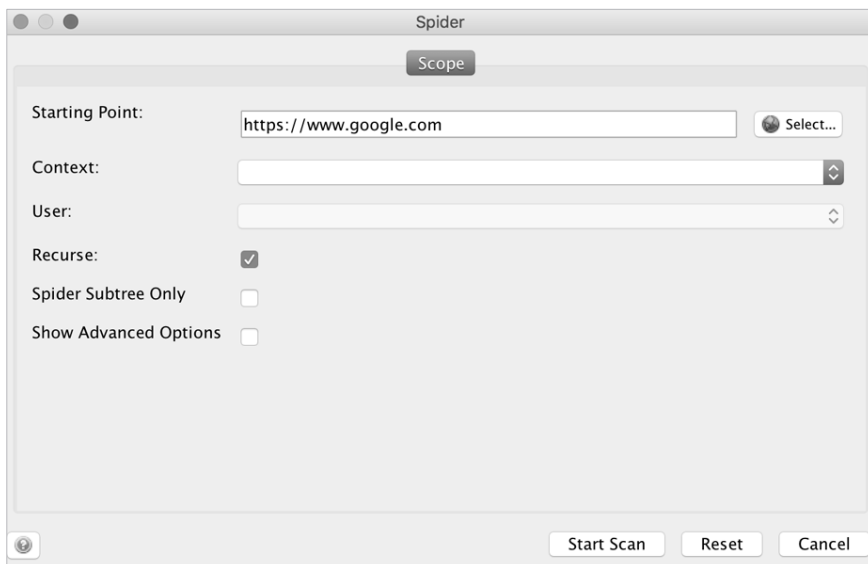
Rysunek 5.2. Strona startowa narzędzia OWASP ZAP

Aby uzyskać dostęp do pająka, otwórz ZAP i wybierz *Tools/Spider*, jak pokazałam na rysunku 5.3.



Rysunek 5.3. Narzędzie pająka znajdziesz w menu *Tools/Spider*

Po uruchomieniu pająka powinno zostać wyświetlone okno umożliwiające określenie początkowego adresu URL (zobacz rysunek 5.4).



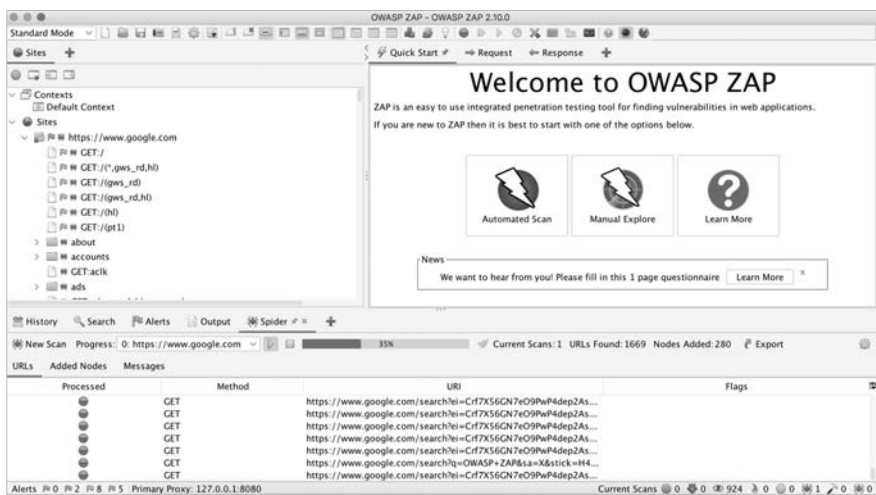
Rysunek 5.4. W tym oknie możesz określić docelowy adres URL do skanowania

Kliknij przycisk *Start Scan*. W dolnym panelu powinny zacząć pojawiać się adresy URL, jak pokazałam na rysunku 5.5.

Processed	Method	URI
●	GET	https://www.google.com/shopping/ratings/account/metrics
●	GET	https://www.google.com/shopping/reviewer
●	GET	https://www.google.com/shopping/seller
●	GET	https://www.google.com/about/careers/applications
●	GET	https://www.google.com/landing/signout.html
●	GET	https://www.google.com/in

Rysunek 5.5. Wyniki skanowania są wyświetlane w dolnym panelu okna narzędzia OWASP ZAP

Po lewej stronie okna ZAP-a powinno również wyświetlić się drzewo witryny (zobacz rysunek 5.6). Pokazuje ono w zorganizowany sposób pliki i katalogi znalezione na serwerze docelowym.



Rysunek 5.6. Drzewo witryny w lewym oknie pokazuje pliki i katalogi znalezione na serwerze docelowym

Hostowanie zewnętrzne

Przjrzyj się cyfrowemu śladowi zewnętrznych usług hostingowych świadczonych na rzecz danej firmy. Poszukaj np. wiaderek S3 wynajmowanych przez tę organizację. S3 (ang. *Simple Storage Service*) to produkt firmy Amazon przeznaczony do przechowywania danych online. Organizacje mogą płacić za przechowywanie zasobów w **wiaderkach** (ang. *bucket*), aby obsłużyć swoje aplikacje internetowe, lub mogą używać wiaderka S3 jako lokalizacji kopii zapasowych lub pamięci masowej. Jeśli organizacja korzysta z usługi S3 Amazona, jej wiaderka S3 mogą zawierać ukryte punkty końcowe, dzienniki, poświadczenia, informacje o użytkownikach, kod źródłowy i inne informacje, które mogą okazać się przydatne.

Jak znaleźć wiaderka organizacji? Jednym ze sposobów jest Google dorking, o którym pisałam wcześniej. Większość wiaderka używa formatu URL **WIADERKO.s3.amazonaws.com** lub **s3.amazonaws.com/WIADERKO**, trafić mogą więc dostarczyć następujące hasła wyszukiwania:

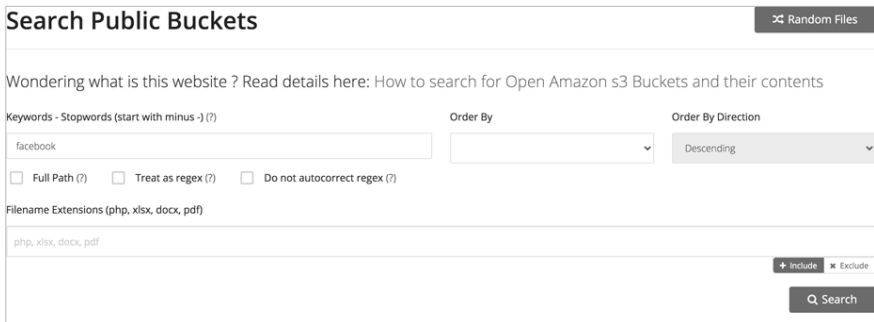
```
site:s3.amazonaws.com NAZWA_FIRMY
site:amazonaws.com NAZWA_FIRMY
```

Jeśli firma używa dla swoich wiaderek S3 niestandardowych adresów URL, spróbuj użyć bardziej elastycznych haseł wyszukiwania. Firmy często umieszczają w swoich niestandardowych adresach URL wiaderek słowa kluczowe, takie jak *aws* i *s3*, dlatego spróbuj następujących wyszukiwań:

```
amazonaws s3 NAZWA_FIRMY
amazonaws bucket NAZWA_FIRMY
amazonaws NAZWA_FIRMY
s3 NAZWA_FIRMY
```

Kolejnym sposobem na szukanie wiaderek jest przeszukiwanie publicznych repozytoriów GitHuba danej firmy pod kątem adresów URL usługi S3. W tym celu spróbuj wyszukać w tych repozytoriach hasło *s3*. Użycie GitHuba do rekonesansu omówię w następnym punkcie.

Aby znaleźć publicznie dostępne wiaderka S3, możesz skorzystać z wyszukiwarki online **GrayhatWarfare** (<https://buckets.grayhatwarfare.com/>), którą pokazałam na rysunku 5.7. Umożliwia ona wyszukanie wiaderek za pomocą słów kluczowych. Podając związane z Twoim celem słowa kluczowe, takie jak aplikacja, projekt lub nazwa organizacji, możesz znaleźć odpowiednie wiaderka.



The screenshot shows the 'Search Public Buckets' interface. At the top right is a 'Random Files' button. Below it is a text input field with the placeholder 'Wondering what is this website? Read details here: How to search for Open Amazon s3 Buckets and their contents'. There are three dropdown menus: 'Keywords - Stopwords (start with minus -) (?)' containing 'facebook', 'Order By' (empty), and 'Order By Direction' set to 'Descending'. Below these are three checkboxes: 'Full Path (?)', 'Treat as regex (?)', and 'Do not autocorrect regex (?)'. A 'Filename Extensions (php, xlsx, docx, pdf)' section contains a text input with 'php, xlsx, docx, pdf'. At the bottom right are '+ Include' and 'X Exclude' buttons, and a 'Search' button.

Rysunek 5.7. Strona główna wyszukiwarki GrayhatWarfare

Na koniec możesz spróbować brute forcing wiaderek i użyć słów kluczowych. Pomoże Ci w tym narzędzie Lazys3 (<https://github.com/naHamsec/lazys3/>). Bazując na liście słów, próbuje ono odgadnąć wiaderka o nazwach będących permutacjami popularnych nazw wiaderek. Innym dobrym narzędziem jest **Bucket Stream** (<https://github.com/eth0izzle/bucket-stream/>), które parsuje certyfikaty należące do organizacji i wyszukuje wiaderka S3 na podstawie permutacji nazw domenowych znalezionych w certyfikatach. Bucket Stream automatycznie sprawdza również, czy dane wiaderko jest dostępne, dzięki czemu oszczędza Ci czasu.

Po znalezieniu kilku wiaderek należących do organizacji docelowej użyj narzędzia wiersza poleceń `aws`, by sprawdzić, czy możesz uzyskać do nich dostęp. Narzędzie to możesz zainstalować za pomocą następującego polecenia:

```
pip install awscli
```

Następnie musisz skonfigurować je do współpracy z AWS-em, postępując zgodnie z dokumentacją Amazona, którą znajdziesz na stronie <https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html>. Po wykonaniu tych czynności powinieneś mieć dostęp do wiaderek bezpośrednio z terminala za pomocą polecenia `aws s3`. Spróbuj wyświetlić zawartość znalezionej wiaderek:

```
aws s3 ls s3://NAZWA_WIADERKA/
```

Jeżeli to zadziała, sprawdź, czy możesz odczytać zawartość interesujących Cię plików, kopiując je na swoją lokalną maszynę:

```
aws s3 cp s3://NAZWA_WIADERKA/NAZWA_PLIKU/ścieżka/do/lokalnego/katalogu
```

Zbierz wszelkie przydatne informacje, które wyciekły z wiaderek i zachowaj je do celów przyszłej eksploatacji! Jeśli organizacja ujawnia informacje, takie jak aktywne klucze API lub dane osobowe, należy to natychmiast zgłosić. Publicznie dostępne wiaderek S3 są często uważane za lukę w zabezpieczeniach. Możesz także spróbować przesłać nowe pliki do wiaderek lub usunąć z niego pliki. Jeżeli nie uda Ci się zmienić jego zawartości, być może będziesz w stanie manipulować operacjami aplikacji internetowej lub uszkodzić dane firmy. Poniższe polecenie kopiuje plik lokalny o nazwie `PLIK_TESTOWY` do wiaderek S3 obiektu docelowego:

```
aws s3 cp PLIK_TESTOWY s3://NAZWA_WIADERKA/
```

Następne polecenie usunie plik `PLIK_TESTOWY`, który właśnie przesłałeś:

```
aws s3 rm s3://NAZWA_WIADERKA/PLIK_TESTOWY
```

Polecenia są nieszkodliwym sposobem na udowodnienie uzyskania dostępu do wiaderek bez ingerowania w pliki firmy docelowej.

Zawsze przesyłaj i usuwaj własne pliki testowe. Nie ryzykuj usunięcia podczas testów ważnych zasobów firmy, chyba że nie lękasz się kosztownego pozwu.

Rekonesans z wykorzystaniem GitHuba

Przeszukaj repozytoria GitHuba docelowej organizacji pod kątem przypadkowo zatwierdzonych w nim poufnych danych lub informacji, które mogą prowadzić do wykrycia luki w zabezpieczeniach.

Zacznij od szukania nazw użytkowników GitHuba związanych z Twoim obiektem docelowym. Powinieneś móc je zlokalizować, wyszukując w pasku wyszukiwania GitHuba nazwę organizacji lub nazwy jej produktów albo sprawdzając konta GitHuba znanych pracowników.

Gdy znajdziesz nazwy użytkowników do skontrolowania, odwiedź ich strony. Znajdź repozytoria związane z testowanymi przez Ciebie projektami i zapisz je razem z nazwami użytkowników głównych kontrybutorów aplikacji, które mogą pomóc w znalezieniu bardziej odpowiednich repozytoriów.

W następnej kolejności przeanalizuj kod. W przypadku każdego repozytorium zwróć szczególną uwagę na sekcje *Issues* i *Commits*. Są one pełne potencjalnych wycieków informacji: mogą wskazać atakującym pozostawione błędy, problematyczny kod, najnowsze poprawki kodu i poprawki zabezpieczeń. Ostatnie zmiany w kodzie, które nie przetrwały próby czasu, nierzadko zawierają błędy. Przyjrzyj się wszelkim zaimplementowanym mechanizmom ochrony, by sprawdzić, czy możesz je ominąć. Możesz również przeszukać sekcję *Code* pod kątem fragmentów kodu, które mogą być podatne na ataki. Po znalezieniu interesującego Cię pliku sprawdź sekcje *Blame* i *History* w prawym górnym rogu strony pliku, aby zweryfikować, jak powstał (zobacz rysunek 5.8).



Rysunek 5.8. Sekcje *Blame* i *History*

Przeoglądaniem kodu źródłowego zajmiemy się w rozdziale 22., ale podczas fazy rekonesansu poszukaj zakodowanych na sztywno sekretów, takich jak klucze API, klucze szyfrowania i hasła do baz danych. Przeszukaj repozytoria organizacji pod kątem terminów takich jak *key*, *secret* i *password*, aby znaleźć zakodowane na sztywno poświadczenia użytkowników, które możesz wykorzystać do uzyskania dostępu do wewnętrznych systemów. Po znalezieniu wyciekających poświadczeń możesz zastosować narzędzie *KeyHacks* (<https://github.com/streaak/keyhacks/>) w celu sprawdzenia, czy poświadczenia są prawidłowe. Jeśli tak, postaraj się dowiedzieć, jak używać ich w celu uzyskiwania dostępu do usług obiektu docelowego.

Powinieneś również wyszukać w projekcie wrażliwe funkcjonalności. Sprawdź, czy którykolwiek z kodów źródłowych dotyczy ważnych funkcji, takich jak uwierzytelnianie, resetowanie hasła, działania zmieniające stan lub odczytywanie

prywatnych informacji. Zwróć uwagę na kod obsługujący wprowadzanie przez użytkownika danych, takich jak parametry żądania HTTP, nagłówki HTTP, ścieżki żądań HTTP, wpisy w bazie danych, odczyty plików i przesyłanie plików, ponieważ zapewniają atakującym potencjalne punkty wejścia w celu eksploatacji luk w zabezpieczeniach aplikacji. Poszukaj plików konfiguracyjnych, gdyż pozwalają one zebrać więcej informacji o infrastrukturze. Ponadto wyszukaj stare punkty końcowe i adresy URL wiaderek S3, które można atakować. Spisz pliki do dalszego przeglądu w przyszłości.

Ogromnym źródłem błędów są także przestarzałe zależności i niekontrolowane korzystanie z niebezpiecznych funkcji. Zwróć uwagę na używane zależności i importy i przejrzyj listę wersji, by sprawdzić, czy są może nieaktualne. Rejestruj wszelkie przestarzałe zależności. Informacje te możesz wykorzystać później w celu wyszukania ujawnionych podatności na ataki, które mogłyby zadziałać na obiekt docelowy.

Proces przeprowadzania rekonesansu na GitHubie możesz zautomatyzować za pomocą narzędzi takich jak Gitrob i TruffleHog. Gitrob (<https://github.com/michenriksen/gitrob/>) lokalizuje potencjalnie poufne pliki przesyłane do publicznych repozytoriów GitHuba. TruffleHog (<https://github.com/trufflesecurity/truffleHog/>) specjalizuje się w wyszukiwaniu w repozytoriach sekretów poprzez wyszukiwanie na podstawie wyrażań regularnych i skanowanie w poszukiwaniu łańcuchów znaków o wysokiej entropii.

Inne podstępne techniki białego wywiadu

Wiele z omówionych przeze mnie dotychczas strategii to przykłady **białego wywiadu** (ang. *Open Source Intelligence* — OSINT), czyli praktyki pozyskiwania informacji z publicznych źródeł. W tym podrozdziale skoncentruję się na jego innych źródłach, których możesz użyć do pozyskania cennych informacji.

Najpierw sprawdź publikowane przez firmę oferty pracy pod kątem stanowisk inżynierskich. Oferty dotyczące stanowisk inżynierskich często ujawniają wykorzystywane przez firmę technologie. Przyjrzyjmy się np. takiej ofercie pracy:

Inżynier full stack

Minimalne kwalifikacje:

- biegła znajomość języków programowania Python i C/C++;
- doświadczenie w pracy z systemem Linux;
- znajomość frameworków Flask, Django i Node.js;
- znajomość usług Amazon Web Services, zwłaszcza EC2, ECS, S3 i RDS.

Po przeczytaniu tego ogłoszenia wiesz, że firma do budowania aplikacji internetowych używa frameworków Flask, Django i Node.js. Inżynierowie prawdopodobnie korzystają również z Pythona, C i C++ na backendzie z maszyną linuxową. Wykorzystują ponadto AWS do outsourcingu swoich operacji i przechowywania plików.

Jeśli nie możesz znaleźć ofert pracy dotyczących odpowiednich stanowisk, wyszukaj profile pracowników na LinkedInie i przeczytaj ich osobiste blogi lub zapytania z dziedziny inżynierii opublikowane na forach, takich jak Stack Overflow i Quora. Wiedza fachowa najlepszych pracowników firmy często odzwierciedla technologię wykorzystywaną w rozwoju oprogramowania.

Kolejnym źródłem informacji są kalendarze Google'a pracowników. Kalendarze robocze często zawierają notatki ze spotkań, slajdy, a czasami nawet dane logowania. Jeśli pracownik przypadkowo udostępni publicznie swoje kalendarze, możesz uzyskać do nich dostęp. Cenne informacje mogą wyciekać także na profilach społecznościowych organizacji lub jej pracowników. Hakerom zdarzało się np. odkrywać zestawy ważnych poświadczeń na karteczkach samoprzylepnych widocznych w tle na selfie robionych w biurze!

Jeśli firma posiada listę mailingową inżynierów, zapisz się, aby uzyskać wgląd w technologię i proces rozwoju oprogramowania firmy. Sprawdź również konta SlideShare'a lub Pastebina firmy. Gdy organizacje występują na konferencjach lub odbywają wewnętrzne spotkania, przesyłają czasami slajdy na SlideShare w celach informacyjnych. Być może uda Ci się znaleźć informacje na temat stosu technologicznego i związanych z bezpieczeństwem wyzwań, z którymi mierzy się firma.

Pastebin (<https://pastebin.com/>) to strona internetowa służąca do wklejania i krótkookresowego przechowywania tekstu online. Jest wykorzystywana do przesyłania tekstu między komputerami lub użytkownikami. Inżynierowie czasami używają jej do udostępniania współpracownikom kodu źródłowego lub dzienników serwera przeglądania w celu prowadzenia współpracy, może więc być doskonałym źródłem informacji. Możesz znaleźć na niej także przesłane poświadczenia i komentarze programistyczne. Wejdź na Pastebin, wyszukaj nazwę organizacji docelowej i zobacz, co się stanie! Do wyszukiwania danych wklejonych w publicznie dostępnych lokalizacjach możesz ponadto zastosować zautomatyzowane narzędzia, takie jak PasteHunter (<https://github.com/kevthehermit/PasteHunter/>).

Zapoznaj się ponadto z archiwizacyjnymi stronami internetowymi, takimi jak Wayback Machine (<https://archive.org/web/>), które stanowią cyfrowy zapis treści internetowych (zobacz rysunek 5.9). Wayback Machine rejestruje zawartość witryny w różnych momentach. Za jego pomocą można znaleźć stare punkty końcowe, listy katalogów, zapomniane poddomeny oraz przestarzałe, ale wciąż będące w użyciu adresy URL i pliki. Narzędzie Waybackurls (<https://github.com/tomnomnom/waybackurls/>) udostępnione przez użytkownika tomnomnom pozwala automatycznie wyodrębnić punkty końcowe i adresy URL z Waybacka.

Fingerprinting stosu technologicznego

Techniki fingerprintingu mogą pomóc Ci w jeszcze lepszym zrozumieniu docelowej aplikacji. **Fingerprinting** polega na identyfikacji producentów i wersji oprogramowania, z których korzysta maszyna lub aplikacja. Informacje te umożliwiają przeprowadzanie ukierunkowanych ataków na aplikację, ponieważ możesz



Rysunek 5.9. Wayback Machine archiwizuje zawartość internetową i pozwala zobaczyć strony, które zostały usunięte przez administratorów danej witryny

szukać wszelkich znanych błędnych konfiguracji i ujawnionych luk związanych z konkretną wersją. Jeśli wiesz, że serwer używa np. starej wersji serwera Apache, na którą może mieć wpływ ujawniona luka w zabezpieczeniach, możesz ją wykorzystać do podjęcia natychmiastowej próby zaatakowania serwera.

Spółeczność bezpieczeństwa klasyfikuje znane luki w zabezpieczeniach jako **CVE** (ang. *Common Vulnerabilities and Exposures*) i każdej CVE nadaje numer referencyjny. Możesz wyszukiwać je w bazie danych CVE (https://cve.mitre.org/cve/search_cve_list.html).

Najprostszym sposobem fingerprintingu aplikacji jest wejście z nią w bezpośrednią interakcję. Najpierw uruchom na swojej maszynie Nmap z flagą `-sV`, by umożliwić wykrywanie wersji podczas skanowania portów. W poniższym listingu możesz zobaczyć, że Nmap spróbował przeprowadzić fingerprinting oprogramowania działającego na docelowym hoście:

```
$ nmap scanme.nmap.org -sV
Starting Nmap 7.60 ( https://nmap.org )
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.065s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 992 closed ports
PORT STATE SERVICE VERSION
22/tcp open  ssh      OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
25/tcp filtered  smtp
80/tcp open  http     Apache httpd 2.4.7 ((Ubuntu))
135/tcp filtered  msrpc
139/tcp filtered  netbios-ssn
445/tcp filtered  microsoft-ssn
9929/tcp open  nping-echo Nping echo
31337/tcp open  tcpwrapped
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/.
Nmap done: 1 IP address (1 host up) scanned in 9.19 seconds
```

Następnie w Burpie wyślij żądanie HTTP do serwera, aby sprawdzić użyte nagłówki HTTP w celu uzyskania wglądu w stos technologiczny. Z serwera może wyciekać wiele informacji przydatnych do fingerprintingu jego technologii:

```
Server: Apache/2.0.6 (Ubuntu)
X-Powered-By: PHP/5.0.1
X-Generator: Drupal 8
X-Drupal-Dynamic-Cache: UNCACHEABLE
Set-Cookie: PHPSESSID=abcde;
```

Nagłówki HTTP, takie jak `Server` i `X-Powered-By`, są dobrymi wskaźnikami stosowanych technologii. Nagłówek `Server` często ujawnia wersje oprogramowania uruchomionego na serwerze, a `X-Powered-By` pokazuje język serwera lub skryptów. Ponadto niektóre nagłówki są wykorzystywane tylko przez określone technologie, np. nagłówków `X-Generator` i `X-Drupal-Dynamic-Cache` używa tylko Drupal. Wskazówek dostarczają również charakterystyczne dla technologii pliki *cookie*, takie jak `PHPSESSID`; jeżeli serwer odsyła plik *cookie* o nazwie `PHPSESSID`, prawdopodobnie jest on tworzony przy użyciu języka PHP.

Podpowiedzi zawiera także kod źródłowy HTML-a stron internetowych. Wiele frameworków WWW lub innych technologii osadza w kodzie źródłowym swoją sygnaturę. Kliknij stronę prawym przyciskiem myszy, wybierz z menu kontekstowego opcję *Pokaż źródło strony* i naciśnij `Ctrl+F`, by wyszukać wyrażenia, takie jak *powered by*, *built with* i *running*; możesz znaleźć np. `Powered by: WordPress 3.3.2`.

Sprawdź charakterystyczne dla technologii rozszerzenia plików, nazwy plików, foldery i katalogi, np. plik o nazwie *phpmyadmin* znajdujący się w katalogu głównym takim jak `https://example.com/phpmyadmin` oznacza, że aplikacja wykorzystuje PHP. Katalog o nazwie *jinja2*, który zawiera szablony, wskazuje, że strona używa prawdopodobnie frameworków Django i Jinja2. Więcej informacji na temat sygnatur systemów plików konkretnych technologii znajdziesz w ich dokumentacji.

Proces ten może zautomatyzować kilka aplikacji. **Wappalyzer** (`https://www.wappalyzer.com/`) to rozszerzenie przeglądarki, które identyfikuje systemy zarządzania zawartością, frameworki i języki programowania wykorzystywane w witrynie. **BuiltWith** (`https://builtwith.com/`) to strona internetowa pokazująca, z jakich technologii internetowych zbudowana jest dana witryna. **StackShare** (`https://stackshare.io/`) to platforma internetowa umożliwiająca programistom udostępnianie technologii, z której korzystają. Możesz jej użyć, by dowiedzieć się, czy programiści organizacji opublikowali swój stos technologiczny. Wreszcie **Retire.js** jest narzędziem wykrywającym przestarzałe biblioteki JavaScriptu i pakiety Node.js. Można je stosować do sprawdzania przestarzałych technologii na stronie.

Pisanie własnych skryptów rekonesansowych

Pewnie zdążyłeś się już zorientować, że dobry rekonesans to rozległy proces. Nie musi być jednak czasochłonny lub trudny do zarządzania. Omówiłam już kilka narzędzi, które wykorzystują moc automatyzacji, by ułatwić ten proces.

Czasami przydatne może się okazać pisanie własnych skryptów. **Skrypt** to lista poleceń przeznaczonych do wykonania przez program. Skrypty służą do automatyzacji zadań, takich jak analiza danych, generowanie stron internetowych i administrowanie systemami. Dla nas, tropicieli bug bounty, uruchamianie skryptów jest sposobem na szybkie i efektywne przeprowadzanie rekonesansów, testów i eksploatacji. Mógłbyś napisać np. skrypt do skanowania obiektu docelowego pod kątem nowych poddomen lub wyliczania potencjalnie wrażliwych plików i katalogów na serwerze. Kiedy już nauczysz się pisać skrypty, możliwości są nieograniczone.

W tym podrozdziale skupię się w szczególności na skryptach powłoki bash — omówię, czym one są i dlaczego warto z nich korzystać. Dowiesz się, jak wykorzystywać bash, aby uprościć proces rekonesansu, a nawet pisać własne narzędzia. Zakładam, że masz podstawową wiedzę na temat działania języków programowania, w tym zmiennych, warunków, pętli i funkcji, dlatego jeśli nie znasz tych koncepcji, zapoznaj się ze wstępem do zajęć z kodowania w Codecademy (<https://www.codecademy.com/>) lub przeczytaj jakąś pozycję dotyczącą programowania.

Skrypty bashowe oraz wszelkiego rodzaju skrypty powłoki są przydatne do zarządzania złożonością i do automatyzacji powtarzających się zadań. Jeśli polecenia zawierają wiele parametrów wejściowych lub dane wejściowe jednego polecenia zależą od danych wyjściowych drugiego, wprowadzenie tego ręcznie szybko może się skomplikować i zwiększyć ryzyko popełnienia błędu w programowaniu. Z drugiej strony możesz mieć listę poleceń, które będziesz chciał wykonywać wiele razy. Skrypty są wtedy przydatne, gdyż oszczędzają kłopotu z wielokrotnym wpisywaniem tych samych poleceń. Wystarczy za każdym razem uruchamiać skrypt i gotowe.

Podstawy pisania skryptów bashowych

Napiszmy pierwszy skrypt. Otwórz dowolny edytor tekstu i wykonuj opisane czynności. Pierwszą linią każdego skryptu powłoki powinna być **linia shebang**. Zaczyna się ona od znaku kratki (#) i wykrzyknika (!) — deklaruje, jaki interpreter ma być stosowany do skryptu. Dzięki temu zwykły plik tekstowy może być wykonywany jak plik binarny. Użyjemy linii shebang do wskazania, że korzystamy z powłoki bash.

Przyjmijmy, że chcemy napisać skrypt, który wykonuje dwa polecenia; powinien uruchamiać dla obiektu docelowego narzędzie Nmap, a następnie Dirsearch. Możemy umieścić te polecenia w skrypcie w następujący sposób:

```
#!/bin/bash
nmap scanme.nmap.org
/ŚCIEŻKA/DO/dirsearch.py -u scanme.nmap.org -e php
```

Ten skrypt nie jest zbyt użyteczny; może skanować tylko jedną stronę: *scanme.nmap.org*. Zamiast tego powinniśmy pozwolić użytkownikom podawać skryptowi argumenty wejściowe, aby mogli wybrać witrynę do skanowania. W składni powłoki `$1` reprezentuje pierwszy przekazywany argument, `$2` drugi argument itd. Ponadto `$@` reprezentuje wszystkie przekazywane argumenty, natomiast `$#` reprezentuje całkowitą liczbę argumentów. Pozwólmy użytkownikom określać swoje cele za pomocą pierwszego argumentu wejściowego przypisanego do zmiennej `$1`:

```
#!/bin/bash
nmap $1
/ŚCIEŻKA/DO/dirsearch.py -u $1 -e php
```

Teraz polecenia te będą wykonywane dla dowolnej domeny przekazanej przez użytkownika jako pierwszy argument.

Zwróć uwagę, że trzecia linia skryptu zawiera ciąg `/ŚCIEŻKA/DO/dirsearch.py`. Segment `/ŚCIEŻKA/DO/` należy zastąpić bezwzględną ścieżką do katalogu, w którym przechowywany jest skrypt Dirsearcha. Jeżeli nie określisz jego lokalizacji, komputer spróbuje wyszukać go w bieżącym katalogu, a jeśli plik Dirsearcha nie został zapisany w tym samym katalogu co skrypt powłoki, `bash go` nie znajdzie.

Innym sposobem upewnienia się, że skrypt będzie mógł znaleźć użyte polecenia, jest zastosowanie zmiennej środowiskowej `PATH`, w systemach uniksowych określającej, gdzie znajdują się wykonywalne pliki binarne. Jeśli wykonasz poniższe polecenie, które spowoduje dodanie katalogu Dirsearcha do Twojej zmiennej `PATH`, będziesz mógł uruchamiać to narzędzie z dowolnego miejsca bez konieczności określania jego ścieżki bezwzględnej:

```
export PATH="/ŚCIEŻKA_DO_DIRSEARCHA:$PATH"
```

Po wykonaniu tego polecenia powinieneś móc używać Dirsearcha bezpośrednio:

```
#!/bin/bash
nmap $1
dirsearch.py -u $1 -e php
```

Zauważ, że po zrestartowaniu terminala konieczne będzie ponowne wykonanie polecenia `export`, aby zmienna `PATH` zawierała ścieżkę do Dirsearcha. Jeżeli nie chcesz w kółko eksportować `PATH`, możesz dodać polecenie `export` do pliku `~/.bash_profile`, który przechowuje preferencje i konfigurację powłoki `bash`. W tym celu możesz otworzyć plik `~/.bash_profile` za pomocą ulubionego edytora tekstu i dodać polecenie `export` na końcu pliku.

Skrypt jest kompletny! Zapisz go w bieżącym katalogu i nazwij *recon.sh*. Rozszerzenie *.sh* jest konwencjonalnym rozszerzeniem skryptów powłoki. Wykonaj polecenie `cd /lokalizacja/twojego/skryptu`, by mieć pewność, że terminal działa w tym samym katalogu roboczym, w którym zapisałeś skrypt. Wykonaj skrypt w terminalu za pomocą poniższego polecenia:

```
$ ./recon.sh
```

Możesz zobaczyć następujący komunikat:

```
permission denied: ./recon.sh
```

Dzieje się tak, ponieważ bieżący użytkownik nie ma uprawnień do wykonania skryptu. Ze względów bezpieczeństwa domyślnie większość plików nie jest wykonywalna. Możesz uruchomić w terminalu poniższe polecenie, aby dodać uprawnienia wykonawcze dla każdego użytkownika:

```
$ chmod +x recon.sh
```

Polecenie `chmod` edytuje uprawnienia dla pliku, a `+x` wskazuje, że chcemy dodać uprawnienia wykonywania dla wszystkich użytkowników. Jeżeli chcesz przyznać uprawnienia wykonywania tylko właścicielowi skryptu, użyj następującego polecenia:

```
$ chmod 700 recon.sh
```

Teraz uruchom skrypt tak jak wcześniej. Spróbuj podać `scanme.nmap.org` jako pierwszy argument. Powinieneś zobaczyć wypisane dane wyjściowe Nmapa i Dirsearcha:

```
$ ./recon.sh scanme.nmap.org
Starting Nmap 7.60 ( https://nmap.org )
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.062s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 992 closed ports
PORT      STATE  SERVICE
22/tcp    open   ssh
25/tcp    filtered smtp
80/tcp    open   http
135/tcp   filtered msrpc
139/tcp   filtered netbios-ssn
445/tcp   filtered microsoft-ds
9929/tcp  open   nping-echo
```

```
31337/tcp open      Elite
Nmap done: 1 IP address (1 host up) scanned in 2.16 seconds

Extensions: php | HTTP method: get | Threads: 10 | Wordlist size: 6023
Error Log: /Users/vickieli/tools/dirsearch/logs/errors.log
Target: scanme.nmap.org
[11:14:30] Starting:
[11:14:32] 403 - 295B - /.htaccessOLD2
[11:14:32] 403 - 294B - /.htaccessOLD
[11:14:33] 301 - 316B - /.svn -> http://scanme.nmap.org/.svn/
[11:14:33] 403 - 298B - /.svn/all-wcprops
[11:14:33] 403 - 294B - /.svn/entries
[11:14:33] 403 - 297B - /.svn/prop-base/
[11:14:33] 403 - 296B - /.svn/pristine/
[11:14:33] 403 - 315B - /.svn/text-base/index.php.svn-base
[11:14:33] 403 - 297B - /.svn/text-base/
[11:14:33] 403 - 293B - /.svn/props/
[11:14:33] 403 - 291B - /.svn/tmp/
[11:14:55] 301 - 318B - /images -> http://scanme.nmap.org/images/
[11:14:56] 200 - 7KB - /index
[11:14:56] 200 - 7KB - /index.html
[11:15:08] 403 - 296B - /server-status/
[11:15:08] 403 - 295B - /server-status
[11:15:08] 301 - 318B - /shared -> http://scanme.nmap.org/shared/
Task Completed
```

Zapisywanie w pliku danych wyjściowych narzędzia

Aby przeanalizować później wyniki rekonesansu, możesz zapisać dane wyjściowe z uruchomienia skryptów w osobnym pliku. Tu do gry wchodzi przekierowanie danych wejściowych i wyjściowych. **Przekierowanie wejścia** polega na wykorzystaniu zawartości pliku lub danych wyjściowych z innego programu jako danych wejściowych dla skryptu. **Przekierowanie wyjścia** to przekazanie danych wyjściowych z programu do innej lokalizacji, takiej jak plik lub inny program. Oto niektóre z najbardziej użytecznych operatorów przekierowań:

```
PROGRAM > NAZWA_PLIKU
```

Zapisuje dane wyjściowe programu w pliku o wskazanej nazwie.
Najpierw czyści zawartość pliku. Jeśli plik nie istnieje, tworzy go.

```
PROGRAM >> NAZWA_PLIKU
```

Dołącza dane wyjściowe programu na końcu pliku bez usuwania jego oryginalnej zawartości.

```
PROGRAM < NAZWA_PLIKU
```

Odczytuje plik i wykorzystuje jego zawartość jako dane wejściowe do programu.

```
PROGRAM1 | PROGRAM2
```

Wykorzystuje dane wyjściowe programu *PROGRAM1* jako dane wejściowe do programu *PROGRAM2*.

Możemy zapisać np. wyniki skanów Nmapa i Dirsearcha w różnych plikach:

```
#!/bin/bash
echo "Tworzenie katalogu $1_recon." ❶
mkdir $1_recon ❷
nmap $1 > $1_recon/nmap ❸
echo "Wynik skanowania nmapa jest zapisywany w pliku $1_recon/nmap."
/ŚCIEŻKA/DO/dirsearch.py -u $1 -e php ❹ --simple-report=$1_recon/dirsearch
echo "Wynik skanowania dirsearcha jest zapisywany w pliku $1_recon/dirsearch."
```

Polecenie `echo` w punkcie ❶ wypisuje komunikat w terminalu. Następnie `mkdir` tworzy w punkcie ❷ katalog o nazwie `DOMAIN_recon`. W punkcie ❸ zapisujemy wyniki uruchomienia `nmap` w pliku o nazwie `nmap` w nowo utworzonym katalogu. Flaga `simple-report` Dirsearcha w punkcie ❹ generuje raport w wyznaczonej lokalizacji. Wyniki Dirsearcha zapisujemy w nowym katalogu w pliku o nazwie `dirsearch`.

Zarządzanie skrypcem możesz ułatwić przez wprowadzenie zmiennych odwołujących się do plików, nazw i wartości. Zmienne w bashu przypisuje się za pomocą składni `NAZWA_ZMIENNEJ=WARTOŚĆ_ZMIENNEJ`. Pamiętaj, że przed znakiem równości i po nim nie powinno być spacji. Do zmiennych odwołuje się w następujący sposób: `$NAZWA_ZMIENNEJ`. Wprowadźmy je do skryptu:

```
#!/bin/bash
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"
DOMAIN=$1
DIRECTORY=${DOMAIN}_recon ❶
echo "Tworzenie katalogu $DIRECTORY."
mkdir $DIRECTORY
nmap $DOMAIN > $DIRECTORY/nmap
echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
$PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-
report=$DIRECTORY/dirsearch ❷
echo "Wynik skanowania dirsearcha jest zapisywany w pliku $DIRECTORY/dirsearch."
```

W punkcie ❶ zamiast `$DOMAIN_recon` używamy `${DOMAIN}_recon`, gdyż w przeciwnym razie powłoka `bash` rozpoznałby jako nazwę zmiennej całe wyrażenie `DOMAIN_recon`. Nawiasy klamrowe instruuja `bash`, że `DOMAIN` jest nazwą zmiennej, a `_recon` jest zwykłym tekstem, który do niej dołączamy. Zwróć uwagę, że w punkcie ❷ ścieżkę do Dirsearcha również zapisaliśmy w zmiennej, co ułatwi ewentualne zmiany w przyszłości.

Korzystając z przekierowań, możesz teraz pisać skrypty powłoki uruchamiające wiele narzędzi w pojedynczym poleceniu i zapisujące ich dane wyjściowe w osobnych plikach.

Dodanie do danych wyjściowych daty skanowania

Załóżmy, że chcesz dodać do danych wyjściowych skryptu bieżącą datę lub wybrać, które skany mają być uruchamiane, zamiast zawsze uruchamiać zarówno Nmap, jak i Dirsearch. Jeżeli chcesz pisać narzędzia z większą liczbą takich funkcjonalności, musisz poznać kilka zaawansowanych koncepcji skryptów powłoki.

Przydatną koncepcją jest np. **podstawianie poleceń** (ang. *command substitution*), czyli wykonywanie operacji na danych wyjściowych polecenia. Użycie znaków `$()` instruuje Unix, aby wykonał polecenie umieszczone w nawiasach i przypisał jego dane wyjściowe do wartości zmiennej. Przećwiczmy stosowanie tej składni:

```
#!/bin/bash
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"
TODAY=$(date) ❶
echo "To skanowanie zostało przeprowadzone $TODAY" ❷
DOMAIN=$1
DIRECTORY=${DOMAIN}_recon
echo "Tworzenie katalogu $DIRECTORY."
mkdir $DIRECTORY
nmap $DOMAIN > $DIRECTORY/nmap
echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
$PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch
echo "Wynik skanowania dirsearcha jest zapisywany w pliku $DIRECTORY/dirsearch."
```

W punkcie ❶ przypisujemy dane wyjściowe polecenia `date` do zmiennej `TODAY`. Polecenie `date` wyświetla bieżącą datę i godzinę. Pozwala to na wypisanie w punkcie ❷ komunikatu wskazującego dzień, w którym wykonaliśmy skanowanie.

Dodawanie opcji wyboru uruchamianych narzędzi

Do selektywnego uruchamiania tylko niektórych narzędzi musisz użyć trybów warunkowych. Składnię instrukcji `if` w powłoce `bash` pokazałam w poniższym listingu. Zwróć uwagę, że instrukcja warunkowa kończy się słowem kluczowym `fi`, czyli zapisanym wstecz słowem `if`:

```
if [ warunek 1 ]
then
    # Wykonaj, jeśli spełniony zostanie warunek 1
elif [ warunek 2 ]
then
    # Wykonaj, jeśli spełniony zostanie warunek 2, a warunek 1 nie
else
    # Wykonaj coś innego, jeśli nie zostanie spełniony żaden warunek
fi
```

Powiedzmy, że chcesz, by użytkownicy mogli określić tryb skanowania:

```
$ ./recon.sh scanme.nmap.org TRYB
```

Funkcjonalność tę możemy zaimplementować w następujący sposób:

```
#!/bin/bash
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"
TODAY=$(date)
echo "To skanowanie zostało przeprowadzone $TODAY"
DIRECTORY=${DOMAIN}_recon
echo "Tworzenie katalogu $DIRECTORY."
mkdir $DIRECTORY
if [ $2 == "nmap-only" ] ❶
then
  nmap $DOMAIN > $DIRECTORY/nmap ❷
  echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
elif [ $2 == "dirsearch-only" ] ❸
then
  $PATH_TO_DIRSEARCH/dirsearch.py
  -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch ❹
  echo "Wynik skanowania dirsearcha jest zapisywany w pliku $DIRECTORY/dirsearch."
else ❺
  nmap $DOMAIN > $DIRECTORY/nmap ❻
  echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
  $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch
  echo "Wynik skanowania dirsearcha jest zapisywany w pliku
  $DIRECTORY/dirsearch."
fi
```

Jeżeli użytkownik określi `nmap-only` w punkcie ❶, uruchamiamy tylko `nmap` i w punkcie ❷ zapisujemy wyniki w pliku o nazwie `nmap`. Jeśli użytkownik określi `dirsearch-only` w punkcie ❸, wykonujemy i zapisujemy wyniki samego `Dirsearcha` w punkcie ❹. Gdy użytkownik nie określi żadnego narzędzia w punkcie ❺, uruchamiamy oba skany w punkcie ❻.

Teraz przez określenie w poleceniu jednej z opcji Twoje narzędzie może uruchomić tylko `Nmap` lub `Dirsearch`:

```
$ ./recon.sh scanme.nmap.org nmap-only
$ ./recon.sh scanme.nmap.org dirsearch-only
```

Uruchamianie dodatkowych narzędzi

A gdybyś chciał mieć również opcję pobierania informacji z narzędzia `crt.sh`? Możesz np. przełączać się między tymi trzema trybami lub uruchamiać wszystkie trzy narzędzia zwiadowcze jednocześnie:

```
$ ./recon.sh scanme.nmap.org nmap-only
$ ./recon.sh scanme.nmap.org dirsearch-only
$ ./recon.sh scanme.nmap.org crt-only
```

Możemy przepisać instrukcje if-else, aby działały z trzema opcjami: najpierw sprawdzamy, czy *TRYB* to *nmap-only*, następnie sprawdzamy, czy *TRYB* to *dirsearch-only*, a na koniec, czy *TRYB* to *crt-only*. To jednak wiele instrukcji if-else, które komplikują kod.

Zamiast tego użyjmy instrukcji case basha, które pozwalają dopasować kilka wartości do jednej zmiennej bez przechodzenia przez długą listę instrukcji if-else. Składnię instrukcji case pokazałam poniżej. Zwróć uwagę, że instrukcja kończy się słowem esac, czyli zapisanym wstecz słowem kluczowym case:

```
case $NAZWA_ZMIENNEJ in
  case1)
    Zrób coś
    ;;
  case2)
    Zrób coś
    ;;
  caseN)
    Zrób coś
    ;;
  *)
    Domyślny przypadek (case), który jest wykonywany,
    jeśli żaden inny nie został dopasowany.
    ;;
esac
```

Możemy ulepszyć nasz skrypt, implementując tę funkcjonalność za pomocą instrukcji case zamiast wielu instrukcji if-else:

```
#!/bin/bash
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"
TODAY=$(date)
echo "To skanowanie zostało przeprowadzone $TODAY"
DOMAIN=$1
DIRECTORY=${DOMAIN}_recon
echo "Tworzenie katalogu $DIRECTORY."
mkdir $DIRECTORY
case $2 in
  nmap-only)
    nmap $DOMAIN > $DIRECTORY/nmap
    echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
    ;;
  dirsearch-only)
    $PATH_TO_DIRSEARCH/dirsearch.py
    -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch
```

```

    echo "Wynik skanowania dirsearcha jest zapisywany w pliku $DIRECTORY/dirsearch."
    ;;
crt-only)
    curl "https://crt.sh?q=$DOMAIN&output=json" -o $DIRECTORY/crt ❶
    echo "Wynik parsowania certa jest zapisywany w pliku $DIRECTORY/crt."
    ;;
*)
    nmap $DOMAIN > $DIRECTORY/nmap
    echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
    $PATH_TO_DIRSEARCH/dirsearch.py
    -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch
    echo "Wynik skanowania dirsearcha jest zapisywany w pliku
$DIRECTORY/dirsearch."
    curl "https://crt.sh?q=$DOMAIN&output=json" -o $DIRECTORY/crt
    echo "Wynik parsowania certa jest zapisywany w pliku $DIRECTORY/crt."
    ;;
esac

```

Polecenie `curl` w punkcie ❶ pobiera zawartość strony. Używamy go tutaj do pobierania danych z `crt.sh`. Opcja `-o` polecenia `curl` pozwala określić plik wyjściowy. Zwróć jednak uwagę, że nasz kod ma wiele powtórzeń! Sekcje kodu, które uruchamia każdy typ skanowania, powtarzają się dwukrotnie. Spróbujmy zredukować powtórzenia za pomocą funkcji. Składnia funkcji powłoki `bash` wygląda następująco:

```

NAZWA_FUNKCJI()
{
    ZRÓB_COS
}

```

Po zadeklarowaniu funkcji możesz ją wywoływać jak każde inne polecenie powłoki w skrypcie. Dodajmy funkcje do skryptu:

```

#!/bin/bash
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"
TODAY=$(date)
echo "To skanowanie zostało przeprowadzone $TODAY"
DOMAIN=$1
DIRECTORY=${DOMAIN}_recon
echo "Tworzenie katalogu $DIRECTORY."
mkdir $DIRECTORY
nmap_scan() ❶
{
    nmap $DOMAIN > $DIRECTORY/nmap
    echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
}
dirsearch_scan() ❷
{

```

```

$PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch
echo "Wynik skanowania dirsearcha jest zapisywany w pliku $DIRECTORY/dirsearch."
}
crt_scan() ❸
{
    curl "https://crt.sh/?q=$DOMAIN&output=json" -o $DIRECTORY/crt
    echo "Wynik parsowania certa jest zapisywany w pliku $DIRECTORY/crt."
}
case $2 in ❹
    nmap-only)
        nmap_scan
        ;;
    dirsearch-only)
        dirsearch_scan
        ;;
    crt-only)
        crt_scan
        ;;
    *)
        nmap_scan
        dirsearch_scan
        crt_scan
        ;;
esac

```

Widać, że uprościliśmy nasz kod. Utworzyliśmy trzy funkcje: `nmap_scan` w punkcie ❶, `dirsearch_scan` w punkcie ❷ i `crt_scan` w punkcie ❸. W tych funkcjach umieszczamy polecenia `scan` i `echo`, dzięki czemu możemy je wywoływać wielokrotnie w punkcie ❹ bez pisania w kółko tego samego kodu. To uproszczenie może nie wydawać się znaczące, ale wielokrotne wykorzystywanie kodu za pomocą funkcji zaoszczędzi Ci wiele zmartwień podczas pisania bardziej złożonych programów.

Należy pamiętać, że wszystkie zmienne powłoki `bash` są *globalne*, z wyjątkiem parametrów wejściowych takich jak `$1`, `$2` i `$3`. Oznacza to, że po zadeklarowaniu zmienne takie jak `$DOMAIN`, `$DIRECTORY` i `$PATH_TO_DIRSEARCH` stają się dostępne w całym skrypcie, nawet jeśli zostały zadeklarowane w ramach funkcji. Z drugiej strony wartości parametrów, takie jak `$1`, `$2` i `$3`, mogą odwoływać się tylko do wartości, z którymi funkcja jest wywoływana, nie można więc używać argumentów wejściowych skryptu w funkcji w taki sposób:

```

nmap_scan()
{
    nmap $1 > $DIRECTORY/nmap
    echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
}
nmap_scan

```

Tutaj \$1 w funkcji odwołuje się do pierwszego argumentu, z którym wywołano `nmap_scan`, a nie do argumentu, z którym wywołany został nasz skrypt `recon.sh`. Ponieważ funkcja `nmap_scan` nie została wywołana z żadnymi argumentami, \$1 jest pusty.

Parsowanie wyników

Mamy narzędzie, które wykonuje trzy rodzaje skanów i zapisuje wyniki w plikach. Jednak po przeprowadzeniu skanowania nadal musielibyśmy własnoręcznie przeczytać i zrozumieć złożone pliki wyjściowe. Czy istnieje sposób, aby przyspieszyć również ten proces?

Powiedzmy, że chcesz wyszukać w plikach wyjściowych pewną informację. Do tego celu możesz użyć narzędzia `grep` (ang. *Global Regular Expression Print*). To narzędzie wiersza poleceń służy do przeszukiwania tekstów, plików i danych wyjściowych poleceń. Proste polecenie `grep` wygląda następująco:

```
grep password file.txt
```

Spowoduje to, że `grep` poszuka łańcucha znaków `password` w pliku `file.txt`, a następnie wypisze pasujące linie w standardowym strumieniu wyjściowym. Możemy np. szybko przeszukać plik wyjściowy `Nmapa`, aby sprawdzić, czy obiekt docelowy ma otwarty port 80:

```
$ grep 80 KATALOG_DOCELOWY/nmap
80/tcp open http
```

Wyszukiwanie może być także bardziej elastyczne, jeżeli w łańcuchu znaków wyszukiwania użyjemy wyrażeń regularnych. **Wyrażenie regularne** (ang. *regular expression — regex*) to specjalny łańcuch znaków, który opisuje wzorzec wyszukiwania. Może pomóc w wyświetlaniu tylko określonych części danych wyjściowych. Zauważyłeś na pewno, że dane wyjściowe polecenia `Nmap` wyglądają następująco:

```
Starting Nmap 7.60 ( https://nmap.org )
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.065s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 992 closed ports
PORT STATE SERVICE
22/tcp open  ssh
25/tcp filtered smtp
80/tcp open  http
135/tcp filtered msrpc
139/tcp filtered netbios-ssn
445/tcp filtered microsoft-ds
```

```
9929/tcp open nping-echo
31337/tcp open Elite
Nmap done: 1 IP address (1 host up) scanned in 2.43 seconds
```

Możesz usunąć nieistotne komunikaty z pliku, aby wyglądał np. tak:

```
PORT STATE SERVICE
22/tcp open  ssh
25/tcp filtered  smtp
80/tcp open  http
135/tcp filtered  msrpc
139/tcp filtered  netbios-ssn
445/tcp filtered  microsoft-ds
9929/tcp open  nping-echo
31337/tcp open  Elite
```

Użyj tego polecenia, aby odfiltrować komunikaty z początku i z końca danych wyjściowych Nmapa i zachowaj tylko istotną część raportu:

```
grep -E "^\s+\s+\s+\s+\s+\s+$" DIRECTORY/nmap > DIRECTORY/nmap_cleaned
```

Flaga `-E` instruuje `grep`, że używasz wyrażenia regularnego. Regex składa się z dwóch części: stałych i operatorów. **Stale** są zestawami łańcuchów znaków, natomiast operatory są symbolami oznaczającymi operacje wykonywane na tych łańcuchach znaków. Połączenie tych dwóch elementów sprawia, że regex jest potężnym narzędziem dopasowywania wzorów. Oto krótki przegląd operatorów wyrażeń regularnych, które reprezentują znaki:

- `\d` dopasowuje dowolną cyfrę;
- `\w` dopasowuje dowolny znak.
- `\s` dopasowuje dowolny znak niedrukowalny, a `\S` dopasowuje dowolne znaki inne niż znaki niedrukowalne;
- `.` dopasowuje dowolny pojedynczy znak;
- `\` znak ucieczki dla znaku specjalnego;
- `^` dopasowuje początek łańcucha znaków lub linii;
- `$` dopasowuje koniec łańcucha znaków lub linii.

Kilka operatorów określa również liczbę dopasowywanych znaków:

- dopasowuje poprzedni znak zero lub więcej razy;
- `+` dopasowuje poprzedni znak co najmniej jeden raz;
- `{3}` dopasowuje poprzedni znak trzy razy;
- `{1, 3}` dopasowuje poprzedni znak od jednego do trzech razy;
- `{1, }` dopasowuje poprzedni znak co najmniej jeden raz;
- `[abc]` dopasowuje jeden ze znaków z nawiasów;

- `[a-z]` dopasowuje jeden ze znaków z zakresu od `a` do `z`;
- `(a|b|c)` dopasowuje `a`, `b` lub `c`.

Przyjrzyjmy się jeszcze raz naszemu wyrażeniu regularnemu. Jak się dowiedziałeś, `\s` dopasowuje znaki niedrukowalne, a `\S` dopasowuje wszystkie pozostałe znaki. Oznacza to, że `\s+` będzie dopasowywać co najmniej jeden znak niedrukowalny, a `\S+` będzie dopasowywać co najmniej jeden znak inny niż niedrukowalny. Ten wzorzec regex określa, że powinniśmy wyodrębnić linie, które zawierają trzy łańcuchy znaków oddzielone dwoma znakami niedrukowalnymi:

```
"^\S+\s+\S+\s+\S+$"
```

Przefiltrowane dane wyjściowe będą wyglądać następująco:

```
PORT STATE SERVICE
22/tcp open  ssh
25/tcp filtered  smtp
80/tcp open  http
135/tcp filtered msrpc
139/tcp filtered netbios-ssn
445/tcp filtered microsoft-ds
9929/tcp open  nping-echo
31337/tcp open  Elite
```

Aby uwzględnić dodatkowe znaki niedrukowalne, które mogą znajdować się w danych wyjściowych polecenia, dodajmy jeszcze dwie opcjonalne spacje wokół naszego łańcucha znaków wyszukiwania:

```
"^\s*\S+\s+\S+\s+\S+\s*\s*$"
```

W celu przeprowadzenia bardziej wyrafinowanego dopasowywania możesz użyć wielu bardziej zaawansowanych funkcjonalności wyrażeń regularnych. Jednak ten prosty zestaw operatorów dobrze służy naszym celom. Więcej informacji na temat składni wyrażeń regularnych znajdziesz w przewodniku *Quick-Start: Regex Cheat Sheet* (<https://www.rexegg.com/regex-quickstart.html>).

Tworzenie raportu głównego

Co zrobić, jeśli chcesz wygenerować raport główny ze wszystkich trzech plików wyjściowych? Musisz sparsować plik JSON z `crt.sh`. Możesz to zrobić za pomocą narzędzia wiersza poleceń `jq`, które przetwarza pliki JSON. Jeśli zbadamy plik wyjściowy JSON z `crt.sh`, zobaczymy, że musimy wyodrębnić pole `name_value` każdego elementu certyfikatu, aby wyodrębnić nazwy domen. Służy do tego następujące polecenie:

```
$ jq -r ".[] | .name_value" $DOMAIN/crt
```

Flaga `-r` instruuje `jq`, by zapisywać dane wyjściowe bezpośrednio w standardowym strumieniu wyjściowym, zamiast formatować je jako łańcuchy znaków JSON. `.[]` iteruje przez tablicę w pliku JSON, a `.name_value` wyodrębnia pole `name_value` każdego elementu. Natomiast `$DOMAIN/crt` jest plikiem wejściowym polecenia `jq`. Więcej informacji o działaniu narzędzia `jq` znajdziesz w jego instrukcji (<https://stedolan.github.io/jq/manual/>).

Aby połączyć wszystkie pliki wyjściowe w raport główny, należy napisać taki skrypt:

```
#!/bin/bash
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"
DOMAIN=$1
DIRECTORY=${DOMAIN}_recon
echo "Tworzenie katalogu $DIRECTORY."
mkdir $DIRECTORY
nmap_scan()
{
    nmap $DOMAIN > $DIRECTORY/nmap
    echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
}
dirsearch_scan()
{
    $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-
report=$DIRECTORY/dirsearch
    echo "Wynik skanowania dirsearcha jest zapisywany w pliku
$DIRECTORY/dirsearch."
}
crt_scan()
{
    curl "https://crt.sh/?q=$DOMAIN&output=json" -o $DIRECTORY/crt
    echo "Wynik parsowania certa jest zapisywany w pliku $DIRECTORY/crt."
}
case $2 in
    nmap-only)
        nmap_scan
        ;;
    dirsearch-only)
        dirsearch_scan
        ;;
    crt-only)
        crt_scan
        ;;
    *)
        nmap_scan
        dirsearch_scan
        crt_scan
        ;;
esac
echo "Generowanie raportu rekonesansu z plików wyjściowych..."
TODAY=$(date)
```

```
echo "To skanowanie zostało przeprowadzone $TODAY" > $DIRECTORY/report ❶  
echo "Wyniki dla Nmapa:" >> $DIRECTORY/report  
grep -E "^\\s*\\S+\\s+\\S+\\s+\\S+\\s+\\S+\\s*$" $DIRECTORY/nmap >> $DIRECTORY/report ❷  
echo "Wyniki dla Dirsearcha:" >> $DIRECTORY/report  
cat $DIRECTORY/dirsearch >> $DIRECTORY/report ❸  
echo "Wyniki dla crt.sh:" >> $DIRECTORY/report  
jq -r ".[] | .name_value" $DIRECTORY/crt >> $DIRECTORY/report ❹
```

Najpierw w punkcie ❶ tworzymy nowy plik o nazwie report i zapisujemy w nim bieżącą datę, aby śledzić, kiedy raport został wygenerowany. Następnie dołączamy do pliku raportu wyniki poleceń nmap i dirsearch w punkcie ❷. Polecenie cat wypisuje zawartość pliku w standardowym strumieniu wyjściowym, ale możemy go użyć również w punkcie ❸ do przekierowania zawartości pliku do innego pliku. Na koniec wyodrębniamy nazwy domen z raportu crt.sh i dołączamy je na końcu pliku raportu w punkcie ❹.

Skanowanie wielu domen

A jeśli chcemy skanować wiele domen jednocześnie? Podczas przeprowadzania rekonesansu obiektu docelowego możemy zacząć od kilku nazw domenowych danej organizacji. Wiemy, że Facebook jest np. właścicielem zarówno domen *facebook.com*, jak i *fbcdn.net*. Jednak nasz obecny skrypt pozwala nam skanować tylko jedną domenę na raz. Musimy napisać narzędzie, które może skanować wiele domen za pomocą jednego polecenia, np. tak:

```
./recon.sh facebook.com fbcdn.net nmap-only
```

Kiedy skanujemy w ten sposób wiele domen, potrzebujemy sposobu na rozróżnienie, które argumenty określają *TRYB* skanowania, a które domeny docelowe. Jak widziałeś, w przypadku poprzednich narzędzi większość z nich pozwala użytkownikom modyfikować zachowanie narzędzia za pomocą *opcji* lub *flag* wiersza poleceń, takich jak `-u` i `--simple-report`.

Narzędzie `getopts` parsuje opcje z wiersza poleceń przy użyciu jednoznakowych flag. W pokazanej poniżej składni *OPTSTRING* określa litery opcji, które `getopts` powinien rozpoznać. Jeżeli ma rozpoznawać np. opcje `-m` oraz `-i`, należy określić `mi`. Jeśli chcesz, aby opcja zawierała wartości argumentów, po literze powinien występować dwukropek, w ten sposób: `m:i`. Argument *NAZWA* określa nazwę zmiennej, która przechowuje literę opcji.

```
getopts OPTSTRING NAZWA
```

Aby zaimplementować naszą funkcjonalność skanowania wielu domen, możemy pozwolić użytkownikom używać flagi `-m` do określenia trybu skanowania i założyć, że wszystkie pozostałe argumenty będą domenami. Instruujemy tutaj narzędzie `getopts`, by rozpoznało opcję, jeśli jej flagą jest `-m`, oraz że ta opcja powinna zawierać

wartość wejściową. Narzędzie `getopts` automatycznie zapisuje wartość wszystkich opcji w zmiennej `$OPTARG`. Wartość tę możemy zapisać w zmiennej o nazwie `MODE`:

```
getopts "m:" OPTION
MODE=$OPTARG
```

Jeśli uruchomisz teraz skrypt powłoki z flagą `-m`, skrypt będzie wiedział, że określasz tryb (`MODE`) skanowania! Zwróć uwagę, że `getopts` przestaje parsować argumenty, gdy natrafia na argument, który nie zaczyna się od znaku dywizu (`-`), podczas uruchamiania skryptu musisz więc umieścić tryb skanowania przed argumentami domeny:

```
./recon.sh -m nmap-only facebook.com fbcdn.net
```

Następnie potrzebujemy sposobu na odczytanie każdego argumentu domeny i wykonanie na nim skanu. Użyjmy pętli! Bash ma dwa typy pętli: `for` i `while`. Do naszych celów lepiej nadaje się pętla `for`, gdyż znamy już liczbę wartości, przez które wykonujemy pętlę. Generalnie z pętli `for` powinieneś korzystać, gdy masz już listę wartości do iteracji. Pętli `while` należy używać, gdy nie jest się pewnym, przez ile wartości będzie przechodzić pętla, ale chce się określić warunek, w którym wykonywanie powinno się zatrzymać.

Oto składnia pętli `for` w powłoce bash. Dla każdego elementu z `LISTA_WARTOŚCI` bash wykona jeden raz kod znajdujący się między `do` i `done`:

```
for i in LISTA_WARTOŚCI
do
    ZRÓB COŚ
done
```

Zaimplementujmy teraz naszą funkcjonalność za pomocą pętli `for`:

```
for i in "${@:$OPTARG:$#}" ❶
do
    # Wykonaj skanowania dla $i
done
```

W punkcie ❶ tworzymy tablicę, która zawiera każdy argument wiersza poleceń oprócz tych, które zostały już sparsowane przez narzędzie `getopts` przechowujące indeks pierwszego argumentu po opcjach parsowanych do zmiennej o nazwie `$OPTARG`. Znaki `@` reprezentują tablicę zawierającą wszystkie argumenty wejściowe, natomiast `$#` jest liczbą przekazanych argumentów wiersza poleceń. `"${@:$OPTARG:$#}"` dzieli tablicę na wycinki w taki sposób, że usuwa argument `MODE`, taki jak `nmap-only`, gwarantując, że iterujemy tylko przez tę część naszego wejścia,

którą są domeny. Tworzenie wycinków tablicy to sposób wyodrębnienia z niej podzbioru elementów. W powłocie bash można wycinać tablice za pomocą następującej składni (należy pamiętać, że cudzysłów wokół polecenia jest konieczny):

```
"${TABLICA_WEJŚCIOWA:INDEKS_POCZĄTKOWY:INDEKS_KONCOWY}"
```

Zmienna `$i` reprezentuje bieżący element w tablicy argumentów. Następnie możemy opakować kod pętlą:

```
#!/bin/bash
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"
nmap_scan()
{
    nmap $DOMAIN > $DIRECTORY/nmap
    echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
}
dirsearch_scan()
{
    $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-
report=$DIRECTORY/dirsearch
    echo "Wynika skanowania dirsearcha jest zapisywany w pliku
$DIRECTORY/dirsearch."
}
crt_scan()
{
    curl "https://crt.sh/?q=$DOMAIN&output=json" -o $DIRECTORY/crt
    echo "Wynik parsowania certa jest zapisywany w pliku $DIRECTORY/crt."
}
getopts "m:" OPTION
MODE=$OPTARG

for i in "${@:$OPTIND:$#}" ❶
do

    DOMAIN=$i
    DIRECTORY=${DOMAIN}_recon
    echo "Tworzenie katalogu $DIRECTORY."
    mkdir $DIRECTORY

    case $MODE in
        nmap-only)
            nmap_scan
            ;;
        dirsearch-only)
            dirsearch_scan
            ;;
        crt-only)
            crt_scan
            ;;
        *)
    
```

```

nmap_scan
dirsearch_scan
crt_scan
;;
esac
echo "Generowanie raportu rekonesansu dla $DOMAIN..."
TODAY=$(date)
echo "To skanowanie zostało przeprowadzone $TODAY" > $DIRECTORY/report
if [ -f $DIRECTORY/nmap ];then ❷
echo "Wyniki dla Nmapa:" >> $DIRECTORY/report
grep -E "^\\s*\\s+\\s+\\s+\\s+\\s+\\s*$" $DIRECTORY/nmap >> $DIRECTORY/report
fi
if [ -f $DIRECTORY/dirsearch ];then ❸
echo "Wyniki dla Dirsearch:" >> $DIRECTORY/report
cat $DIRECTORY/dirsearch >> $DIRECTORY/report
fi
if [ -f $DIRECTORY/crt ];then ❹
echo "Wyniki dla crt.sh:" >> $DIRECTORY/report
jq -r ".[] | .name_value" $DIRECTORY/crt >> $DIRECTORY/report
fi
done ❺

```

Pętla for rozpoczyna się słowem kluczowym for w punkcie ❶, a kończy słowem kluczowym done w punkcie ❺. Zauważ, że dodaliśmy również kilka linii w sekcji raportu, aby sprawdzić, czy musimy wygenerować każdy typ raportu. W punktach ❷, ❸ i ❹ sprawdzamy, czy istnieją pliki wyjściowe skanowania Nmapa, Dirsearcha lub crt.sh, byśmy mogli określić, czy musimy wygenerować raport dla tego skanowania typu.

Nawiasy kwadratowe, w których umieszczamy warunek, oznaczają, że przekazujemy zawartość do polecenia test: [-f \$DIRECTORY/nmap] jest równoważne z test -f \$DIRECTORY/nmap.

Polecenie test przeprowadza ewaluację warunku i jako dane wyjściowe generuje true lub false. Flaga -f wykonuje test na istnienie pliku. Możesz jednak przetestować więcej warunków! Omówię kilka przydatnych warunków testowych. Flagi -eq i -ne są testami odpowiednio na równość i nierówność. To zwraca wartość true, jeśli \$3 jest równe 1:

```
if [ $3 -eq 1 ]
```

To zwraca wartość true, jeśli \$3 nie jest równe 1:

```
if [ $3 -ne 1 ]
```

Flagi -gt, -ge, -lt i -le testują wartości odpowiednio: „większe niż”, „większe lub równe”, „mniejsze niż” oraz „mniejsze lub równe”:

```
if [ $3 -gt 1 ]
if [ $3 -ge 1 ]
if [ $3 -lt 1 ]
if [ $3 -le 1 ]
```

Flagi `-z` i `-n` sprawdzają, czy łańcuch znaków jest pusty. Oba te warunki są spełnione:

```
if [ -z "" ]
if [ -n "abc" ]
```

Flagi `-d`, `-f`, `-r`, `-w` i `-x` sprawdzają statusy katalogów i plików. Możesz używać ich do weryfikowania istnienia i uprawnień pliku przed uruchomieniem skryptu powłoki. Poniższe polecenie zwraca np. `true`, jeśli katalog `/bin` istnieje:

```
if [ -d /bin]
```

To zwraca wartość `true`, jeśli plik `/bin/bash` istnieje:

```
if [ -f /bin/bash ]
```

A to zwraca `true`, jeśli `/bin/bash` jest plikiem do odczytu:

```
if [ -r /bin/bash ]
```

Plik z uprawnieniami zapisu:

```
if [ -w /bin/bash ]
```

Plik wykonywalny:

```
if [ -x /bin/bash ]
```

Aby łączyć wyrażenia testowe, możesz ponadto stosować operatory `&&` i `||`. Poniższe polecenie zwraca `true`, jeżeli oba wyrażenia są prawdziwe:

```
if [ $3 -gt 1 ] && [ $3 -lt 3 ]
```

Z kolei to polecenie zwraca `true`, jeśli przynajmniej jeden z warunków jest prawdziwy:

```
if [ $3 -gt 1 ] || [ $3 -lt 0 ]
```

Więcej flag porównawczych znajdziesz w instrukcji polecenia `test` po uruchomieniu `man test`. (Jeżeli nie masz pewności odnośnie do używanych poleceń, zawsze możesz wpisać w terminalu `man`, a następnie nazwę polecenia, aby uzyskać dostęp do pliku instrukcji polecenia).

Pisanie biblioteki funkcji

Gdy Twoja baza kodu będzie się powiększać, powinieneś rozważyć napisanie **biblioteki funkcji** w celu wielokrotnego wykorzystywania kodu. Wszystkie powszechnie używane funkcje możesz przechowywać w pliku o nazwie *scan.lib*. Dzięki temu będziesz mógł wywoływać te funkcje, gdy będą potrzebne do przyszłych zadań rekonesansowych:

```
#!/bin/bash
nmap_scan()
{
    nmap $DOMAIN > $DIRECTORY/nmap
    echo "Wynik skanowania nmapa jest zapisywany w pliku $DIRECTORY/nmap."
}
dirsearch_scan()
{
    $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-
report=$DIRECTORY/dirsearch
    echo "Wynik skanowania dirsearcha jest zapisywany w pliku
$DIRECTORY/dirsearch."
}
crt_scan()
{
    curl "https://crt.sh/?q=$DOMAIN&output=json" -o $DIRECTORY/crt
    echo "Wynik parsowania certa jest zapisywany w pliku $DIRECTORY/crt."
}
```

W kolejnym pliku możesz wskazać źródłowy plik biblioteki w celu wykorzystania wszystkich jego funkcji i zmiennych. Skrypt źródłowy importuje się za pomocą polecenia `source`, po którym następuje ścieżka do skryptu:

```
#!/bin/bash
source ./scan.lib
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"
getopts "m:" OPTION
MODE=$OPTARG
for i in "${@:$OPTIND:$#}"
do
    DOMAIN=$i
    DIRECTORY=${DOMAIN}_recon
    echo "Tworzenie katalogu $DIRECTORY."
    mkdir $DIRECTORY

    case $MODE in
```

```

nmap-only)
    nmap_scan
    ;;
dirsearch-only)
    dirsearch_scan
    ;;
crt-only)
    crt_scan
    ;;
*)
    nmap_scan
    dirsearch_scan
    crt_scan
    ;;
esac
echo "Generowanie raportu rekonesansu dla $DOMAIN..."
TODAY=$(date)
echo "To skanowanie zostało przeprowadzone $TODAY" > $DIRECTORY/report
if [ -f $DIRECTORY/nmap ];then
    echo "Wyniki dla Nmapa:" >> $DIRECTORY/report
    grep -E "^\\s*\\S+\\s+\\S+\\s+\\S+\\s+\\S+\\s*$" $DIRECTORY/nmap >> $DIRECTORY/report
fi
if [ -f $DIRECTORY/dirsearch ];then
    echo "Wyniki dla Dirsearcha:" >> $DIRECTORY/report
    cat $DIRECTORY/dirsearch >> $DIRECTORY/report
fi
if [ -f $DIRECTORY/crt ];then
    echo "Wyniki dla crt.sh:" >> $DIRECTORY/report
    jq -r ".[] | .name_value" $DIRECTORY/crt >> $DIRECTORY/report
fi
done

```

Korzystanie z biblioteki może być bardzo przydatne podczas tworzenia wielu narzędzi, które wymagają tych samych funkcjonalności. Możesz budować np. wiele narzędzi sieciowych wymagających rozwiązywania DNS-owego. W takim przypadku możesz po prostu napisać tę funkcjonalność raz i używać jej we wszystkich swoich narzędziach.

Budowanie interaktywnych programów

A gdybyś chciał zbudować interaktywny program, który podczas działania pobiera dane wejściowe od użytkownika? Jeśli użytkownik wprowadzi np. opcję wiersza poleceń `-i`, program wejdzie w tryb interaktywny i pozwoli na bieżąco określać domeny do skanowania:

```
./recon.sh -i -m nmap-only
```

Do tego celu możesz użyć polecenia `read`. Odczytuje ono dane wejściowe użytkownika i zapisuje w zmiennej wejściowy łańcuch znaków:

```
echo "Proszę wpisać domenę!"
read $DOMAIN
```

Te polecenia wyświetlą monit, w którym użytkownik zostanie poproszony o wprowadzenie domeny, a następnie zapiszą dane wejściowe w zmiennej o nazwie `$DOMAIN`.

Aby wielokrotnie wyświetlać zapytanie dla użytkownika, musimy użyć pętli `while`, która będzie wypisywać monit z prośbą o wpisanie domeny, dopóki użytkownik nie zakończy działania programu. Poniżej przedstawiłam składnię pętli `while`. Dopóki `WARUNEK` jest spełniony, pętla `while` będzie wielokrotnie wykonywać kod znajdujący się pomiędzy `do` i `done`:

```
while WARUNEK
do
    ZRÓB COŚ
done
```

Pętlę `while` możemy wykorzystać do ponawiania zapytania o domeny, dopóki użytkownik nie wykona polecenia `quit`:

```
while [ $INPUT != "quit" ];do
    echo "Proszę wpisać domenę!"
    read INPUT
    if [ $INPUT != "quit" ];then
        scan_domain $INPUT
        report_domain $INPUT
    fi
done
```

Potrzebujemy również sposobu, aby użytkownicy faktycznie wywoływali opcję `-i`, a obecnie nasze polecenie `getopts` tego nie obsługuje. Możemy użyć pętli `while` do parsowania opcji przez wielokrotne używanie `getopts`:

```
while getopts "m:i" OPTION; do
    case $OPTION in
        m)
            MODE=$OPTARG
            ;;
        i)
            INTERACTIVE=true
            ;;
    esac
done
```

Określamy tutaj pętlę `while`, która ustawicznie pobiera opcje wiersza poleceń. Jeśli flagą opcji jest `-m`, ustawiamy dla zmiennej `MODE` tryb skanowania określony przez użytkownika. Jeżeli flagą opcji jest `-i`, ustawiamy dla zmiennej `$INTERACTIVE` wartość `true`. Dalej w skrypcie możemy zdecydować, czy wywołać tryb interaktywny, przez sprawdzenie wartości zmiennej `$INTERACTIVE`. Po złożeniu tego wszystkiego do kupy otrzymujemy nasz ostateczny skrypt:

```
#!/bin/bash
source ./scan.lib

while getopts "m:i" OPTION; do
    case $OPTION in
        m)
            MODE=$OPTARG
            ;;
        i)
            INTERACTIVE=true
            ;;
    esac
done

scan_domain(){
    DOMAIN=$1
    DIRECTORY=${DOMAIN}_recon
    echo "Tworzenie katalogu $DIRECTORY."
    mkdir $DIRECTORY
    case $MODE in
        nmap-only)
            nmap_scan
            ;;
        dirsearch-only)
            dirsearch_scan
            ;;
        crt-only)
            crt_scan
            ;;
        *)
            nmap_scan
            dirsearch_scan
            crt_scan
            ;;
    esac
}

report_domain(){
    DOMAIN=$1
    DIRECTORY=${DOMAIN}_recon
    echo "Generowanie raportu rekonesansu dla $DOMAIN..."
    TODAY=$(date)
    echo "To skanowanie zostało przeprowadzone $TODAY" > $DIRECTORY/report
    if [ -f $DIRECTORY/nmap ];then
        echo "Wyniki dla Nmapa:" >> $DIRECTORY/report
        grep -E "\s*\S+\s+\S+\s+\S+\s*\$" $DIRECTORY/nmap >> $DIRECTORY/report
```



```

fi
if [ -f $DIRECTORY/dirsearch ];then
    echo "Wyniki dla Dirsearcha:" >> $DIRECTORY/report
    cat $DIRECTORY/dirsearch >> $DIRECTORY/report
fi
if [ -f $DIRECTORY/crt ];then
    echo "Wyniki crt.sh:" >> $DIRECTORY/report
    jq -r ".[] | .name_value" $DIRECTORY/crt >> $DIRECTORY/report
fi
}
if [ $INTERACTIVE ];then ❶
    INPUT="BLANK"
    while [ $INPUT != "quit" ];do ❷
        echo "Proszę wpisać domenę!"
        read INPUT
        if [ $INPUT != "quit" ];then ❸
            scan_domain $INPUT
            report_domain $INPUT
        fi
    done
else
    for i in "${@:$OPTIND:$#}";do
        scan_domain $i
        report_domain $i
    done
fi

```

W tym programie najpierw w punkcie ❶ sprawdzamy, czy użytkownik wybrał tryb interaktywny przez określenie opcji `-i`. Następnie wielokrotnie pytamy użytkownika o domenę za pomocą pętli `while` w punkcie ❷. Jeśli danymi wejściowymi użytkownika nie jest słowo kluczowe `quit`, zakładamy, że użytkownik wprowadził domenę docelową, więc skanujemy i tworzymy raport dla tej domeny. Pętla `while` będzie kontynuować działanie i pytać użytkownika o domeny, dopóki użytkownik nie wpisze `quit`, co spowoduje wyjście z pętli `while` w punkcie ❸ i zakończenie programu.

Interaktywne narzędzia mogą pomóc w sprawniejszym działaniu przepływu pracy. Możesz zbudować np. narzędzia testowe, które umożliwią wybór sposobu postępowania na podstawie wstępnych wyników.

Używanie specjalnych zmiennych i znaków

Masz już wystarczającą wiedzę na temat powłoki `bash`, by zbudować wiele wszechstronnych narzędzi. Ten punkt podrozdziału zawiera kolejne wskazówki dotyczące specyfiki skryptów powłoki.

W Unikse polecenia zwracają `0` w przypadku powodzenia i dodatnią liczbę całkowitą w przypadku niepowodzenia. Zmienna `?` zawiera wartość wyjścia z ostatniego wykonanego polecenia. Można jej używać do testowania powodzenia i niepowodzenia wykonywania poleceń:

```
#!/bin/sh
chmod 777 script.sh
if [ "$?" -ne "0" ]; then
    echo "Uruchomienie chmod się nie powiodło. Prawdopodobnie nie masz uprawnień!"
fi
```

Kolejną specjalną zmienną jest \$\$, która zawiera identyfikator bieżącego procesu. Jest to przydatne, gdy musisz utworzyć pliki tymczasowe dla skryptu. Jeśli masz uruchomionych w tym samym czasie wiele instancji tego samego skryptu lub programu, każda z nich może potrzebować własnych plików tymczasowych. W takim przypadku możesz tworzyć dla nich wszystkich pliki tymczasowe o nazwach */tmp/nazwa_skryptu_\$\$*.

Jak zapewne pamiętasz, wcześniej w tym rozdziale pisałam o zakresach zmiennych w skryptach powłoki. Zmienne, które nie są parametrami wejściowymi, są globalne dla całego skryptu. Jeżeli chcesz, aby inne programy również korzystały z konkretnej zmiennej, musisz ją wyeksportować:

```
export NAZWA_ZMIENNEJ=WARTOŚĆ_ZMIENNEJ
```

Powiedzmy, że w jednym ze skryptów ustawisz zmienną VAR:

```
VAR="hello!"
```

Jeżeli jej nie wyeksportujesz lub nie zaimportujesz jako źródła w innym skrypcie, wartość zostanie zniszczona po wykonaniu skryptu. Jeśli jednak wyeksportujesz VAR w pierwszym skrypcie i uruchomisz ten skrypt przed uruchomieniem kolejnego, ten drugi skrypt będzie mógł odczytać wartość VAR.

Powinieneś również pamiętać o znakach specjalnych w bashu. W Uniksie znak wieloznaczny (*) oznacza **wszystko**. Poniższe polecenie wypisze np. nazwy wszystkich plików z bieżącego katalogu, które mają rozszerzenie *.txt*:

```
$ ls *.txt
```

Grawisy (^) oznaczają podstawianie poleceń. Mają one to samo zastosowanie, co wspomniana wcześniej składnia podstawiania poleceń \$(). Poniższe polecenie echo wypisze dane wyjściowe polecenia whoami:

```
echo `whoami`
```

Większość znaków specjalnych, takich jak znak wieloznaczny lub pojedynczy cudzysłów, nie jest interpretowana w specjalny sposób, jeśli jest umieszczona w cudzysłowie. Znaki te są wtedy traktowane jako część łańcucha znaków. To polecenie wypisze np. łańcuch znaków "abc '*' 123":

```
$ echo "abc '*' 123"
```

Kolejnym ważnym znakiem specjalnym jest lewy ukośnik (`\`), który w powłoce bash jest znakiem ucieczki. Instruuje powłokę, że dany znak powinien być interpretowany dosłownie, a nie jako znak specjalny.

Niektóre znaki specjalne, takie jak podwójny cudzysłów, znak dolara, grawis i lewe ukośniki, zachowują specjalne funkcje nawet kiedy są umieszczone w podwójnych cudzysłowach, więc jeśli chcesz, aby bash traktował je literalnie, musisz zastosować do nich znak ucieczki, czyli lewy ukośnik:

```
$ echo "\" to podwójny cudzysłów. \$ to znak dolara. ` to jest grawis.  
\\ to lewy ukośnik."
```

To polecenie wypisze następujące dane wyjściowe:

```
" to podwójny cudzysłów. $ to znak dolara. ` to jest grawis. \ to lewy ukośnik.
```

Możesz również użyć lewego ukośnika przed nową linią, by wskazać, że linia kodu nie została zakończona, np. to polecenie:

```
chmod 777 \  
script.sh
```

jest równoważne z tym:

```
chmod 777 script.sh
```

Gratulacje! Możesz już pisać skrypty basha. Z początku pisanie ich może się wydawać przerażające, ale stopniowo opanujesz tę umiejętność i stanie się to potężnym dodatkiem do arsenału hakerskiego. Będziesz w stanie przeprowadzać lepszy rekonesans, wykonywać bardziej efektywne testy i mieć bardziej zorganizowany przepływ pracy hakerskiej.

Jeśli planujesz zaimplementować automatyzację w szerokim zakresie, dobrym pomysłem jest rozpoczęcie porządkowania skryptów od samego początku. Skonfiguruj katalog skryptów i posortuj je według ich funkcjonalności. Stanie się to początkiem rozwoju Twojej własnej metodologii hakerskiej. Po zebraniu kilku skryptów, z których będziesz regularnie korzystać, będziesz mógł je uruchamiać automatycznie również za pomocą skryptów. Możesz np. podzielić skrypty na rekonesansowe, fuzzingowe, automatycznie raportujące itd. W ten sposób za każdym razem, gdy znajdziesz odpowiedni skrypt lub właściwe narzędzie, będziesz mógł szybko dodać je w zorganizowany sposób do swojego przepływu pracy.

Planowanie automatycznego skanowania

Przenieśmy teraz automatyzację na wyższy poziom i utwórzmy system ostrzegania, który będzie nas informował, gdy w naszych skanach pojawi się coś interesującego. Unikniemy dzięki temu konieczności ręcznego uruchamiania poleceń i wielokrotnego przeczesywania wyników.

Do zaplanowania skanowania możemy użyć zadań `crona`. **Cron** to dyspozytor zadań w systemach operacyjnych opartych na Uniksie. Umożliwia planowanie okresowego uruchamiania zadań. Możesz np. zaplanować uruchamianie skryptu, który codziennie o tej samej porze będzie sprawdzał nowe punkty końcowe w określonej witrynie. Możesz też uruchomić skaner, który każdego dnia będzie przeprowadzał sprawdzanie tego samego obiektu pod kątem luk w zabezpieczeniach. W ten sposób można monitorować zmiany w zachowaniu aplikacji i szukać sposobów jej eksploatacji.

Zachowanie `crona` można konfigurować przez edycję plików o nazwie `crontab`. Unix przechowuje różne kopie plików `crontab` dla poszczególnych użytkowników. Aby wyedytować `crontab` swojego użytkownika, uruchom następujące polecenie:

```
crontab -e
```

We wszystkich plikach `crontab` stosuje się tę samą składnię:

```
A B C D E polecenie_do_wykonania
```

gdzie:

- *A*: minuta (0 – 59)
- *B*: godzina (0 – 23)
- *C*: dzień (1 – 31)
- *D*: miesiąc (1 – 12)
- *E*: dzień tygodnia (0 – 7; niedziela to 0 lub 7, poniedziałek to 1 itd.)

Każda linia wskazuje polecenie do uruchomienia i czas, w którym powinno to nastąpić. Do zdefiniowania czasu używa się pięciu liczb: pierwsza liczba z przedziału od 0 do 59 określa minutę, druga liczba z przedziału od 0 do 23 określa godzinę, trzecia i czwarta liczba to dzień i miesiąc, a ostatnia liczba z przedziału od 0 do 7 określa dzień tygodnia, w którym polecenie powinno być uruchamiane (0 i 7 oznaczają, że polecenie powinno być uruchamiane w niedziele, 1 oznacza, że polecenie powinno być uruchamiane w poniedziałki itd.).

Aby uruchamiać skrypt rekonesansu np. codziennie o 21:30, możesz dodać do swojego pliku `crontab` tę linię:

```
30 21 * * * ./scan.sh
```

Skrypty można również uruchamiać wsadowo z katalogów. Wpisanie w pliku w *crontab* polecenia `run-parts` instruuje cron, by uruchomił wszystkie skrypty zapisane w określonym katalogu. Możesz np. przechowywać wszystkie narzędzia zwiadowcze w katalogu i okresowo skanować cele. Poniższa linia instruuje cron, aby uruchamiał wszystkie skrypty z mojego katalogu *security* codziennie o 21:30:

```
30 21 * * * run-parts /Users/vickie/scripts/security
```

Kolejne polecenie `git diff` wypisuje różnicę między dwoma plikami. By z niego korzystać, musisz zainstalować program Git. Polecenia `git diff` możesz używać do porównywania wyników skanowania z różnych okresów, co pozwala szybko sprawdzić, czy obiekt docelowy zmienił się od ostatniego skanowania:

```
git diff SKAN_1 SKAN_2
```

Pomaga to zidentyfikować wszelkie nowe domeny, poddomeny, punkty końcowe i inne nowe zasoby celu. Możesz napisać skrypt, który będzie codziennie powiadamiał Cię o zmianach w obiekcie docelowym:

```
#!/bin/bash
DOMAIN=$1
DIRECTORY=${DOMAIN}_recon
echo "Sprawdzanie zmian w celu: $DOMAIN.\n Znaleziono następujące nowe rzeczy."
git diff <SKAN_Z_CZASU_1> <SKAN_Z_CZASU_2>
```

Następnie możesz zaplanować uruchamianie tego skryptu za pomocą crona:

```
30 21 * * * ./scan_diff.sh facebook.com
```

Te techniki automatyzacji pomagają mi szybko wyszukiwać w celach nowe pliki JavaScriptu, punkty końcowe i funkcjonalności. Lubię używać tej techniki szczególnie do automatycznego wykrywania luk w zabezpieczeniach umożliwiających przejście poddomeny. Przejęcia poddomeny omówię w rozdziale 20.

Do śledzenia zmian możesz ewentualnie używać GitHuba. Na stronie <https://github.com/new/> skonfiguruj repozytorium do przechowywania wyników skanowania. GitHub posiada funkcjonalność powiadomień, która informuje o istotnych zdarzeniach w repozytorium. Aby ją skonfigurować, po zalogowaniu do GitHuba wybierz z menu *Settings*, a następnie w lewym panelu *Notifications*. Podaj adres e-mailowy, który będzie używany do powiadamiania o zmianach. Następnie w katalogu, gdzie przechowujesz wyniki skanowania, uruchom poniższe polecenia, aby zainicjować w nim git:

```
git init
git remote add origin https://ŚCIEŻKA_DO_REPOZYTORIUM
```

Ostatnim krokiem jest zastosowanie crona do okresowego skanowania celu i przesyłania plików do GitHuba:

```
30 21 * * * ./recon.sh facebook.com
40 21 * * * git add *; git commit -m "nowy skan"; git push -u origin master
```

GitHub będzie wtedy wysyłał Ci e-maile o plikach, które zmieniły się podczas nowego skanowania.

Kilka słów na temat interfejsów API rekonesansu

Wiele narzędzi wymienionych w tym rozdziale posiada interfejsy API, które umożliwiają integrację ich usług z aplikacjami i skryptami. Interfejsy API opiszę szerzej w rozdziale 24., na razie możesz jednak traktować je jako punkty końcowe, które umożliwiają wysyłanie zapytań do bazy danych określonej usługi. Korzystając z tych API, możesz w skryptach kwerendować narzędzia rekonesansu i dodawać wyniki do raportu zwiadowczego bez konieczności ręcznego odwiedzania stron narzędzi.

Shodan ma np. interfejs API (<https://developer.shodan.io/>), który umożliwia przeszukiwanie jego bazy danych. Dostęp do wyników skanowania hosta możesz uzyskać za pośrednictwem adresu URL https://api.shodan.io/shodan/host/{ip}?key={TWÓJ_KLUCZ_API}. Możesz skonfigurować skrypt basha, aby wysyłał żądania do tego adresu URL i parsował wyniki. LinkedIn również ma API (<https://www.linkedin.com/developers/>), które umożliwia kwerendowanie jego bazy danych. Do uzyskania dostęp do informacji o użytkowniku LinkedIna możesz użyć np. tego adresu URL: https://api.linkedin.com/v2/people/{ID_UŻYTKOWNIKA}. Interfejs API Censysa (<https://censys.io/api>) umożliwia dostęp do certyfikatów przez kwerendowanie punktu końcowego <https://censys.io/api/v1>.

Także inne narzędzia wymienione w tym rozdziale, takie jak BuiltWith, wyszukiwanie Google i wyszukiwanie w GitHubie, mają własne usługi API. Te interfejsy API mogą pomóc w skuteczniejszym odkrywaniu zasobów i zawartości dzięki zintegrowaniu zewnętrznych narzędzi w skrypcie rekonesansowym. Zwróć uwagę, że większość usług API wymaga utworzenia konta na ich stronie internetowej w celu uzyskania **klucza API**, który służy do uwierzytelnienia użytkowników. Informacje o tym, jak uzyskać klucze API popularnych usług zwiadowczych, znajdziesz na stronie <https://github.com/lanmaster53/recon-ng-marketplace/wiki/API-Keys/>.

Zacznij hakować!

Gdy przeprowadzisz już szeroko zakrojony rekonesans, nasuwa się pytanie, co powinieneś zrobić z zebranymi danymi. Odpowiedź jest prosta: zaplanuj swoje ataki, wykorzystując zgromadzone informacje! Swoim testom nadawaj priorytety, opierając się na funkcjonalności aplikacji i jej technologii.

Jeżeli znajdziesz np. funkcjonalność przetwarzania numerów kart kredytowych, możesz najpierw poszukać luk w zabezpieczeniach, np. IDOR (zobacz rozdział 10.), które mogą powodować wycieki tych numerów kart. Skoncentruj się na wrażliwych funkcjonalnościach, takich jak karty kredytowe i hasła, ponieważ mogą one zawierać krytyczne luki w zabezpieczeniach. Podczas rekonesansu powinieneś być w stanie wyrobić sobie dobre wyobrażenie o tym, na czym zależy firmie i jakie poufne dane chroni. Idź śladem tych konkretnych informacji podczas całego procesu tropienia błędów, aby odkryć problemy, które mają jak największy wpływ na prowadzoną działalność biznesową. Podczas wyszukiwania można również skoncentrować się na błędach lub lukach wpływających na ten odkryty stos technologiczny lub na elementach kodu źródłowego, które udało Ci się znaleźć.

Nie zapominaj także, że rekonesans to nie jest działanie jednorazowe. Powinieneś stale monitorować swoje cele pod kątem zmian. Organizacje nieustannie modyfikują swoje systemy, technologie i bazę kodów, więc ciągły rekonesans zapewni, że zawsze będziesz wiedział, jak wygląda powierzchnia ataku. Korzystając z kombinacji powłoki bash, narzędzi do planowania i narzędzi ostrzegania, zbuduj silnik zwiadowczy, który wykona większość pracy za Ciebie.

Narzędzia wymienione w tym rozdziale

W tym rozdziale przedstawiłam wiele narzędzi, które można wykorzystać w procesie zwiadowczym. Dostępnych jest oczywiście o wiele więcej dobrych narzędzi. Te, o których wspomniałam, to jedynie moje osobiste preferencje. Zamieściłam je tutaj w porządku chronologicznym dla celów referencyjnych.

Pamiętaj, aby przed użyciem tych narzędzi zapoznać się ze sposobem ich działania! Zrozumienie stosowanego oprogramowania pozwala dostosować je do własnego przepływu pracy.

Wykrywanie zakresu

- WHOIS umożliwia odszukanie właściciela domeny lub adresu IP.
- Odwrotne WHOIS (<https://viewdns.info/reversewhois/>) to narzędzie do wyszukiwania odwrotnych danych WHOIS za pomocą słowa kluczowego.
- nslookup kwerenduje serwery nazw internetowych w celu uzyskania informacji o adresie IP hosta.

- Odwrotne IP (<https://viewdns.info/reverseip/>) wyszukuje domeny hostowane na tym samym serwerze dla podanego adresu IP lub domeny.
- crt.sh (<https://crt.sh/>), Censys (<https://censys.io/>) i Cert Spotter (<https://ssllmate.com/certspotter/>) to platformy, których można użyć do szukania informacji o domenie w certyfikatach.
- Sublist3r (<https://github.com/about3la/Sublist3r/>), SubBrute (<https://github.com/TheRook/subbrute/>), Amass (<https://github.com/OWASP/Amass/>) i Gobuster (<https://github.com/OJ/gobuster/>) enumerują poddomeny.
- SecLists Daniela Miesslera (<https://github.com/danielmiessler/SecLists/>) to lista słów kluczowych, które można wykorzystać podczas różnych faz rekonesansu i hakowania. Zawiera np. listy, które mogą być używane do brute forcingu poddomen i ścieżek plików.
- Commonspeak2 (<https://github.com/assetnote/commonspeak2/>) generuje listy, które mogą być wykorzystywane do brute forcingu poddomen i ścieżek plików przy użyciu publicznie dostępnych danych.
- Altdns (<https://github.com/infosec-au/altdns/>) służy do brute forcingu poddomen za pomocą permutacji popularnych nazw poddomen.
- Nmap (<https://nmap.org/>) i Masscan (<https://github.com/robertdavidgraham/masscan/>) skanują cele pod kątem otwartych portów.
- Shodan (<https://www.shodan.io/>), Censys (<https://censys.io/>) i Project Sonar (<https://www.rapid7.com/research/project-sonar/>) mogą być używane do wyszukiwania usług w obiektach docelowych bez ich aktywnego skanowania.
- Dirsearch (<https://github.com/maurosoria/dirsearch/>) i Gobuster (<https://github.com/OJ/gobuster/>) to katalogi wykorzystywane przez narzędzia brute force do znajdowania ukrytych ścieżek plików.
- EyeWitness (<https://github.com/FortyNorthSecurity/EyeWitness/>) i Snapper (<https://github.com/dxa4481/Snapper/>) pobierają zrzuty ekranu na podstawie listy adresów URL. Można je wykorzystać do szybkiego skanowania interesujących stron z listy enumerowanych ścieżek.
- OWASP ZAP (<https://owasp.org/wv-project-zap/>) to narzędzie bezpieczeństwa, które zawiera m.in. skaner i serwer proxy. Jego pająk internetowy może być używany do wykrywania zawartości serwerów WWW.
- GrayhatWarfare (<https://buckets.grayhatwarfare.com/>) to wyszukiwarka online, za pomocą której można znaleźć publiczne wiaderka S3 Amazona.
- Lazys3 (<https://github.com/naHamsec/lazys3/>) i Bucket Stream (<https://github.com/eth0izzle/bucket-stream/>) oferują brute forcing wiaderek przy użyciu słów kluczowych.

Biały wywiad

- Google Hacking Database (<https://www.exploit-db.com/google-hacking-database/>) zawiera przydatne hasła wyszukiwania Google'a, które często ujawniają luki w zabezpieczeniach lub poufne pliki.
- KeyHacks (<https://github.com/streaak/keyhacks/>) pomaga zweryfikować poprawność zestawu poświadczeń i dowiedzieć się, jak z nich korzystać, aby uzyskać dostęp do usług obiektu docelowego.
- Gitrob (<https://github.com/michenriksen/gitrob/>) znajduje potencjalnie poufne pliki, które są przesyłane do publicznych repozytoriów GitHuba.
- TruffleHog (<https://github.com/trufflesecurity/truffleHog/>) specjalizuje się w wyszukiwaniu sekretów w publicznych repozytoriach GitHuba przez wyszukiwanie wzorców tekstowych i łańcuchów znaków o wysokiej entropii.
- PasteHunter (<https://github.com/kevthehermit/PasteHunter/>) skanuje pod kątem poufnych informacji witryny internetowe służące do wklejania danych.
- Wayback Machine (<https://archive.org/web/>) to cyfrowe archiwum treści internetowych. Można go użyć do znalezienia starych wersji witryn i ich plików.
- Waybackurls (<https://github.com/tomnomnom/waybackurls/>) pobiera adresy URL z Wayback Machine.

Fingerprinting stosu technologicznego

- Baza danych CVE (https://cve.mitre.org/cve/search_cve_list.html) zawiera ujawnione luki w zabezpieczeniach. Możesz skorzystać z jej strony internetowej, aby wyszukać luki, które mogą pozwolić na eksploatację Twojego celu.
- Wappalyzer (<https://www.wappalyzer.com/>) identyfikuje systemy zarządzania zawartością, frameworki i języki programowania używane w danej witrynie.
- BuiltWith (<https://builtwith.com/>) to witryna internetowa, która pokazuje, z wykorzystaniem jakich technologii internetowych została zbudowana docelowa witryna.
- StackShare (<https://stackshare.io/>) to platforma internetowa, która umożliwia programistom udostępnianie wykorzystywanych przez nich technologii. Możesz jej używać do zbierania informacji o swoim celu.
- Retire.js (<https://retirejs.github.io/retire.js/>) wykrywa przestarzałe biblioteki JavaScriptu i pakiety Node.js.

Automatyzacja

- Git (<https://git-scm.com/>) to dostępny na licencji open source system kontroli wersji. Możesz skorzystać z jego polecenia `git diff`, by śledzić zmiany w plikach.

Poznałeś teraz solidne podstawy dotyczące przeprowadzania rekonesansu obiektów docelowych. Pamiętaj, aby w trakcie całego procesu rozpoznania robić obszernie notatki, gdyż zbierane informacje naprawdę mogą z czasem zwiększyć znacznie objętość. Gdy zyskasz już trochę doświadczenia w przeprowadzaniu rozpoznania celu i dobrze zrozumiesz ten proces, możesz spróbować wykorzystać platformy zwiadowcze, takie jak Nuclei (<https://github.com/projectdiscovery/nuclei/>) lub Intrigue Core (<https://github.com/intrigueio/intrigue-core/>), aby usprawnić rekonesans. Jednak na początek zalecam ręczny rekonesans za pomocą indywidualnych narzędzi lub pisanie własnych zautomatyzowanych skryptów rozpoznania, aby lepiej poznać ten proces.

Skorowidz

A

adres

- e-mailowy, 143
- IP, 56, 89, 91
- IPv6, 56, 267
- URL, 66, 165
 - bezwzględny, 167
 - danych, 172
 - walidacja, 166, 170
 - wewnętrzny, 262
 - względny, 167

adresy niezroutowane, 274

aktywności, activity, 408

alternatywa, |, 87

analiza statyczna, 437

Android

- Android Studio, 410
- Debug Bridge, 409
- hakowanie aplikacji, 404
- składniki aplikacji, 408

API, Application Programming

Interface, 27, 413

enumeracja, 421

testowanie, 423, 425

tropienie błędów, 420

uzyskiwanie kluczy, 134

API-centric application, 420

APK, Android Package Kit, 407

Apktool, 410

aplikacje

- internetowe ogólne, 26
- mobilne, 26
 - tropienie luk w zabezpieczeniach, 411
- skoncentrowane na API, 420
- społecznościowe, 25

atak

CSRF, 192

DoS, 306

łańcucha dostarczania

oprogramowania, 340

miliardów śmiechów, 307

open redirect, 169

przebieranie interfejsu

użytkownika, 184

SSRF, 303

SSTI, 310

trawersacji ścieżek, 379

typu „człowiek pośrodku”, 407

wstrzyknięcia SQL, 230

XInclude, 301

XSS, 142, 146

eskalacja, 161

odbijany, Reflected XSS, 148

omijanie ochrony, 159

oparty na modelu DOM,

DOM-based XSS, 148

ślepy, Blind XSS, 147

własny, Self-XSS, 150

zapisywany, Stored XSS, 146

zapobieganie, 150

XXE, 294

autokorekta przeglądarki, 170

automatyzacja

ataku IDOR, 226

eksploatacji, 290

skanowania, 132

tropienia błędów XSS, 162

wstrzyknięć SQL, 246

wstrzyknięć szablonu, 324

wykrywania luk

w zabezpieczeniach, 427

autoryzacja, 366

B

baner usługi, 263

baza danych, 229

NoSQL, 242

bezpieczeństwo w internecie, 60

biały wywiad, 102

blob, 384

błędna konfiguracja CORS-u, 356

błędy

API, 420

clickjackingu, 186

CSRF, 196, 198

czasu kontroli, 250

IDOR, 215, 218

JSONP, 358

logiki aplikacji, 326, 331

logiki filtra, 161

niezabezpieczonej deserializacji, 292

obejścia SOP-u, 355

open redirect, 166

postMessage, 357

przejęcia poddomeny, 370

RCE, 340

reguły

tego samego pochodzenia, 347

SAML-a, 373

SSRF, 259, 260, 270

ujawnienia informacji, 379

- wstrzyknięcia
 - obiektu PHP, 281
 - SQL, 237
 - szablonu, 315
- XSS, 152
- XXE, 297
- bomba XML, 307
- brute force, 65
- brute forcing
 - katalogów, 94
 - uwierzytelniania, 434
- Bucket Stream, 99
- budowanie
 - interaktywnych programów, 126
 - partnerstwa, 45
- bug slump, 50
- BuiltWith, 105
- Burp Suite, 73, 75

C

- CA, Certificate Authority, 73
- certyfikat
 - SSL, 91
 - urzędu certyfikacji, 73
- commit, 384
- comparer, 81
- CORS, Cross-Origin Resource Sharing, 349
 - eksploatacja, 349
 - szukanie błędnej konfiguracji, 356
- CPU, central processing unit, 248
- crawler, 96
- cron, 132, 133
- cross-site scripting, 141
- CSRF, Cross-Site Request Forgery, 192
 - brakujące zabezpieczenia, 199
 - dostarczanie ładunku, 212
 - eskalacja ataku, 209
 - luka w zabezpieczeniach, 200
 - obejście tokenów, 203
 - omijanie
 - kontroli nagłówka, 207
 - ochrony, 201, 209
 - podwójnie przesyłanych tokenów, 205

- tokeny, 196
- tropienie błędów, 198
- wykorzystanie luki, 209–211
- zapobieganie, 196
- cudzysłów, 87
- curl, 114
- CVE, Common Vulnerabilities and Exposures, 104
- CVSS, Common Vulnerability Scoring System, 38
- czas reakcji programu, 32

D

- dane wprowadzone przez użytkownika, 398
- data skanowania, 111
- decoder, 81
- definicja typów dokumentu, DTD, 295
- deserializacja, deserialization, 277, 288
- Dirsearch, 110, 111
- DNS, Domain Name System, 56
 - oszukiwanie serwera, 268
- DOM, Document Object Model, 148
 - XSS, 148, 149
- DoS, Denial-of-Service, 306
- dostarczanie ładunku CSRF-a, 212
- dostawca
 - tożsamości, 363
 - treści, content provider, 408
 - usług, 363
- dostęp
 - do powłoki internetowej, 245
 - do systemu, 319
- DTD, Document Type Definition, 295
- działania zmieniające stan, 186
- działanie
 - fuzera internetowego, 428
 - internetu, 55
 - ładunku, 158
 - OAuth, 366
 - SAML-a, 363

E

- eksfiltracja danych, 307
- eksploatacja
 - błędu wstrzyknięcia obiektu PHP, 281
 - dekodowania URL, 173
 - JSONP, 353
 - logiki wadliwego walidatora, 171
 - luki CSRF, 194
 - łączenie technik, 175
 - mechanizmu CORS, 349
 - metody postMessage(), 351
 - ślepych SSRF, 274
 - za pomocą narzędzia Ysoserial, 290
- eksploity, 324
 - RCE, 334
- emulator, 27
- encja
 - exfiltrate, 304
 - zewnętrzna XML-a, *Patrz* XXE
- encje parametryczne, 304
- enumeracja API, 421
 - poddomen, 92
 - ścieżek, 433
 - usług, 93
- eskalacja ataku
 - clickjackingu, 189
 - CSRF, 209
 - IDOR, 226
 - niezabezpieczonej deserializacji, 293
 - open redirect, 175
 - RCE, 343
 - SQL injection, 244
 - SSRF, 270
 - wstrzyknięcia szablonu, 318
 - XXE, 302
- błędów
 - logiki aplikacji, 332
 - obejścia SOP-u, 358
 - omijania SSO, 375
 - uszkodzonej kontroli dostępu, 332
 - sytuacji wyścigu, 255
- EyeWitness, 96

F

falszowanie żądań
przesyłanych między
witrynami, *Patrz* CSRF
wykonywanych po stronie
serwera, *Patrz* SSRF
filetype, 86
fingerprinting, 103
Firefox, 70
firewall aplikacji internetowej,
340
formularz HTML-a, 193
frame-busting, 187, 188
framework
Django, 102
Flask, 102
Mobile Security
Framework, 411
Node.js, 102
Frida, 410
full stack, 102
funkcja unserialize(), 281, 282
funkcje
potencjalnie podatne
na ataki, 394
Pythona, 320
funkcjonalności debugowania,
397
fuzzer internetowy, 428
fuzzing, 427–431, 437
pułapki, 438
za pomocą Wfuzza, 433,
436, 437

G

gadżet, gadget, 284
getopts, 120, 121
Git, 383
GitHub, 35, 101, 134
Google dorking, 85, 167
GraphQL, 417
GrayhatWarfare, 99
grep, 116, 117, 392

H

HackerOne, 141
hakowanie, 135
aplikacji mobilnych, 404
interfejsów API, 413
stron internetowych
konfiguracja środowiska,
68
hostowanie zewnętrzne, 98
HTML, Hypertext Markup
Language, 55
HTTP, HyperText Transfer
Protocol, 27, 58

I

identyfikatory
kodowane, 222
mieszane, 222
przekazywanie, 223
sesji, 61
wyciekające, 223
IDOR, insecure direct object
reference, 215
automatyzacja ataku, 226
eskalacja ataku, 226
omijanie ochrony, 221
oparty na odczycie, 226
oparty na zapisie, 226
ślepy, 224
tropienie błędów, 218
zapobieganie, 217
indeksowanie witryn, 96
inicjowanie ataków SSRF, 303
inspekcja kodu źródłowego,
391, 401
instrukcja
case, 113
if-else, 113
instrukcje przygotowane, 234
interfejs programistyczny
aplikacji, API, 27, 413
API SOAP, 416
GraphQL API, 417
hakowanie, 413
rekonesansu, 134
RESTful API, 415

internet, 55
kontrola bezpieczeństwa, 60
internet rzeczy, IoT, 26, 28
intitle, 86
Intrigue Core, 138
intruder, 78
intruder Burpa
lista ładunków, 432
inurl, 86
inżynier full stack, 102
inżynieria społeczna, 150, 165
IoT, Internet of Things, 26, 28
IP, Internet Protocol, 56

J

Java, 288
JavaScript, 56
alternatywna składnia, 159
język
HTML, 142
Java, 288
JavaScript, 56
PHP, 278
SAML, 294, 363
SQL, 229
XML, 294
JSON, JavaScript Object
Notation, 27, 415
z wypełnianiem, JSONP,
353
JSONP, JSON with Padding,
353
eksploatacja, 353
JWT, JSON Web Token, 63

K

Kali Linux, 69
katalog .git, 382, 383
Kibana, 88
klasa
bazowa, base class, 321
CodeSnippet, 285
list, 321
klienty, 55
klucz API, 134

kod
 statusu 404, 95
 wstrzyknięcia szablonu, 312
 źródłowy, 27
kodowanie, 159
 base64, 60
 base64url, 60
 heksadecymalne, 61
 ósemkowe, 269
 URL, 61
komentarze programistów, 397
komunikaty
 HTTP, 57
 HTTPS, 57
konfigurowanie
 Burpa, 73
 przeglądarki Firefox, 70
 serwera proxy, 405
 środowiska, 68
konflikty, 45
kwoty wypłat, 31

L

linia
 shebang, 106
 zadania HTTP, 58
link, 86
lista
 elementów blokowanych,
 259, 267
 elementów dozwolonych,
 260, 265
 ładunków, 430
 ładunków w intruderze
 Burpa, 432
logika filtra, 161
luka
 automatyczne wykrywanie,
 427
 CSRF, 198
 IDOR, 217
 poza zakresem, 31
 SSRF, 257
 trawersacji katalogów, 330
 w logice biznesowej, 327
 w zabezpieczeniach
 OAuth, 368
 SAML-a, 364
 w zakresie, 31

wstrzyknięcia
 kodu, 335
 obiektu PHP, 279
 polecenia, 336
XSS, 141
załączenia pliku zdalnego,
 337

Ł

ładunek
 clickjackingu, 190
 CSRF-a, 212
 intrudera Burpa, 431
 PHP, 342
 potwierdzanie działania, 158
 Pythona, 341
 uniksowy, 342
 XSS, 153, 157
 XXE, 300
łańcuch otwartych
 przekierowań, 369
łańcuchy POP, 284

M

metadane
 instancji, 272
 usługi EC2, 272
 usługi Google Cloud, 273
metoda
 __destruct(), 281
 __wakeup(), 281
 evaluate(), 285
 postMessage(), 351, 357
metody magiczne PHP, 281, 283
MFA, Multifactor
 Authentication, 327
minus, -, 87
Mobile Security Framework, 411
model
 DOM, 148
 klient-serwer, 55

N

nagłówek, 63
referencyjny, 207
odpowiedzi, 187
zadania HTTP, 58

narzędzia służące
 do automatyzacji, 138
 białego wywiadu, 137
 fingerprintingu stosu
 technologicznego, 137
 wykrywania zakresu, 135
 zautomatyzowanego
 testowania, 438

narzędzie

 Amass, 92
 Android Debug Bridge, 409
 Apktool, 410
 Bucket Stream, 99
 Censys, 94
 comparer, 81
 crawler, 96
 cron, 132, 133
 decoder, 81
 Dirsearch, 95
 EyeWitness, 96
 Frida, 410
 getopts, 120, 121
 Git, 383
 Gitrob, 102
 Gobuster, 92, 95
 GrayhatWarfare, 99
 grep, 116, 117, 392
 intruder, 78
 KeyHacks, 101
 Kibana, 88
 Masscan, 93
 Nmap, 93
 OWASP ZAP, 96, 98
 pastebin-scraper, 382
 PasteHunter, 382
 Project Sonar, 94
 repeater, 80
 Retire.js, 105
 Shodan, 94
 SSRFmap, 264
 SubBrute, 92
 Sublist3r, 92, 439
 TruffleHog, 102
 Waybackurls, 103
 Wfuzz, 433, 436, 437
 wget, 336
 wiersza polecenia, 92
 Ysoserial, 290
nauka programowania, 67
nazwa domeny, 57
nieaktualne zależności, 396

niebezpieczne wzorce, 392
niezabezpieczona
 deserializacja, insecure
 deserialization, 277
 eskalacja ataku, 293
 tropienie błędów, 292
 zapobieganie, 291
niezabezpieczone
 bezpośrednie odwołania do
 obiektów, *Patrz* IDOR
Nmap, 104, 110, 111
notatki, 82
Nuclei, 138
numery systemów
 autonomicznych, 91

O

OAuth, 366
 działanie, 366
 luki w zabezpieczeniach, 368
 tokeny, 375
 tropienie kradzieży
 tokenów, 375
obiektywny model dokumentu,
 DOM, 148
obiekty JSON, 415
odbijany XSS, Reflected XSS,
 148
odbiorcy komunikatów,
 broadcast receiver, 408
odczytywanie
 plików, 302
 poufnych informacji, 66
odgadywanie klucza, 65
odnoszenie sukcesu, 46
odpowiedź HTTP, 59
odwrotne
 WHOIS, 89
 wyszukiwanie adresów IP, 90
ogólne aplikacje internetowe, 26
omijanie
 list elementów
 blokowanych, 267
 dozwolonych, 265
ochrony przed atakami
 CSRF, 201, 209
 IDOR, 221
 open redirect, 169

RCE, 344
SSRF, 265
XSS, 159
przypinania certyfikatu, 407
SOP-u, 355
walidatora otwartych
 przekierowań, 171
zabezpieczeń clickjacking,
 188
opcje wyboru, 111
open redirect
 tropienie błędów, 166
otwarte przekierowania, open
 redirect, 164
 eskalacja ataku, 175
 omijanie ochrony, 169
 oparte na odsyłaczach, 169
 oparte na parametrach, 169
 tropienie błędów, 166
 zapobieganie, 166
OWASP, Open Web Application
 Security Project, 51
OWASP ZAP, 96, 98

P

pająk internetowy, 96
pakiet APK, 407
panele administracyjne, 329
parametry przekierowania, 166,
 167
parsowanie
 certyfikatu, 91
 wyników, 116
partnerstwo, 45
Pastebin, 103, 382
pętla while, 127, 128
phishing, 165
piaskownica, 320
pisanie
 biblioteki funkcji, 125
 raportów, 36
planowanie, scheduling, 249
platforma
 bug bounty, 24, 28
 HackerOne, 32
plik AndroidManifest.xml, 408
pliki
 cookie, 61, 193, 361
 JavaScriptu, 56
 konfiguracyjne, 397
pobieranie plików, 383
PoC, Proof-of-Concept, 40, 283
podpisy, 63
podstawianie poleceń, 111
podwójne kodowanie, 173
pojedyncze logowanie, SSO, 360
pole alg, 64
polecenie
 curl, 114
 gopts, 127
 grep, 392
 quit, 127
 read, 126
 run-parts, 133
 source, 125
 test, 123
 touch, 323
 wget, 336
 whois, 90
poliglot XSS-a, 157
pomijalny etap
 uwierzytelniania, 327
POP, Property-Oriented
 Programming, 284
poprawki kodu, 396
porównanie programów bug
 bounty, 34
port internetowy, 57
 443, 57
 80, 57
porywanie kliknięć,
 clickjacking, 178
 eskalacja ataku, 189
 obchodzenie zabezpieczeń,
 188
 tropienie błędów, 186
 zapobieganie, 184
potwierdzanie działania
 ładunku, 158
powierzchnia ataku, 25, 85
powłoka odwrócona, reverse
 shell, 336
poziom uprawnień, 253, 340
proces
 fuzzingu, 429
 uwierzytelniania SAML-a, 363

- program
 - Facebook, 34
 - GitHub, 35, 101, 134
 - HackerOne, 34
 - Burp Suite Community Edition, 73
 - ujawniania luk
 - w zabezpieczeniach, VDP, 31
- programy
 - bug bounty, 23, 33
 - prywatne, 32
 - publiczne, 32
 - wykonywalne, 27
- protokół HTTP, 58
- proxy, 75
- przebieranie interfejsu użytkownika, 184
- przechwytywanie ruchu, 68
- żądań, 220
- przełęczarka, 70
 - Firefox, 70
 - używanie autokorekty, 170
- przejęcie poddomeny, subdomain takeover, 361
 - ciągłe monitorowanie, 372
 - tropienie błędów, 370
- przekierowanie, 267
 - wejścia, 109
 - wyjścia, 109
- przesyłanie ładunków, 323
- przetwarzanie wieloprocesorowe, 248
- przyпинanie certyfikatów, certificate pinning, 407
- SSL, 407
- punkty wstrzyknięcia danych, 429

R

- ramka iframe, 178–184, 195
 - podwójna, 188
- raport
 - dowód koncepcji, 40
 - główny
 - tworzenie, 118
 - ocena dotkliwości błędu, 37
 - odtworzenie ataku, 39

- opisowy tytuł, 37
- przejrzyste podsumowanie, 37
- rekomendacja, 41
- scenariusze ataku, 40
- sprawdzenie, 41
- status
 - informacyjny, 43
 - nie dotyczy, 44
 - rozwiązany, 44
 - triażowany, 44
 - wymaga uzupełnienia, 43
 - zduplikowany, 44
- RCE, Remote Code Execution, 282, 290, 334
 - eskalacja ataku, 343
 - omijanie ochrony, 344
 - przesyłanie ładunku, 341
 - tropienie błędów, 340
 - załączenie pliku, 337
- referer, 165
- reguła tego samego pochodzenia, SOP, 66
- rekonesans, 84, 421
- rekord CNAME, 362, 372
- relacje
 - z zespołem programistyczny m, 43
- repeater, 80
- Retire.js, 105
- rodzaje ataków XSS, 146
- ROP, Return-Oriented Programming, 288

S

- S3, Simple Storage Service, 98
- SAML, Security Assertion Markup Language, 294, 363
 - działanie, 363
 - luki w zabezpieczeniach, 364
 - proces uwierzytelniania, 363
 - tropienie błędów, 373
- sanityzacja danych wejściowych, 144
- SAST, Static Application Security Testing, 403
- sekcja
 - ładunku, 64
 - podpisu, 64
- serializacja, serialization, 277, 288
- serwer, 55
 - DNS, 56
 - proxy, 69
 - dla urządzeń mobilnych, 405
 - Burp Suite, 73, 75
 - Burpa, 405
- serwisy, 25
- sesja, 61
- Shodan, 94
- silnik szablonów, 311, 317
- site, 86
- skanowanie
 - aktywne, 93
 - automatyczne, 132
 - domen, 120
 - internetu, 96
 - pasywne, 94
 - portów, 274
 - sieci, 270, 274
- składnia alternatywna JavaScriptu, 159
- skrypt Dirsearcha, 107
- skrypty
 - bashowe, 106
 - rekonesansowe, 106
 - wstawiane, 143
- SlideShare, 103
- słabe szyfrowanie, 395
- SOAP, Simple Object Access Protocol, 416
- socjotechnika, 165
- SOP, Same-Origin Policy, 66, 347
 - eskalacja ataku, 358
 - omijanie, 355
 - tropienie błędów obejścia, 355
- sort, 92
- sprzęt, 28
- SQL, Structured Query Language, 229
- SQL injection, 228
- SSL, Secure Sockets Layer, 91
- SSO, Single Sign-On, 360
 - eskalacja ataku, 375
 - tropienie luk
 - w zabezpieczeniach, 370

SSRF, Server-Side Request Forgery, 257

- eksploatacja, 274
- eskalacja ataku, 270
- funkcjonalności podatne na ataki, 260
- inicjowanie ataków, 303
- omijanie zabezpieczeń, 265
- sprawdzanie wyników, 263
- ślepe, 263
- wewnętrzne adresy URL, 262
- zapobieganie, 259

SSRFmap, 264

SSTI, Server-Side Template Injection, 310

StackShare, 105

stany raportu, 43

statyczna analiza bezpieczeństwa aplikacji, SAST, 403

sterowanie wartościami zmiennych, 280

Sublist3r, 439

sukces, 46

symbol wieloznaczny, *, 87, 130, 350

system

- kontroli wersji, 383
- nazw domenowych, DNS, 56, 268
- operacyjny Kali Linux, 68

sytuacja wyścigu, race condition, 248

- eskalacja, 255
- kontrola bezpieczeństwa, 250
- nieprawidłowe obliczenia, 250
- tropienie sytuacji, 253
- zapobieganie, 253

sztuczka z podwójną ramką iframe, 188

szukanie błędów

- API, 420
- clickjackingu, 186
- CSRF, 198
- IDOR, 218
- logiki aplikacji, 331, 333
- niezabezpieczonej deserializacji, 292

- obejścia SOP-u, 355
- open redirect, 166
- przejęcia poddomeny, 370
- RCE, 340
- SAML-a, 373
- SSRF, 260
- ujawnienia informacji, 379
- uszkodzonej kontroli dostępu, 331
- w aplikacjach mobilnych, 411
- wstrzyknięcia SQL, 237
- wstrzyknięcia szablonu, 315
- XSS, 152, 162
- XXE, 297

pierwszego błędu clickjackingu, 191

- CSRF, 214
- IDOR, 227
- obejścia SOP-u, 359
- obejścia SSO, 376
- otwartego przekierowania, 177
- RCE, 346
- SSRF, 276
- ujawnienia informacji, 387
- uszkodzonej kontroli dostępu, 333
- wstrzyknięcia SQL, 246
- wstrzyknięcia szablonu, 324
- XSS, 163
- XXE, 309

pierwszej niezabezpieczonej deserializacji, 293

sytuacji wyścigu, 253, 256

Ś

ślepy XSS, Blind XSS, 147

T

tabela wypłat, payout table, 31

tag, 384

techniki eksploatacji, 175

testowanie pod kątem błędów technicznych, 425

- luk w zabezpieczeniach sieciowych, 436
- problemów z ograniczaniem przepustowości, 425
- uszkodzonej kontroli dostępu, 423
- wycieków informacji, 423

testy

- białej skrzynki, 392
- czarnej skrzynki, 392
- fuzzingowe, 427
- szarej skrzynki, 392
- zautomatyzowane, 438

token

- CSRF, 196, 203
- przesyłane podwójnie, 205

JSON, 63

- atak brute force, 65
- nagłówkek, 63
- odczytywanie poufnych informacji, 66
- pole alg, 64
- sekcja ładunku, 64
- sekcja podpisu, 64

OAuth, 375

trawersacja katalogów, 330

tree, 384

tryb warunkowy, 111

tworzenie raportu głównego, 118

U

ucieczka, escaping, 151

ujawnienie informacji, information disclosure, 377

- eskalacja ataku, 387
- zapobieganie, 378

uprawnienia, 253

- najniższego poziomu, 340

uruchamianie narzędzi, 111, 112

usługa, service, 93, 408
 EC2, 272
 Google Cloud, 273
 proxy, 261
uszkodzona kontrola dostępu, 329
 eskalacja ataku, 332
uwierzytelnianie
 brute forcing, 434
 oparte na tokenach, 62
 SAML-a, 363
 wielopoziomowe, MFA, 327

V

VDP, Vulnerability Disclosure Program, 31

W

walidacja danych wejściowych, 144
walidator
 adresu URL, 166, 170
 otwartych przekierowań, 171
Wappalyzer, 105
Wayback Machine, 103, 104, 380
ważne funkcje, 398
wątek, 249
Wfuzz, 433, 436, 437
wget, 336
WHOIS, 89
wiaderko, bucket, 98
wielkość liter, 159
wielowątkowość, 248
własny XSS, Self-XSS, 150
współbieżność, 248
współdzielenie plików cookie, 361
wstrzyknięcie
 kodu, 335
 zapobieganie, 339
 NoSQL, 242
 obiektu PHP, 279
 SQL, SQL injection, 228
 automatyzacja, 246
 drugiego rzędu, 233

eksfiltrowanie informacji, 241
eskalacja ataku, 244
klasyczne, 238
mechanizmy, 229
oparte na czasie, 240
ślepe, 239
tropienie błędów, 237
zapobieganie, 234
szablonu
 automatyzacja, 324
 eskalacja ataku, 318
 kod, 312
 ładunek testowy, 316
 po stronie serwera, SSTI, 310
 przesyłanie ładunków, 323
 tropienie błędów, 315
 zapobieganie, 315
wybór programu, 33
wyciekające sekrety, 395
wykorzystanie luki CSRF, 209–211
wykrywanie zakresu, 89
wyrażenia regularne, 116
wyrażenie, 311
wyszukiwarka
 GrayhatWarfare, 99
 Shodan, 94
wzorzec regex, 118

X

XInclude, 302
XML, Extensible Markup Language, 27, 294
XSS, Cross-Site Scripting, 141
 automatyzacja tropienia błędów, 162
 omijanie ochrony przed CSRF, 209
 omijanie SOP-u, 355
 oparty na modelu DOM, 148
 rodzaje ataków, *Patrz* atak XSS, 142, 146
 tropienie błędów, 152
XXE, External Entity Attack, 66

XXE, XML external entity, 294
 eksfiltracja danych, 307
 eskalacja ataku, 302
 klasyczne, 298
 ślepe, 299, 303
 tropienie błędów, 297
 zapobieganie, 296

Y

Ysoserial, 290

Z

zaatakowanie sieci, 275
zaczepy sieciowe, webhook, 260
zakres, 366
 IP, 90
 programu, 30, 89
 zasobów, 30
załączenie pliku lokalnego, 337, 338
zamykanie znaczników, 156
zapisywanie
 w pliku, 109
 żądań Burpa, 82
zapisywany XSS, Stored XSS, 146
zapobieganie
 atakami
 deserializacji, 291
 wstrzyknięcia kodu, 339
 wstrzyknięcia SQL, 234
 wstrzyknięcia szablonu, 315
 XSS, 150
 XXE, 296
błędami
 CSRF, 196
 IDOR, 217
 logiki aplikacji, 331
 SSRF, 259
clickjackingowi, 185
otwartym przekierowaniom, 166
sytuacjom wyścigu, 253
wyciekiem poufnych informacji, 378

zapytania sparametryzowane,
234
zapytanie SQL, 235
zarządzanie sesją, 61
zdalne wykonywanie kodu,
Patrz RCE
złośliwa witryna, 67
złośliwy adres URL, 145
zmienianie
 identyfikatorów, 220
 kodowania, 268

 metody żądania, 202, 225
 typu żądanego pliku, 225
zmiennie specjalne, 129
znacznik
 <form>, 194
 <script>, 154
 <style>, 182
znak wieloznaczny, *, 87, 130,
350
znaki inne niż ASCII, 174

Ż

żądania Burpa, 82
żądanie
 DELETE, 58
 GET, 58, 198
 OPTIONS, 58
 POST, 58, 181

PROGRAM PARTNERSKI

— GRUPY HELION —

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Bug bounty: wyśledź, zhakuj, opisz — i zgarnij nagrodę!

Wyśledzenie luki w zabezpieczeniach aplikacji i przejście kontroli nad chronionym zasobem jest wyjątkowo ekscytującym doświadczeniem: oto dzięki własnym umiejętnościom można pokonywać kolejne ograniczenia i osiągać cele, które na pozór wydają się nierealne. Od pewnego czasu takie eksperymenty z hakowaniem można prowadzić całkowicie legalnie, a nawet nieźle na tym zarabiać. Wiele firm uruchamia programy bug bounty, w ramach których nagradza hakerów i badaczy bezpieczeństwa za odnajdywanie luk w zabezpieczeniach w korporacyjnych systemach.

Ta książka jest kompleksowym i praktycznym przewodnikiem po hakowaniu aplikacji internetowych w ramach udziału w programach bug bounty. Znajdziesz w niej wszystkie niezbędne informacje, od budowania relacji z klientami i pisania znakomitych raportów o błędach w zabezpieczeniach po naukę zaawansowanych technik hakerskich. Dowiesz się, jak przygotować własne laboratorium hakarskie i zgłębisz typowe techniki działania, takie jak XSS czy SQL injection. Zapoznasz się również ze strategiami prowadzenia rekonesansu i sposobami jego automatyzacji za pomocą skryptów powłoki bash. Nie zabrakło tu opisu hakowania aplikacji mobilnych, testowania interfejsów API i inspekcji kodu źródłowego pod kątem luk w zabezpieczeniach

Najciekawsze zagadnienia:

- identyfikowanie typowych luk w zabezpieczeniach sieciowych
- praca z pakietem Burp Suite
- kumulowanie wpływu wielu błędów w zabezpieczeniach
- obchodzenie zabezpieczeń metodą sanityzacji danych wejściowych i listy elementów blokowanych
- automatyzacja żmudnych zadań za pomocą fuzzingu i skryptów powłoki bash
- konfiguracja środowiska testowego dla aplikacji pracujących w Androidzie

Vickie Li jest programistką i ekspertką w dziedzinie bezpieczeństwa aplikacji internetowych. Bierze udział w programach bug bounty takich firm jak Facebook, Yelp czy Starbucks. Chętnie dzieli się swoją bogatą wiedzą o technikach poprawy bezpieczeństwa aplikacji i nieoczywistymi tajnikami pracy tropicielki bug bounty.

Helion

KOD KORZYŚCI
Sięgnij po więcej ▶



helion.pl



HELION SA
ul. Kościuski 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-9411-7



9 788328 394117

Cena: 99,00 zł

