HAKOWANIE INTERFEJSÓW API

ŁAMANIE INTERFEJSÓW PROGRAMOWANIA APLIKACJI INTERNETOWYCH

COREY J. BALL

Helion

no starch press

Tytuł oryginału: Hacking APIs: Breaking Web Application Programming Interfaces

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-8322-408-4

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/hakint
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

6

Odkrywanie interfejsów API

Zanim zaczniesz atakować interfejs API, musisz go zlokalizować i sprawdzić, czy działa. Musisz również uzyskać poświadczenia (klucze, nazwy użytkowników, hasła), numer wersji, dokumentację i informacje o przeznaczeniu. Im więcej danych zbierzesz, tym więcej luk w zabezpieczeniach będziesz mógł wykryć i wykorzystać. W tym rozdziale poznasz procesy pasywnego i aktywnego rekonesansu oraz wykorzystywane do tego celu narzędzia.

Przystępując do rozpoznania interfejsu API, warto zastanowić się nad jego przeznaczeniem. Interfejs może być przeznaczony do użytku wewnętrznego, może być otwarty tylko dla partnerów i klientów firmy albo też dostępny publicznie. Interfejs publiczny lub udostępniany tylko partnerom prawdopodobnie posiada dokumentację dla programistów, opisującą jego punkty końcowe i zawierającą instrukcje użytkowania. Korzystaj z takiej dokumentacji podczas rozpoznawania interfejsu.

Jeśli interfejs API jest przeznaczony dla wybranych klientów lub do użytku wewnętrznego, musisz oprzeć się na innych danych, takich jak konwencje nazewnictwa, informacje zawarte w odpowiedziach w nagłówkach HTTP (np. `Content-Type: application/json`), treściach JSON/XML odpowiedzi oraz w wykorzystywanych w aplikacji plikach JavaScript.

Rekonesans pasywny

Rekonesans pasywny to proces uzyskiwania informacji o systemie bez bezpośredniej interakcji z jego urządzeniami. Celem jest określenie i udokumentowanie obszaru ataku bez informowania jego właściciela o wykonywanych czynnościach. **Obszar ataku** to zbiór wszystkich dostępnych systemów w sieci, z których można pozyskać informacje pozwalające po ich odpowiednim wykorzystaniu otworzyć dostęp do innych systemów lub spowodować ich awarię.

Zazwyczaj w rekonesansie pasywnym wykorzystuje się metodykę OSINT (ang. *Open-Source Intelligence*, biały wywiad), polegającą na pozyskiwaniu informacji z publicznie dostępnych źródeł. Szuka się punktów końcowych, poświadczeń, dokumentacji, informacji o wersji i przeznaczeniu biznesowym interfejsu. Znalezione punkty końcowe stają się celami badań w następnym kroku, podczas rekonesansu aktywnego. Dane uwierzytelniające pozwolą Ci testować interfejs z perspektywy uprawnionego użytkownika lub — jeszcze lepiej — administratora. Numer wersji może wskazać potencjalne wrażliwe zasoby i zidentyfikowane luki w zabezpieczeniach. Z dokumentacji dowiesz się dokładnie, jak testować interfejs. Wreszcie znajomość przeznaczenia interfejsu ułatwia odkrywanie błędów w algorytmie biznesowym.

W trakcie przeprowadzania białego wywiadu z dużym prawdopodobieństwem natkniesz się na krytyczne dane, na przykład klucze API, poświadczenia, tokeny JWT, które zapewnią Ci natychmiastowy sukces. Innym poważnym zagrożeniem może być wyciek poufnych danych osobowych użytkowników, takich jak numery PESEL, imiona, nazwiska, adresy e-mail, informacje o kartach kredytowych. Tego rodzaju przypadki musisz natychmiast dokumentować i zgłaszać, ponieważ stanowią krytyczne zagrożenie bezpieczeństwa interfejsu.

Proces rekonesansu pasywnego

Rozpoczynając rekonesans pasywny, prawdopodobnie nie będziesz wiedział nic albo wiedział niewiele o badanym interfejsie. Po zebraniu podstawowych informacji skoncentruj wysiłki na poznaniu różnych aspektów organizacji i na określeniu obszaru ataku. Przeznaczenie każdego interfejsu może być inne w zależności od branży i celów biznesowych, więc będziesz musiał dostosowywać się do napływających nowych informacji. Zacznij od zarzucenia szerokiej sieci, tj. użycia szeregu narzędzi do zbierania danych, a następnie na podstawie zebranych informacji zawężaj kierunek badań. Powtarzaj ten proces, aż uzyskasz mapę obszaru ataku.

Faza pierwsza: zarzucenie szerokiej sieci

Poszukaj w internecie ogólnych informacji o badanym interfejsie API. Wyszukiwarki takie jak Google, Shodan i ProgrammableWeb dostarczą Ci danych takich jak jego przeznaczenie, projekt, architektura, dokumentacja, cel biznesowy, informacje branżowe i wiele innych szczegółów, które mogą okazać się istotne.

Przeprowadź dodatkowe badanie obszaru ataku za pomocą narzędzi takich jak DNS Dumpster i OWASP Amass. Pierwsze z nich tworzy mapę DNS, tj. obraz

hostów powiązanych z badaną domeną wraz z połączeniami między nimi (być może zbadasz je później). Narzędzie OWASP Amass zostało opisane w rozdziale 4.

Faza druga: adaptacja i koncentracja

Na podstawie informacji zebranych w fazie pierwszej dostosuj swoje działania wywiadowcze. Może to oznaczać uszczegółowienie zapytań lub scalenie informacji zebranych za pomocą osobnych narzędzi w celu uzyskania nowych wniosków. Możesz np. poszukać w serwisie GitHub repozytoriów związanych z Twoim celem lub spróbować pozyskać poufne informacje przy użyciu narzędzia Pastehunter.

Faza trzecia: dokumentacja obszaru ataku

Robienie notatek ma kluczowe znaczenie dla przeprowadzenia skutecznego ataku. Dokumentuj i wykonuj zrzuty ekranów ze wszystkimi interesującymi wynikami. Na bazie dotychczasowych ustaleń buduj listę zadań, które mogą Ci się przydać w przyszłych atakach. Później, gdy będziesz aktywnie badał luki w interfejsie API, wrócisz do tej listy i sprawdzisz, czy czegoś nie pominąłeś.

W kolejnych podrozdziałach są przedstawione narzędzia, których będziesz używał w opisanym procesie. Podczas eksperymentowania z nimi zauważysz zależności między dostarczanymi przez nie informacjami. Zachęcam Cię do korzystania z kilku narzędzi, abyś mógł weryfikować wyniki. Gdyby na przykład okazało się, że w serwisie GitHub znajdują się poufne klucze API, cyberprzestępca mógłby je wykorzystać, aby się włamać na konto Twojego klienta.

Hakowanie za pomocą Google

Hakowanie za pomocą Google (ang. *Google dorking*) polega na umiejętnym stosowaniu zaawansowanych parametrów wyszukiwarki w celu pozyskania wszelkiego rodzaju publicznie dostępnych informacji o badanym interfejsie API, takich jak luki w zabezpieczeniach, klucze i nazwy użytkowników, które można wykorzystać podczas testów. Ponadto można w ten sposób zdobywać informacje o branży, w której działa organizacja, i sposobie, w jaki wykorzystuje swoje interfejsy API. Tabela 6.1 zawiera kilka użytecznych parametrów. Ich pełną listę znajdziesz w Wikipedii pod hasłem *Google Hacking*.

Tabela 6.1. Wybrane parametry wyszukiwarki Google

Parametr	Opis
`intitle`	Wyszukiwanie frazy w tytułach stron.
`inurl`	Wyszukiwanie frazy w adresach URL stron.
`filetype`	Wyszukiwanie plików określonego typu.
`site`	Wyszukiwanie frazy w określonych witrynach.

Zacznij od ogólnego wyszukiwania, aby się przekonać, jakie informacje są dostępne. Następnie zastosuj parametry właściwe dla badanego interfejsu i skoncentruj

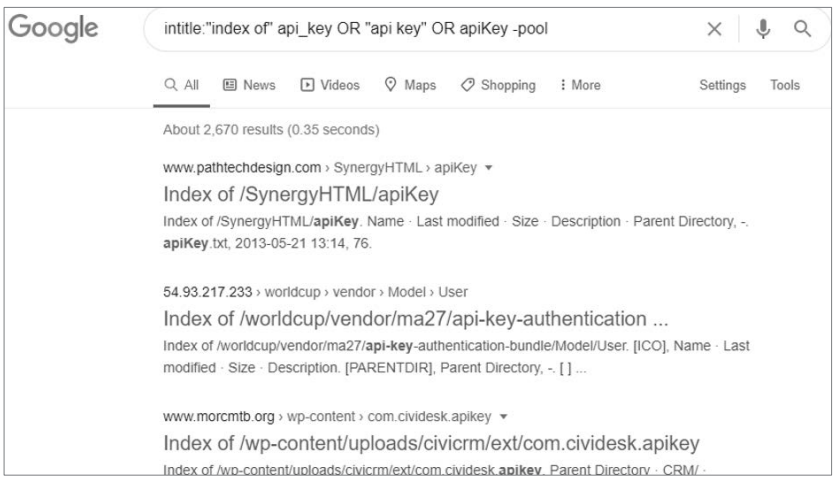
się na wynikach. Na przykład fraza `inurl: /api/` zwraca ponad 2 150 000 wyników — za dużo, aby móc cokolwiek z nimi zrobić. Możesz zawęzić obszar poszukiwań, wprowadzając nazwę domeny. Na przykład fraza `"<nazwa> api key"` może zwrócić mniejszą liczbę bardziej trafnych wyników.

Możesz korzystać z własnych, umiejętnie dobranych fraz lub też z bazy GHDB (Google Hacking Database, *https://www.exploit-db.com/google-hacking-database*) firmy Offensive Security. Jest to zbiór fraz, które mogą ujawnić podatne na ataki systemy i poufne informacje. Tabela 6.2 zawiera kilka przydanych przykładów z tej bazy.

Tabela 6.2. Wybrane frazy z bazy GHDB

Fraza	Wyszukiwane dane
`inurl:"/wp-json/wp/v2/users"`	Publicznie dostępne katalogi WordPress.
`intitle:"index.of" intext:"api.txt"`	Publicznie dostępne pliki z kluczami API.
`inurl:"/includes/api/" intext: "index of /"`	Potencjalnie interesujące katalogi interfejsów API.
`ext:php inurl:"api.php?action="`	Interfejsy XenAPI podatne na wstrzykiwanie zapytań SQL. Ta fraza została opublikowana w 2016 r. Cztery lata później zwracała 141 000 wyników.
`intitle:"index of" api_key OR "api key" OR apiKey -pool`	Publicznie dostępne klucze API (jedna z moich ulubionych fraz).

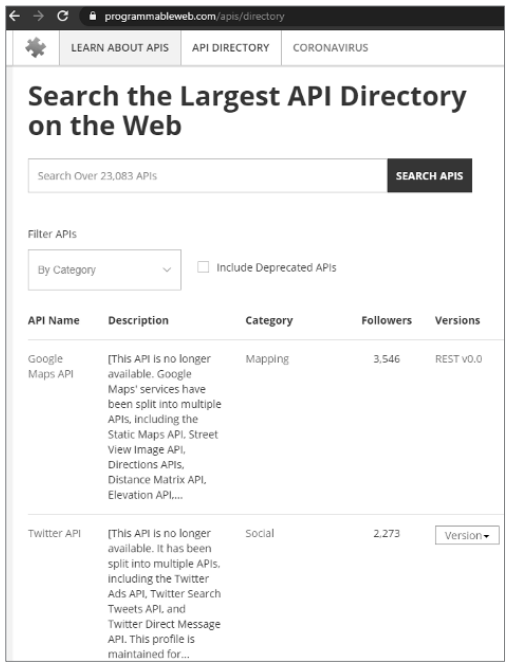
Jak pokazuje rysunek 6.1, ostatnia z wymienionych wyżej fraz zwróciła adresy 2760 stron, na których znajdują się publicznie dostępne klucze API.



Rysunek 6.1. Wyniki hakowania za pomocą wyszukiwarki Google, zawierające strony z publicznie dostępnymi kluczami API

Katalog interfejsów API — ProgrammableWeb

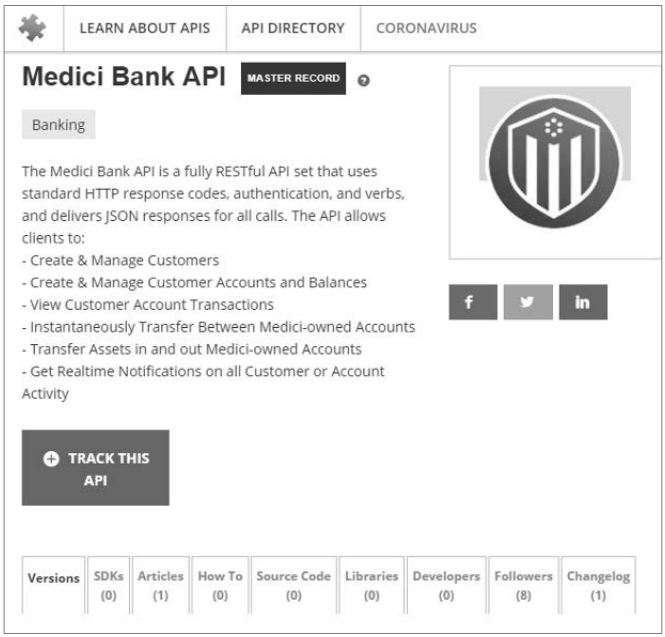
ProgrammableWeb (*https://www.programmableweb.com*) to źródło informacji o interfejsach API. Jeżeli chcesz się dowiedzieć więcej o interfejsach, zajrzyj do sekcji *API University*. W bazie *API DIRECTORY*, zawierającej ponad 23 000 rekordów (patrz rysunek 6.2), znajdziesz różnego rodzaju informacje o badanym interfejsie, takie jak punkty końcowe, numer wersji, algorytm biznesowy, status, kod źródłowy, pakiet SDK, artykuły, dokumentacja i dziennik zmian.



Rysunek 6.2. Baza API DIRECTORY w serwisie ProgrammableWeb

UWAGA *Skrót **SDK** (ang.* Software Development Kit*) oznacza **pakiet rozwoju oprogramowania**. Jeżeli jest dostępny pakiet dla danego interfejsu API, prawdopodobnie można pobrać implementujące go oprogramowanie. W serwisie ProgrammableWeb znajduje się m.in. odnośnik do strony Twitter Ads SDK, na której zamieszczono kod źródłowy, który można pobrać i przetestować.*

Załóżmy, że za pomocą wyszukiwarki Google odkryłeś, że badana organizacja korzysta z interfejsu API banku Medici Bank i że w bazie ProgrammableWeb znalazłeś wpis pokazany na rysunku 6.3.



Rysunek 6.3. Informacje w bazie ProgrammableWeb o interfejsie API banku Medici Bank

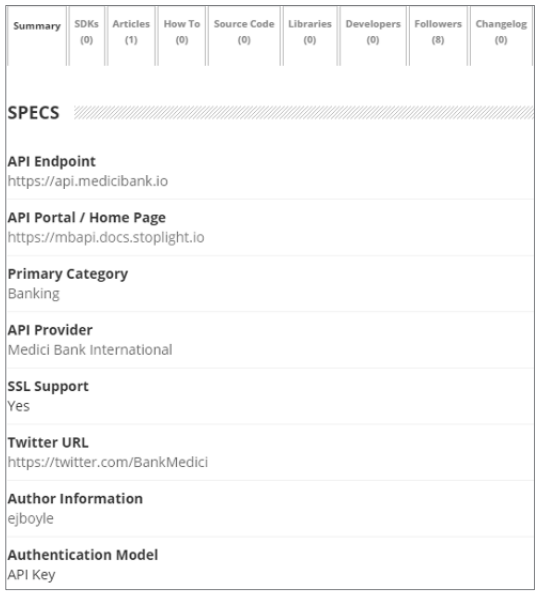
Z opisu wynika, że interfejs wchodzi w interakcje z danymi klientów i upraszcza realizację transakcji finansowych. Oznacza to, że jest to interfejs wysokiego ryzyka. Po znalezieniu tak podatnego interfejsu jak ten poszukaj informacji, które będziesz mógł wykorzystać do przeprowadzenia ataku, czyli dokumentacji, punktów końcowych, portalu, kodu źródłowego, dziennika zmian i zastosowanego modelu uwierzytelniania użytkowników.

Przejrzyj poszczególne zakładki na stronie i zanotuj dostępne informacje. Aby zobaczyć lokalizację punktu końcowego interfejsu, lokalizację portalu i model uwierzytelniania, kliknij wybraną wersję w zakładce *Versions*, jak na rysunku 6.4. W tym przykładzie odnośniki zarówno do portalu, jak i do punktów końcowych prowadzą również do dokumentacji interfejsu.

W zakładce *Changelog* (dziennik zmian) znajdziesz informacje o zidentyfikowanych wcześniej lukach w zabezpieczeniach, poprzednich wersjach i najważniejszych aktualizacjach (jeżeli są dostępne).

W zakładce *Libraries* (biblioteki) umieszczone są, według opisu na stronie, „narzędzia właściwe dla platformy, które po zainstalowaniu udostępniają określony interfejs API". Użyj tej zakładki, aby dowiedzieć się o oprogramowaniu wykorzystywanym przez dany interfejs, np. podatnych na ataki bibliotekach.

W zależności od interfejsu API możesz znaleźć jego kod źródłowy, podręczniki (w zakładce *How To* — Jak to zrobić) i artykuły, które mogą zawierać przydatne informacje wywiadowcze. Repozytoria interfejsów API znajdują się również na stronach *https://rapidapi.com* i *https://apis.guru/browse-apis*.



Rysunek 6.4. Specyfikacja interfejsu API Medici Bank zawierająca lokalizację punktu końcowego, lokalizację portalu oraz model uwierzytelniania

Shodan

Shodan to popularna wyszukiwarka urządzeń dołączonych do internetu. Regularnie skanuje całą przestrzeń adresową IPv4 w poszukiwaniu systemów z otwartymi portami, a uzyskane informacje udostępnia na stronie *https://shodan.io*. Za jej pomocą możesz odkrywać publiczne interfejsy API i pozyskiwać informacje o otwartych portach. Są to przydatne dane, jeśli znany jest tylko adres IP lub nazwa organizacji, którą trzeba zbadać. W wyszukiwarce Shodan, podobnie jak w Google, możesz po prostu wpisać nazwę domeny lub adres IP, jak również stosować parametry. Tabela 6.3 zawiera kilka przydatnych fraz.

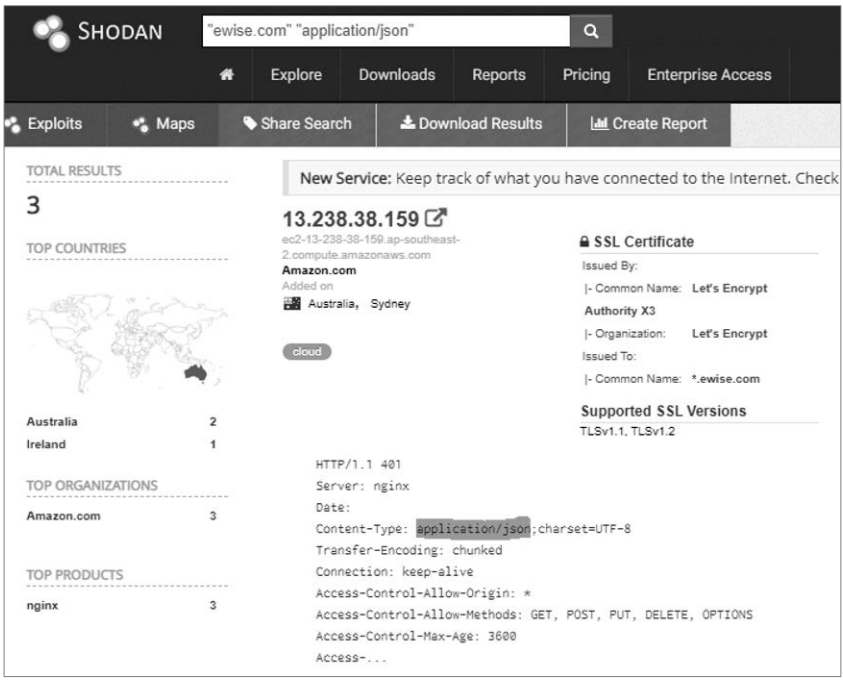
Za pomocą wyszukiwarki Shodan możesz również identyfikować punkty końcowe interfejsów API, w których nie są stosowane ogólnie przyjęte konwencje nazewnictwa. Jak pokazuje rysunek 6.5, aby przeszukać punkty firmy finansowej eWise (*https://www.ewise.com*), należy użyć następującej frazy:

```
"ewise.com" "content-type: application/json"
```

Jak pokazuje rysunek 6.5, wyszukiwarka Shodan znalazła potencjalnie interesujący punkt końcowy. Dokładniejsze badanie ujawnia informacje o certyfikacie SSL, serwerze WWW Nginx i odpowiedzi zawierającej nagłówek `application/json`. Serwer wysłał odpowiedź w powszechnie stosowanym w interfejsach REST API formacie JSON, zawierającą kod stanu `401`. W tym przykładzie udało się znaleźć punkt końcowy, w którym nie są stosowane powszechnie przyjęte konwencje nazewnictwa.

Tabela 6.3. Wybrane frazy wyszukiwarki Shodan

Fraza	Opis
`hostname:"targetname.com"`	Ogólne wyszukiwanie informacji o zadanej domenie. Aby uzyskać dokładniejsze dane, należy użyć parametru `hostname` z innymi, opisanymi niżej parametrami.
`"content-type: application/json"`	Interfejsy API powinny wykorzystywać format JSON lub XML. Ta fraza zwraca wyniki zawierające dane w formacie JSON.
`"content-type: application/xml"`	Fraza zwracająca wyniki zawierające dane w formacie XML.
`"200 OK"`	Fraza zwracająca pomyślnie przetworzone zapytania. Jednak interfejsy API, które nie akceptują formatu zapytań wyszukiwarki Shodan, zazwyczaj zwracają odpowiedzi z kodami stanu z grup 300 i 400.
`"wp-json"`	Fraza zwracająca aplikacje wykorzystujące interfejs WordPress API.



Rysunek 6.5. Przykładowe wyniki zwrócone przez wyszukiwarkę Shodan

Wyszukiwarka Shodan posiada również rozszerzenie umożliwiające wygodne sprawdzanie wyników za pomocą przeglądarki.

OWASP Amass

Opisane w rozdziale 4. narzędzie terminalowe OWASP Amass służy do tworzenia mapy publicznej sieci domeny na podstawie danych z ponad 55 źródeł. Może przeprowadzać skanowanie pasywne lub aktywne. W trybie aktywnym pozyskuje informacje bezpośrednio od badanego interfejsu, żądając jego certyfikatu. Natomiast w trybie pasywnym korzysta z wyszukiwarek internetowych (Google, Bing i HackerOne), baz certyfikatów (GoogleCT, Censys i FacebookCT), wyszukiwarek interfejsów API (Shodan, AlienVault, Cloudflare i GitHub) i archiwum stron Wayback Machine.

W rozdziale 4. znajdziesz informacje, jak zainstalować narzędzie Amass i dodać klucze API. Poniższe polecenie inicjuje pasywne skanowanie domeny *twitter.com*. Instrukcja `grep` powoduje wyświetlenie wyłącznie wyników dotyczących interfejsów API.

```
$ amass enum -passive -d twitter.com | grep api
legacy-api.twitter.com
api1-backup.twitter.com
api3-backup.twitter.com
tdapi.twitter.com
failover-urls.api.twitter.com
cdn.api.twitter.com
pulseone-api.smfc.twitter.com
urls.api.twitter.com
api2.twitter.com
apistatus.twitter.com
apiwiki.twtter.com
```

Wyniki skanowania zawierają 86 unikatowych poddomen API, w tym *legacy-api.twitter.com*. Zgodnie z informacjami zawartymi w rankingu OWASP API Security Top 10 nazwa *legacy* jest szczególnie interesująca, ponieważ sugeruje lukę w zarządzaniu zasobami.

Narzędzie Amass posiada kilka przydatnych argumentów. Jednym z nich jest `intel`, który służy do wyszukiwania certyfikatów SSL, odwrotnego przeszukiwania bazy WHOIS oraz numerów ASN (ang. *Autonomous System Number*, numer systemu autonomicznego) badanej domeny. W pierwszej kolejności należy użyć adresu IP w następujący sposób:

```
$ amass intel -addr <badany_adres_IP>
```

Jeżeli skanowanie zakończy się pomyślnie, wyniki będą zawierały nazwy domen, które można w następnym kroku wykorzystać z parametrem `whois`, aby przeprowadzić odwrotne przeszukiwanie bazy WHOIS:

```
$ amass intel -d <badana_domena> -whois
```

Powyższe polecenie może zwrócić mnóstwo wyników. Skoncentruj się na tych, które dotyczą badanej organizacji. Mając listę interesujących domen, użyj parametru `enum`, aby wyszukać poddomeny. Jeśli użyjesz parametru `-passive`, narzędzie nie będzie wchodziło w bezpośrednią interakcję z serwerem:

```
$ amass enum -passive -d <badana_domena>
```

Aktywne skanowanie przebiega podobnie jak pasywne, ale w tym trybie narzędzie dodatkowo rozpoznaje nazwy domen, podejmuje próbę transferu stref DNS i pobiera informacje o certyfikatach SSL:

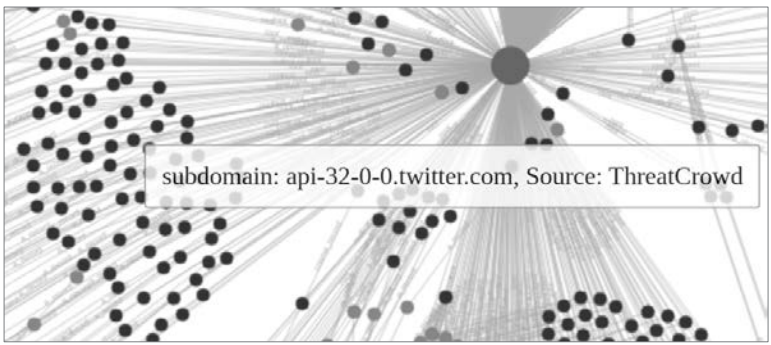
```
$ amass enum -active -d <badana_domena>
```

Jeszcze więcej informacji uzyskasz, gdy za pomocą argumentu `-brute` przeszukasz poddomeny metodą brutalnej siły. Przy użyciu argumentu `-w` wskaż plik z listą słów, a następnie zapisz wyniki w wybranym katalogu, używając argumentu `-dir`:

```
$ amass enum -active -brute -w <plik_z_listą_słów> -d <badana_domena> -dir
<nazwa_katalogu>
```

Za pomocą argumentu `viz` możesz w ciekawy sposób zwizualizować zależności między znalezionymi domenami i zapisać je w pliku w formacje HTML. Ilustruje to rysunek 6.6. Diagram możesz powiększać i sprawdzać powiązania między uzyskanymi wynikami. Być może wśród nich znajdziesz punkty końcowe.

```
$ amass viz -enum -d3 -dir <nazwa_katalogu>
```



Rysunek 6.6. Wizualizacja wyników badania domeny twitter.com, zapisana w pliku HTML

W ten sam sposób możesz wizualizować typy rekordów DNS, zależności między hostami i węzłami. Wszystkie węzły po lewej stronie rysunku 6.6 są poddomenami API. Duży punkt w środku reprezentuje domenę *twitter.com*.

Informacje eksponowane w serwisie GitHub

Niezależnie od tego, czy badana organizacja posiada własny dział programistów, czy nie, warto sprawdzić, czy ujawnia w serwisie GitHub (*https://github.com*) poufne informacje. Programiści wykorzystują ten serwis w zespołowej pracy nad projektem. Przeszukując go, można odkryć funkcjonalności interfejsu API, dokumentację, poufne informacje, klucze, hasła i tokeny, a następnie wykorzystać je w atakach.

Najpierw przeszukaj serwis GitHub, używając nazwy organizacji i słów oznaczających poufne informacje, np. *api-key*, *password* lub *token*. Przeglądaj zakładki repozytorium, ponieważ możesz w nich znaleźć punkty końcowe interfejsu API i jego słabe punkty. Przejrzyj kod źródłowy w zakładce *Code* (kod), przeczytaj w zakładce *Issues* (problemy) opisy błędów i zapoznaj się z proponowanymi zmianami w zakładce *Pull requests* (żądania zmian).

Zakładka Code

Zakładka *Code* zawiera aktualny kod źródłowy i różne pliki, m.in. *README* (patrz rysunek 6.7). Znajduje się tam nazwisko programisty, który zatwierdził plik, data tego zdarzenia, nazwiska współautorów i właściwy kod źródłowy.

Rysunek 6.7. Przykładowa zawartość zakładki Code w serwisie GitHub zawierająca pliki z kodem źródłowym

W zakładce *Code* możesz przejrzeć kod w zwykły sposób lub użyć klawiszy *Ctrl-F* i poszukać interesujących Cię terminów, np. *API*, *key* lub *password*. Oprócz tego przy użyciu przycisku *History* (historia) widocznego w prawym górnym rogu

możesz przeglądać wcześniej zatwierdzone wersje kodu. Jeżeli znajdziesz komentarz sugerujący, że kod był podatny na ataki, przejrzyj poprzednie wersje i sprawdź, czy luki wciąż istnieją.

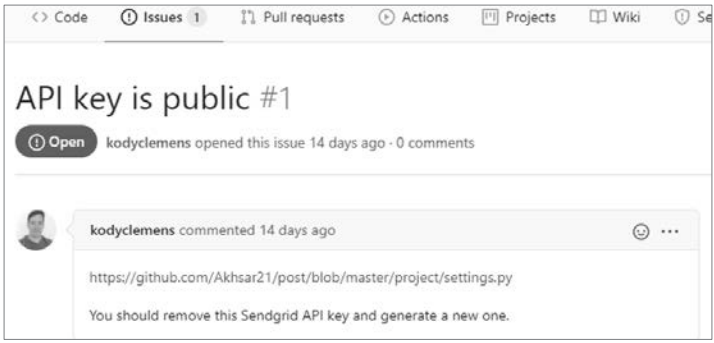
Przeglądając kod, możesz użyć przycisku *Split* (podziel), aby porównać wersje i szybko odszukać miejsca, w których zostały wprowadzone zmiany (patrz rysunek 6.8).

Rysunek 6.8. Widok podzielony za pomocą przycisku Split, umożliwiający porównanie poprzedniej (po lewej) i bieżącej (po prawej) wersji kodu

Jak widać na powyższym rysunku, z kodu został usunięty poufny klucz API do aplikacji SonarQube. Ponadto ujawniony jest punkt końcowy API, w którym klucz był używany.

Zakładka Issues

W zakładce *Issues* programiści notują problemy, zadania i żądania funkcjonalności. Otwarty problem z dużym prawdopodobieństwem oznacza, że kod jest wciąż narażony na ataki (patrz rysunek 6.9).

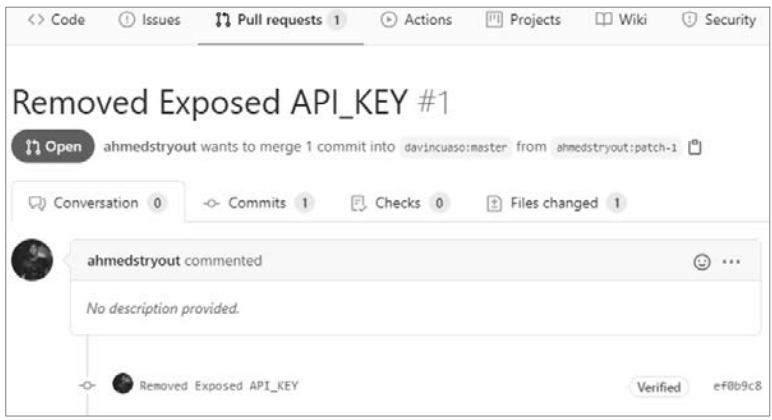


Rysunek 6.9. Opis problemu w serwisie GitHub, ujawniający dokładną lokalizację klucza API w kodzie aplikacji

Jeśli problem został rozwiązany, zanotuj datę zatwierdzenia kodu, a następnie przeszukaj historię wersji pod kątem dokonanych w tym czasie zmian.

Zakładka Pull requests

Zakładka *Pull requests* to miejsce, w którym programiści współpracują ze sobą nad zmianami w kodzie. Jeżeli będziesz miał szczęście, to przeglądając proponowane zmiany, znajdziesz w interfejsie API błąd, który jest właśnie naprawiany. Rysunek 6.10 przedstawia przykładowe zrealizowane żądanie usunięcia z kodu klucza API.



Rysunek 6.10. Komentarz programisty do żądania zmiany, ujawniający klucz API

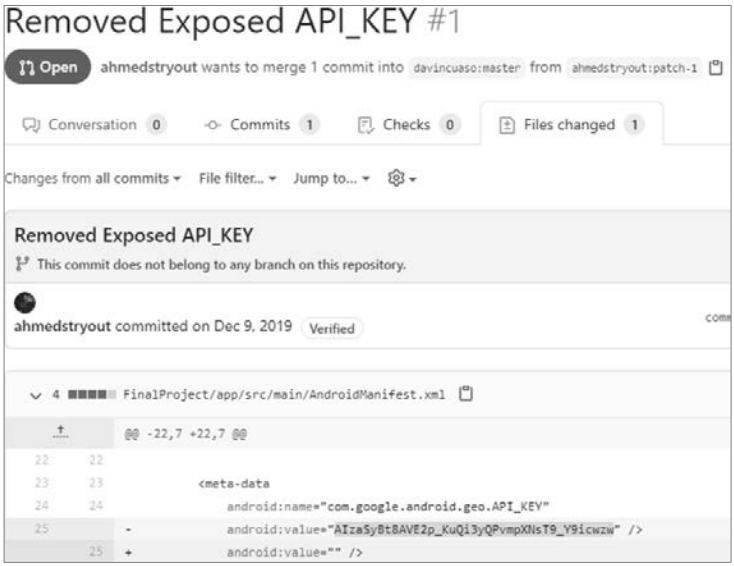
Ponieważ żądanie nie zostało jeszcze uwzględnione w ostatecznej wersji kodu, w zakładce *Files changed* (zmienione pliki) klucz jest wciąż widoczny (patrz rysunek 6.11).

Zakładka *Files changed* ujawnia fragment kodu, w którym programista zamierza wprowadzić zmiany. Jak widać, klucz API znajduje się w wierszu nr 25. Pod nim jest umieszczony proponowany nowy wiersz, bez klucza.

Jeżeli w serwisie GitHub nie znajdziesz słabych punktów, spróbuj wykorzystać go do opracowania profilu badanego interfejsu. Zanotuj wykorzystywane języki programowania, punkty końcowe, dokumentację itp. Wszelkie tego rodzaju informacje mogą Ci się przydać w przyszłości.

Rekonesans aktywny

Rekonesans pasywny ma ten mankament, że gromadzone w jego trakcie informacje pochodzą z drugiej ręki. Jako tester musisz je zweryfikować. Najlepszym sposobem jest porównanie ich z informacjami uzyskanymi bezpośrednio poprzez skanowanie portów i słabych punktów, użycie polecenia `ping`, wysyłanie zapytań HTTP i wykonanie innych operacji na badanym interfejsie API.



Rysunek 6.11. Zakładka Files changed zawierająca zmiany wprowadzone w kodzie

W tym podrozdziale skupimy się na wykrywaniu interfejsu API organizacji poprzez ogólne skanowanie środowiska, ręczną analizę interfejsu i skanowanie ukierunkowane. Techniki te wykorzystasz w ćwiczeniu na końcu rozdziału.

Proces rekonesansu aktywnego

Celem opisanego w tym podrozdziale procesu aktywnego rekonesansu jest skuteczne i wszechstronne zbadanie interfejsu oraz wykrycie wszystkich słabych punktów, które można wykorzystać do uzyskania dostępu do systemu. Każda kolejna faza procesu bazuje na informacjach uzyskanych w poprzedniej fazie i zawęża obszar poszukiwań. W fazie pierwszej, podczas zautomatyzowanego skanowania wykrywającego, identyfikuje się aktywne usługi wykorzystujące protokoły HTTP i HTTPS. Faza druga polega na ręcznej analizie tych usług z perspektywy użytkownika i hakera, a jej celem jest znalezienie ciekawych punktów interfejsu. Faza trzecia bazuje na informacjach pozyskanych w fazie drugiej. Obszar skanowania jest zawężany, aby można było dokładnie zbadać wykryte porty i usługi. Jest to wydajny proces, ponieważ w czasie, gdy w tle jest wykonywane zautomatyzowane skanowanie, można się skupić na celu badań. Gdy w trakcie analizy trafisz w ślepy zaułek, wróć do wyników skanowania i poszukaj innej drogi.

Proces nie jest liniowy. Po każdym coraz bardziej ukierunkowanym skanowaniu analizuj wyniki i wykorzystuj je w kolejnym skanowaniu. W każdej chwili możesz znaleźć lukę w zabezpieczeniach i spróbować ją wykorzystać. Jeśli Ci się to uda, możesz przejść do fazy poeksploracyjnej. W przeciwnym wypadku wróć do skanowania i analiz.

Faza zerowa: skanowanie oportunistyczne

Jeśli odkryjesz lukę w zabezpieczeniach, od razu skorzystaj z okazji i spróbuj ją wykorzystać. Na komentarz pozostawiony na nieukończonej stronie internetowej możesz natknąć się zarówno po kilku sekundach skanowania, jak i po miesiącach badań. W takiej sytuacji od razu przejdź do eksploracji słabego punktu. W razie potrzeby zawsze możesz wrócić do bieżącej fazy procesu rekonesansu. Gdy nabierzesz doświadczenia, będziesz wiedział, czy warto gubić się analizach, które mogą prowadzić w ślepy zaułek, czy też przeznaczyć wszystkie siły na eksplorację.

Faza pierwsza: skanowanie wykrywające

Celem skanowania wykrywającego jest zidentyfikowanie potencjalnych punktów wyjścia do badań. Zacznij od ogólnego skanowania, wykrywania hostów, portów, usług i systemów operacyjnych zgodnie z opisem w podrozdziale „Ogólne skanowanie za pomocą Nmap". Interfejsy API bazują na protokołach HTTP i HTTPS, więc gdy tylko je wykryjesz, przejdź do fazy drugiej.

Faza druga: ręczna analiza

Ręczna analiza polega na eksplorowaniu aplikacji internetowej za pomocą przeglądarki i klienta interfejsu API. Celem jest poznanie i przetestowanie wszystkich możliwości interakcji. W tej fazie bada się stronę internetową, przechwytuje zapytania, szuka odnośników do interfejsu API i dokumentacji oraz analizuje algorytm biznesowy. Aplikację należy rozpatrywać z trzech perspektyw: gościa, uprawnionego użytkownika i administratora. **Gościem** jest anonimowy użytkownik, który odwiedza stronę po raz pierwszy. Stronę, która zawiera publicznie dostępne informacje i nie wymaga uwierzytelnienia użytkownika, odwiedzają tylko goście. **Uprawniony użytkownik** to ten, który przeszedł proces rejestracji i otrzymał określony poziom dostępu. Natomiast **administrator** posiada uprawnienia do zarządzania i utrzymywania interfejsu API.

Pierwszym krokiem jest otwarcie strony w przeglądarce, zbadanie jej i przeanalizowanie z wyżej wymienionych perspektyw. Oto kilka uwag dotyczących każdej z nich:

Gość

Jak nowy użytkownik korzysta ze strony? Czy może wchodzić w interakcje z interfejsem API? Czy dokumentacja interfejsu jest publicznie dostępna? Jakie operacje może wykonywać użytkownik?

Uprawniony użytkownik

Jakie operacje, niedostępne dla gościa, może wykonywać uprawnionych użytkownik? Czy może ładować pliki? Czy ma dostęp do nowych sekcji aplikacji? Czy może korzystać z interfejsu API? Jak aplikacja rozpoznaje uprawionego użytkownika?

Administrator

Gdzie administrator loguje się, aby móc zarządzać aplikacją? Co znajduje się w kodzie źródłowym strony? Jakie komentarze pozostawili programiści? Jakie języki programowania zostały użyte? Jakie sekcje są w fazie rozwoju lub eksperymentów?

W następnym kroku przeanalizuj aplikację z perspektywy hakera. Przechwyć zapytania HTTP za pomocą programu Burp Suite. Gdy użytkownik korzysta z paska wyszukiwania lub uwierzytelnia się, aplikacja może w celu wykonania żądanej operacji wysyłać zapytania do interfejsu API, które zobaczysz w programie Burp Suite.

Jeżeli napotkasz przeszkody, przejrzyj uzyskane w fazie pierwszej wyniki skanowania w tle i przejdź do fazy trzeciej: skanowania ukierunkowanego.

Faza trzecia: skanowanie ukierunkowane

W tej fazie stosuje się udoskonalone metody skanowania i narzędzia właściwe dla badanego interfejsu. Skanowanie ukierunkowane, w odróżnieniu od wykrywającego, powinno koncentrować się na określonym typie interfejsu API, jego wersji, typie aplikacji internetowej, wykrytych wersjach usług (niezależnie od tego, czy aplikacja wykorzystuje protokół HTTP, czy HTTPS), aktywnych portach TCP i wnioskach z analizy algorytmu biznesowego. Jeśli na przykład odkryjesz, że interfejs wykorzystuje niestandardowy port TCP, ustaw skaner tak, aby dokładniej zbadał ten port. Jeśli dowiesz się, że aplikacja opiera się na platformie WordPress, otwórz adres */wp-json/wp/v2*, aby sprawdzić, czy jest dostępny jej interfejs API. W tej fazie powinieneś znać adresy URL aplikacji i możesz metodą brutalnej siły poszukać typowych identyfikatorów zasobów, ukrytych katalogów i plików (patrz podrozdział „Wyszukiwanie identyfikatorów URI metodą brutalnej siły za pomocą programu Gobuster"). Podczas używania narzędzi przeglądaj na bieżąco pojawiające się wyniki, aby prowadzić bardziej ukierunkowaną ręczną analizę.

W kolejnych podrozdziałach są opisane narzędzia i techniki, które będziesz stosował na wszystkich etapach rekonesansu aktywnego: skanowanie wykrywające za pomocą Nmap, ręczna analiza za pomocą Chrome DevTools oraz skanowanie ukierunkowane za pomocą Burp Suite i OWASP ZAP.

Ogólne skanowanie za pomocą Nmap

Nmap to potężne narzędzie do skanowania portów, wyszukiwania luk w zabezpieczeniach, identyfikowania usług i hostów. Jest moim ulubionym programem do przeprowadzania skanowania wykrywającego w fazie pierwszej, ale używam go również do skanowania ukierunkowanego. Potędze programu Nmap jest poświęconych wiele książek i stron internetowych, więc nie będziemy się tu zagłębiać w ten temat.

Aby wykryć interfejsy API, należy użyć narzędzia Nmap w dwóch trybach: ogólnego wykrywania i skanowania wszystkich portów. W pierwszym trybie narzędzie wykorzystuje domyślne skrypty i identyfikuje usługi. Wyniki zapisuje w trzech formatach na potrzeby późniejszej analizy. Aby zapisać wyniki w formacie XML,

należy użyć argumentu `-oX`, w wewnętrznym formacie Nmap — argumentu `-oN`, w formacie umożliwiającym przeszukiwanie za pomocą polecenia `grep` — argumentu `-oG`, a we wszystkich trzech formatach — argumentu `-oA`. Składnia polecenia jest następująca:

```
$ nmap -sC -sV <adres_IP_lub_zakres_adresów> -oA <nazwa_pliku>
```

Aby szybko przeskanować wszystkie 65 535 portów TCP w poszukiwaniu uruchomionych usług, ich wersji i systemów operacyjnych, użyj następującego polecenia:

```
$ nmap -p- <adres_IP> -oA <nazwa_pliku>
```

Gdy tylko ogólne polecenie skanujące zacznie zwracać wyniki, rozpocznij skanowanie wszystkich portów. Następnie przeprowadź ręczną analizę wyników. Przeglądając wyniki dotyczące komunikacji HTTP i serwerów WWW, prawdopodobnie odkryjesz interfejsy API. Zazwyczaj interfejsy wykorzystują porty o numerach 80 i 443, ale ogólnie mogą to być dowolne porty. Jeżeli znajdziesz serwer WWW, otwórz przeglądarkę i rozpocznij analizę.

Wyszukiwanie ukrytych ścieżek w pliku robots.txt

Robots.txt to plik tekstowy zawierający informacje dla robotów indeksujących, jakie strony mają pomijać. Jak na ironię, są to również podpowiedzi, jakie ścieżki mają pozostać tajemnicą. Plik ten znajduje się w głównym katalogu serwera WWW (np. *https://www.twitter.com/robots.txt*).

Poniżej znajduje się plik *robots.txt* z rzeczywistego, aktywnego serwera WWW, zawierający ścieżkę */api/*:

```
User-agent: *
Disallow: /appliance/
Disallow: /login/
Disallow: /api/
Disallow: /files/
```

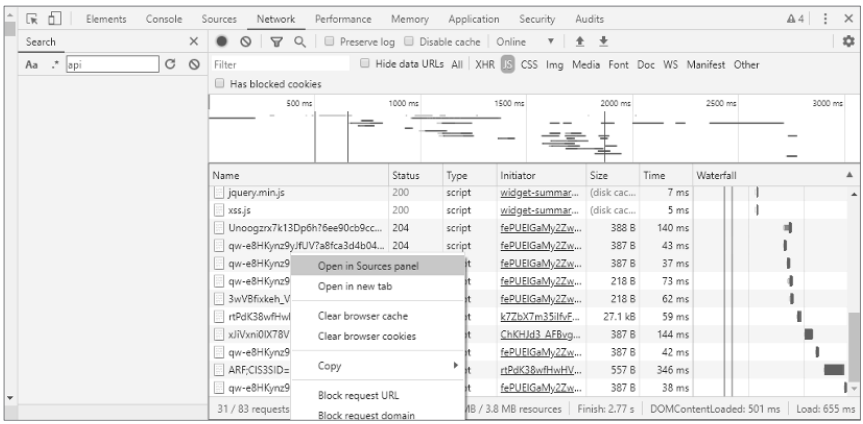
Wyszukiwanie poufnych informacji za pomocą Chrome DevTools

W rozdziale 4. wspomniałem, że przeglądarka Chrome zawiera kilka bardzo niedocenianych narzędzi do badania aplikacji internetowych. Stosując opisane niżej techniki, będziesz mógł łatwo i systematycznie filtrować tysiące linii kodu i wyszukiwać poufne informacje w kodach źródłowych stron internetowych.

Otwórz w przeglądarce stronę, którą zamierzasz zbadać, a następnie naciśnij klawisz *F12* lub *Ctrl+Shift+I*, aby otworzyć narzędzia Chrome DevTools. Dopasuj

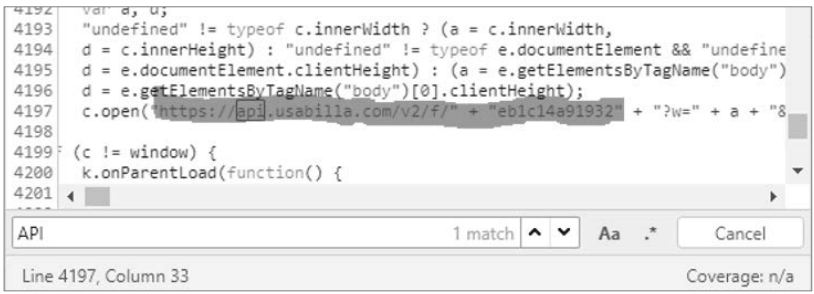
wielkość panelu tak, aby mieć odpowiednią ilość miejsca do pracy. Następnie kliknij zakładkę *Network* i odśwież stronę.

Teraz poszukaj plików JavaScript o ciekawych nazwach (może zajdziesz nawet *API*). Aby zobaczyć kod źródłowy w wybranym pliku, kliknij go prawym przyciskiem myszy i wybierz polecenie *Open in Sources panel* (otwórz w panelu Sources), jak na rysunku 6.12. Możesz również kliknąć plik XHR i przejrzeć zapytanie AJAX.



Rysunek 6.12. Polecenie Open in Sources panel w zakładce Network

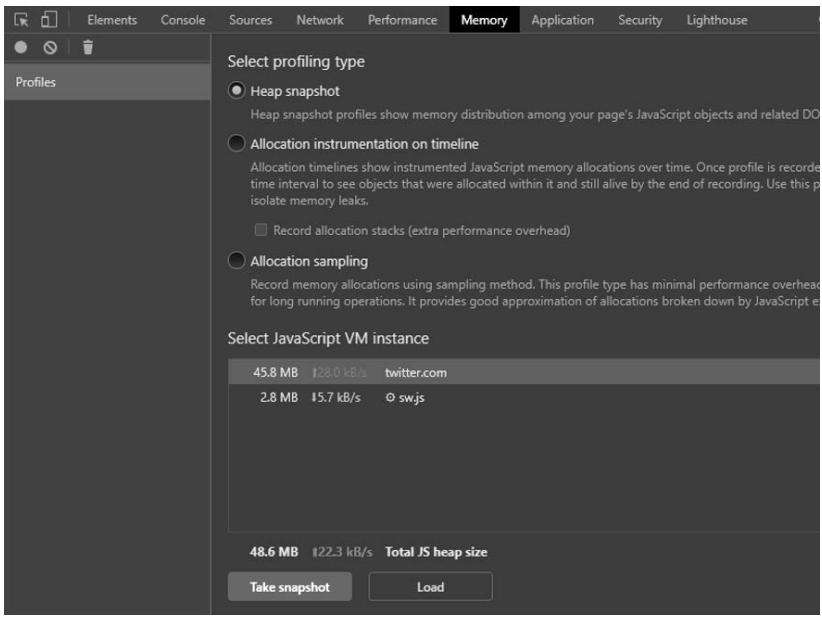
Poszukaj potencjalnie interesujących wierszy kodu JavaScript, zawierających np. ciągi `API`, `APIkey`, `secret`, `password`. Rysunek 6.13 przedstawia przykładowy skrypt, w którym interfejs API jest ukryty na głębokości prawie 4200 wierszy.



Rysunek 6.13. Wiersz nr 4197 zawierający odwołanie do interfejsu API

W zakładce *Memory* możesz zrobić migawkę sterty pamięci. Niektóre pliki JavaScript zawierają wszelkiego rodzaju informacje, składają się z tysięcy linii kodu i nie do końca wiadomo, w jaki sposób aplikacja korzysta z interfejsu API. W takich przypadkach można w tej zakładce sprawdzić, jak aplikacja internetowa wykorzystuje zasoby podczas interakcji z interfejsem API.

Kliknij zakładkę *Memory* i zaznacz opcję *Heap snapshot* (migawka sterty pamięci). Następnie w panelu *Select JavaScript VM instance* (instancja maszyny wirtualnej JavaScript) zaznacz instancję i kliknij przycisk *Take snapshot* (zrób migawkę), jak na rysunku 6.14.

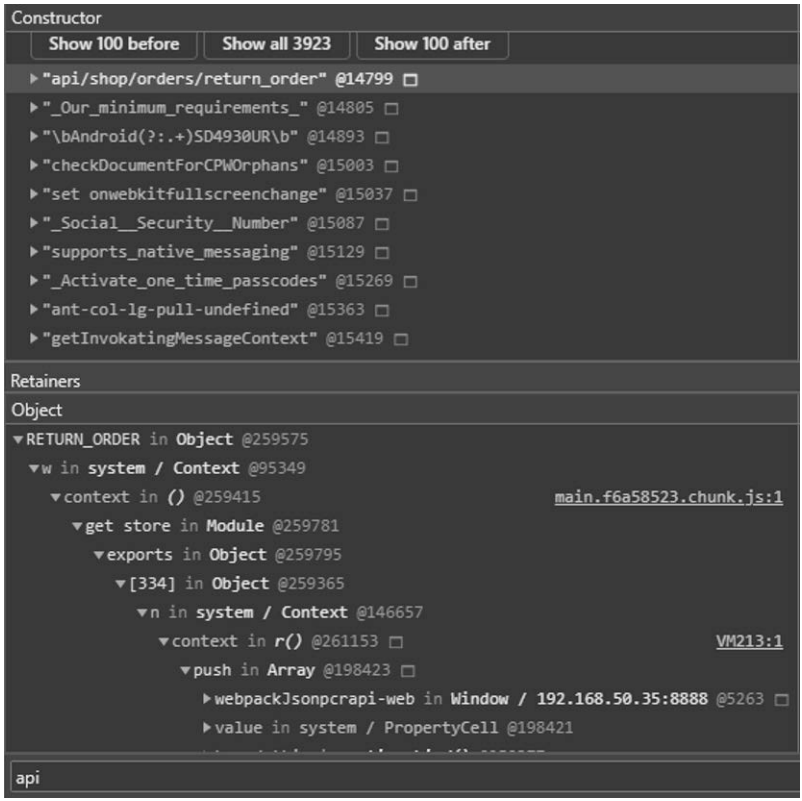


Rysunek 6.14. Zakładka Memory

Gdy w panelu po lewej stronie zakończy się analiza migawki, zaznacz ją i naciśnij klawisze *Ctrl+F*, aby wyszukać ścieżki interfejsu API. Poszukaj typowych ciągów `api`, `v1`, `v2`, `swagger`, `rest` i `dev`. Jeśli potrzebujesz inspiracji, przejrzyj listę słów Assetnote API (*http://wordlists.assetnote.io*). Jeśli zbudowałeś system hakerski zgodnie opisem w rozdziale 4., listy słów znajdziesz również w katalogu */api/wordlists*. Rysunek 6.15 przedstawia wynik wyszukiwania w migawce ciągu `api`.

Jak widać, zakładka *Memory* może pomóc w wykryciu interfejsu API i jego ścieżek. Możesz zrobić wiele migawek, a następnie je porównywać. W ten sposób możesz wyszukiwać ścieżki w różnych częściach i funkcjonalnościach aplikacji, zarówno przed uwierzytelnieniem użytkownika, jak i po nim.

Na koniec kliknij zakładkę *Performance* (wydajność). Za jej pomocą możesz rejestrować różne operacje wykonywane przez użytkownika (na przykład kliknięcia przycisków) i przedstawiać je na osi czasu z milisekundową dokładnością. W ten sposób możesz sprawdzać, czy jakieś zainicjowane na stronie zdarzenia powodują wysyłanie w tle zapytań do interfejsu API. Wystarczy, że klikniesz w tym celu przycisk rejestrowania, wykonasz jakieś operacje na stronie i zatrzymasz rejestrowanie. Pojawią się zgłoszone zdarzenia i zainicjowane przez nie działania, które będziesz mógł zbadać. Rysunek 6.16 przedstawia zarejestrowane zdarzenia wywołane kliknięciem przycisku.



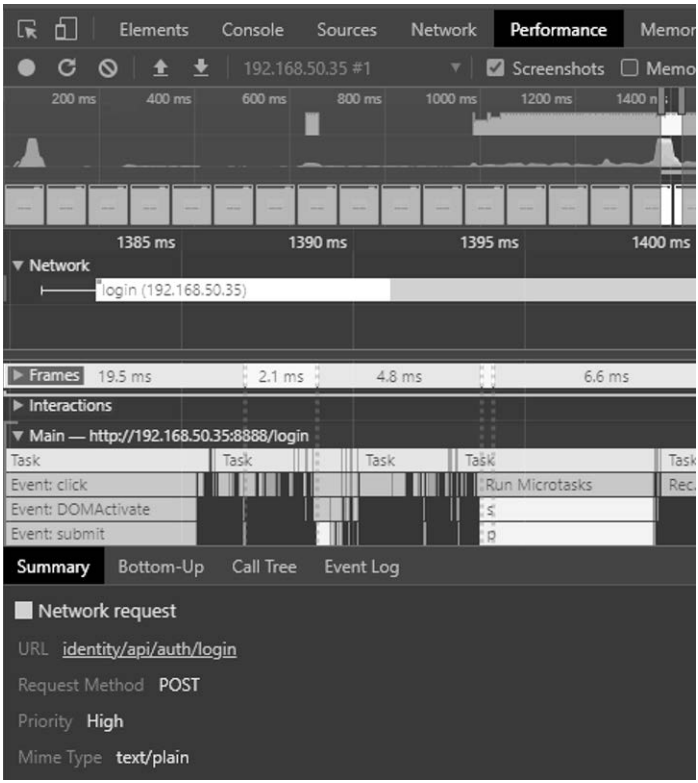
Rysunek 6.15. Wyniki przeszukiwania migawki sterty pamięci

W listingu *Main* (główna strona) widać zdarzenie, które spowodowało wysłanie zapytania `POST` ze ścieżką */identity/api/auth/login*. To jednoznaczny sygnał, że istnieje interfejs API. Aby dokładniej poznać aktywność przeglądarki w skali czasu, przejrzyj wzniesienia i spadki wykresu znajdującego się w górnej części okna. Wzniesienie oznacza zdarzenie, na przykład kliknięcie na stronie. Przejrzyj zdarzenia, klikając wykres w różnych miejscach.

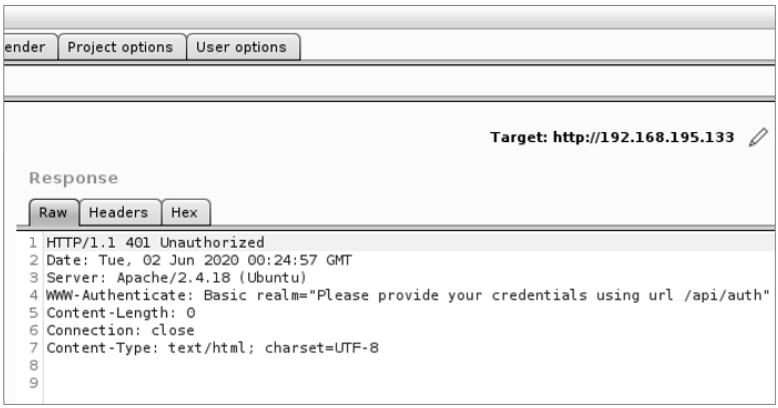
Jak widać, narzędzia DevTools posiadają wiele potężnych funkcjonalności, które mogą pomóc w wykrywaniu interfejsów API. Nie lekceważ ich użyteczności.

Weryfikacja interfejsu API za pomocą Burp Suite

Za pomocą Burp Suite możesz nie tylko odkrywać interfejsy API, ale również weryfikować zebrane informacje. Przechwytuj wysyłane przez przeglądarkę zapytania HTTP, a następnie, klikając przycisk *Forward*, przesyłaj je do serwera. Możesz je również wysyłać do modułu *Repeater* i przeglądać odpowiedzi serwera, jak na rysunku 6.17.



Rysunek 6.16. Zapis zdarzeń w zakładce Performance



Rysunek 6.17. Odpowiedź serwera zawierająca kod stanu 401 Unauthorized

Jak widać, serwer zwrócił odpowiedź z kodem stanu `401 Unauthorized`, oznaczającą, że bieżący użytkownik nie jest uprawniony do korzystania z interfejsu API. Porównaj ją z odpowiedzią na zapytanie o nieistniejący zasób. W ten sposób dowiesz

się, jak serwer reaguje na tego typu zapytania. Aby wysłać zapytanie o nieistniejący zasób, po prostu dopisz w zakładce *Repeater* losowe znaki do ścieżki URL, np. `GET /user/testO98765`. Następnie wyślij zapytanie i sprawdź, jak odpowie serwer WWW. Zazwyczaj jest to kod stanu `404` lub podobny. W tym przykładzie szczegółowy komunikat o błędzie, zawarty w nagłówku `WWW-Authenticate`, ujawnia ścieżkę */api/auth* potwierdzającą istnienie interfejsu API. Aby przejść przyspieszony kurs korzystania z programu Burp Suite, wróć do rozdziału 4.

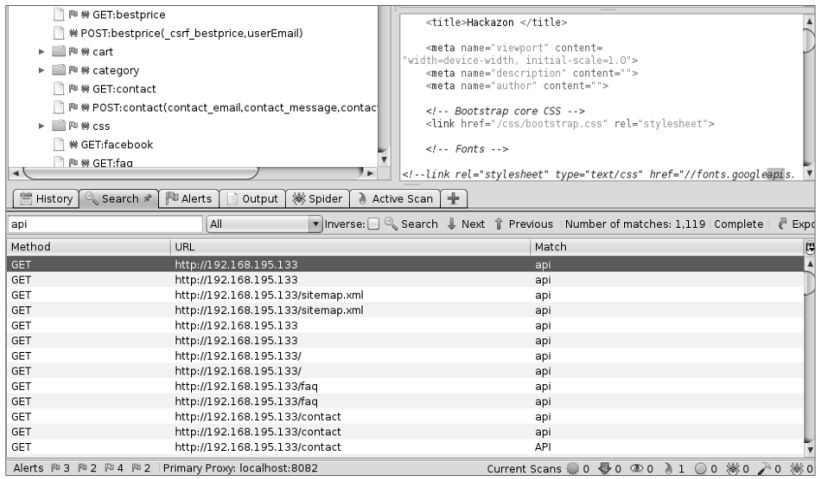
Skanowanie identyfikatorów URI za pomocą OWASP ZAP

Jednym z celów rekonesansu aktywnego jest wykrycie wszystkich katalogów i plików strony internetowej, czyli identyfikatorów **URI** (ang. *Uniform Resource Identifier*, **jednolity identyfikator zasobów**). Istnieją dwie metody wykrywania identyfikatorów: skanowanie i brutalna siła. Program OWASP ZAP indeksuje strony internetowe w poszukiwaniu treści i skanuje je pod kątem odnośników do innych stron.

Otwórz program ZAP. Zamknij okno z opcjami zapisywania sesji i kliknij zakładkę *Quick Start* (szybki start), jak na rysunku 6.18. Następnie kliknij przycisk *Automated Scan* (skanowanie automatyczne), wpisz adres URL strony, którą zamierzasz zbadać, i kliknij przycisk *Attack* (atakuj).

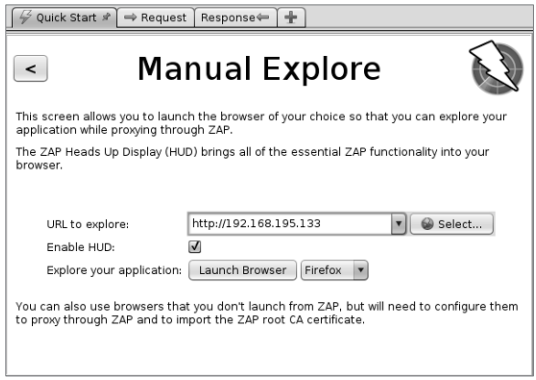
Rysunek 6.18. Automatyczne skanowanie strony za pomocą programu OWASP ZAP

W zakładkach *Spider* (pająk) i *Sites* (strony) obserwuj na bieżąco wyniki pojawiające się w trakcie skanowania. Wśród nich możesz odkryć interfejsy API. Jeśli nie znajdziesz żadnych oczywistych wpisów, kliknij zakładkę *Search* (szukaj) i poszukaj typowych dla punktów końcowych fraz *API*, *GraphQL*, *JSON*, *RPC* i *XML*.



Rysunek 6.19. Wyszukiwanie interfejsów API w wynikach automatycznego skanowania

Jeżeli zechcesz dokładniej zbadać jakąś sekcję witryny, przeprowadź ręczną eksplorację za pomocą komponentu ZAP HUD, umożliwiającego interakcję z przyciskami aplikacji i wprowadzanie danych w polach formularza. Program ZAP wykona wtedy dodatkowe skanowanie w poszukiwaniu luk w zabezpieczeniach. Kliknij ponownie zakładkę *Quick Start*, następnie przycisk ze strzałką, aby opuścić automatyczne skanowanie, i przycisk *Manual Explore* (ręczna eksploracja). Wpisz adres badanej strony, wybierz z listy przeglądarkę, jak na rysunku 6.20, i kliknij przycisk *Launch Browser* (uruchom przeglądarkę).



Rysunek 6.20. Ręczna eksploracja strony w sekcji Manual Explore

Na stronie powitalnej, pokazanej na rysunku 6.21, kliknij przycisk *Continue to your target* (przejdź do swojego celu).

Rysunek 6.21. Ekran powitalny komponentu ZAP HUD

Teraz możesz ręcznie eksplorować aplikację internetową. Program ZAP będzie ją skanował w tle w poszukiwaniu luk w zabezpieczeniach i szukał dodatkowych ścieżek. Przy lewej i prawej krawędzi okna będzie widocznych szereg przycisków. Kolorowe flagi oznaczają alerty o lukach i ciekawych anomaliach i będą się zmieniać, gdy będziesz przeglądać witrynę.

Wyszukiwanie identyfikatorów URI metodą brutalnej siły za pomocą programu Gobuster

Gobuster to terminalowe narzędzie służące do wyszukiwania identyfikatorów URI i poddomen metodą brutalnej siły. (Jeżeli preferujesz interfejs graficzny, użyj programu OWASP Dirbuster). Program odczytuje z zadanej listy popularne nazwy katalogów i poddomen, wysyła zapytania do serwera i prezentuje istotne odpowiedzi. Wyniki zawierają ścieżki URL i kody stanu HTTP. Identyfikatory URI możesz również wyszukiwać za pomocą modułu Burp Suite Intruder, ale jest on znacznie wolniejszy niż Gobuster.

Przed każdym użyciem narzędzia wykorzystującego metodę brutalnej siły pamiętaj o kompromisie między długością listy słów a czasem potrzebnym na zebranie wyników. W systemie Kali w katalogu */usr/share/wordlists/dirbuster* znajdują się obszerne listy słów, ale ich przetworzenie może zająć dużo czasu. Zamiast nich możesz użyć listy, którą zapisałeś w rozdziale 4. w katalogu *~/api/wordlists*. Wyszukiwanie będzie znacznie szybsze, ponieważ lista zawiera tylko słowa charakterystyczne dla interfejsów API, więc jest dość krótka.

Poniższe polecenie wyszukuje słowa z przykładowej listy w interfejsie o zadanym adresie IP:

```
$ gobuster dir -u http://192.168.195.132:8000 -w
/home/hapihacker/api/wordlists/common_apis_160
===============================================================
Gobuster
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.195.132:8000
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /home/hapihacker/api/wordlists/common_apis_160
[+] Negative Status codes:   404
[+] User Agent:              gobuster
[+] Timeout:                 10s
===============================================================
09:40:11 Starting gobuster in directory enumeration mode
===============================================================
/api                 (Status: 200) [Size: 253]
/admin               (Status: 500) [Size: 1179]
/admins              (Status: 500) [Size: 1179]
/login               (Status: 200) [Size: 2833]
/register            (Status: 200) [Size: 2846]
```

Gdy znajdziesz w powyższych wynikach katalog taki jak na przykład */api*, możesz go dokładniej zbadać za pomocą programu Burp Suite.

Program Gobuster posiada dodatkowe argumenty, które możesz wyświetlić, używając argumentu `-h`:

```
$ gobuster dir -h
```

Jeśli chcesz wykluczyć z wyników niektóre kody stanu, użyj argumentu `-b`. Natomiast w celu uwzględnienia dodatkowych stanów zastosuj argument `-x`. Przy użyciu poniższego polecenia możesz przeprowadzić dokładniejsze poszukiwania:

```
$ gobuster dir -u http://192.168.195.132:8000 -w /usr/share/wordlists/
api_list/common_apis_160 -x 200,202,301 -b 302
```

Za pomocą narzędzia Gobuster można szybko wyszukiwać aktywne adresy URL i ścieżki interfejsów API.

Wykrywanie zasobów interfejsów API za pomocą narzędzia Kiterunner

W rozdziale 4. opisałem niezwykły produkt firmy Assetnote: program Kiterunner. Jest to najlepsze narzędzie do wykrywania punktów końcowych i zasobów interfejsów API. Nadszedł czas, aby zrobić z niego użytek.

Program Gobuster sprawdza się w szybkim skanowaniu aplikacji internetowych i wyszukiwaniu adresów URL. W tym celu wykorzystuje standardowe zapytania HTTP GET. Natomiast Kiterunner nie tylko stosuje wszystkie metody HTTP właściwe dla interfejsów API (`GET`, `POST`, `PUT` i `DELETE`), ale też testuje typowe ścieżki. Na przykład wysyła nie tylko zapytanie `GET /api/v1/user/create`, ale również bardziej odpowiednie `POST /api/v1/user/create`.

Przykładowe skanowanie interfejsu o zadanym adresie IP wygląda następująco:

```
$ kr scan http://192.168.195.132:8090 -w ~/api/wordlists/data/kiterunner/
routes-large.kite
+---------------------+-------------------------------------------------------------
------------------+-------------------------------------------------------------
| SETTING             | VALUE                                                       |
+---------------------+-------------------------------------------------------------
------------------+-------------------------------------------------------------
| delay               | 0s                                                          |
| full-scan           | false                                                       |
| full-scan-requests  | 1451872                                                     |
| headers             | [x-forwarded-for:127.0.0.1]                                 |
| kitebuilder-apis    | [/home/hapihacker/api/wordlists/data/kiterunner/routes-
large.kite]                                                                         |
| max-conn-per-host   | 3                                                           |
| max-parallel-host   | 50                                                          |
| max-redirects       | 3                                                           |
| max-timeout         | 3s                                                          |
| preflight-routes    | 11                                                          |
| quarantine-threshold | 10                                                         |
| quick-scan-requests | 103427                                                      |
| read-body           | false                                                       |
| read-headers        | false                                                       |
| scan-depth          | 1                                                           |
| skip-preflight      | false                                                       |
| target              | http://192.168.195.132:8090                                 |
| total-routes        | 957191                                                      |
| user-agent          | Chrome. Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.96 Safari/537.36            |
+---------------------+-------------------------------------------------------------
POST    400 [    941,   46,  11] http://192.168.195.132:8090/
trade/queryTransationRecords 0cf689f783e6dab12b6940616f005ecfcb3074c4
POST    400 [    941,   46,  11] http://192.168.195.132:8090/event
0cf6890acb41b42f316e86efad29ad69f54408e6
GET     301 [    243,    7,  10] http://192.168.195.132:8090/api-docs -> /
api-docs/?group=63578528&route=33616912 0cf681b5cf6c877f2e620a8668a4abc7ad07e2db
```

Jak widzisz, Kiterunner tworzy listę ciekawych ścieżek. Unikatowa odpowiedź na zapytanie zawierające ścieżkę */api/* wskazuje, że istnieje interfejs API.

Zwróć uwagę, że podczas skanowania nie zostały użyte nagłówki autoryzacyjne, których prawdopodobnie wymaga badany interfejs. W rozdziale 7. dowiesz się, jak używać narzędzia Kiterunner z takimi nagłówkami.

Jeśli zamiast pliku *.kite* chciałbyś użyć tekstowej listy słów, wskaż ją za pomocą argumentu `brute`:

```
$ kr brute <badana_domena> -w <plik_tekstowy>
```

Jeżeli chcesz zbadać kilka domen, zapisz je w osobnych wierszach w pliku tekstowym. Możesz stosować dowolne formaty adresów URL, jak niżej:

Test.com

Test2.com:443

http://test3.com

http://test4.com

http://test5.com:8888/api

Jedną z najciekawszych funkcjonalności narzędzia Kiterunner jest możliwość powtarzania zapytań. Dzięki niej można nie tylko uzyskać ciekawe wyniki, ale też dokładnie przeanalizować wybrane zapytanie. Aby ponownie wysłać zapytanie, użyj argumentów `kb replay`, wklej treść zapytania i wskaż plik z listą słów, na przykład:

```
$ kr kb replay "GET     414 [    183,    7,   8] http://192.168.50.35:8888/
api/privatisations/count 0cf6841b1e7ac8badc6e237ab300a90ca873d571" -w ~/api/
wordlists/data/kiterunner/routes-large.kite
```

Powyższe polecenie ponownie wysyła zapytanie i odbiera odpowiedź, którą możesz przejrzeć, aby sprawdzić, czy zawiera coś wartego zbadania. Zwykle wyszukuję ciekawe wyniki, które następnie testuję za pomocą programów Postman i Burp Suite.

Podsumowanie

W tym rozdziale dowiedziałeś się, jak wygląda w praktyce wykrywanie interfejsów API w drodze pasywnego i aktywnego rekonesansu. Zbieranie informacji jest prawdopodobnie najważniejszym etapem testowania interfejsów. Istnieje ku temu kilka powodów. Po pierwsze nie sposób zbadać interfejsu API, jeśli nie można go znaleźć. Rekonesans pasywny daje obraz organizacji i obszaru ataku. Pozwala łatwo pozyskać ważne dane, takie jak hasła, klucze, tokeny i luki w zabezpieczeniach powodujące wyciek informacji. Po drugie aktywna interakcja ze środowiskiem klienta pozwala określić operacyjny kontekst jego interfejsu API, na który

składa się system operacyjny serwera, wersja i typ interfejsu, wersje oprogramowania pomocniczego, podatność interfejsu na znane eksploity, przeznaczenie systemów i ich wzajemne zależności.

W następnym rozdziale zaczniesz manipulować interfejsami i je zakłócać, aby wykryć ich słabe punkty.

Ćwiczenie 3. Rekonesans aktywny w teście czarnej skrzynki

Znana firma Car Services świadcząca usługi samochodowe zwróciła się do Ciebie z zapytaniem o wykonanie testów penetracyjnych jej interfejsu API. Może Ci udostępnić pewne informacje, takie jak adres IP, numer portu, i być może dokumentację interfejsu, ale test ma być wykonany w trybie czarnej skrzynki. Firma liczy więc na to, że znajdziesz jej interfejs i sprawdzisz, czy ma jakieś luki w zabezpieczeniach.

Zanim zaczniesz pracę, sprawdź, czy jest dostępna aplikacja crAPI. Korzystając z narzędzi zawartych w systemie Kali, odszukaj jej adres IP. W tym celu użyj narzędzia Netdiscover. Potwierdź wynik, wpisując w przeglądarce znaleziony adres IP (w moim przypadku był to 192.168.50.35).

Następnie za pomocą programu Nmap wykonaj ogólne skanowanie wykrywające, aby dowiedzieć się, z czym masz do czynienia. Jak wspomniałem wcześniej, polecenie `nmap -sC -sV 192.168.50.35 -oA crapi_scan` identyfikuje usługi, skanuje interfejs za pomocą domyślnych skryptów i zapisuje wyniki w kilku formatach na potrzeby późniejszej analizy. Poniżej jest przedstawiony przykładowy wynik skanowania.

```
Nmap scan report for 192.168.50.35
Host is up (0.00043s latency).
Not shown: 994 closed ports
PORT     STATE SERVICE    VERSION
1025/tcp open  smtp       Postfix smtpd
|_smtp-commands: Hello nmap.scanme.org, PIPELINING, AUTH PLAIN,
5432/tcp open  postgresql PostgreSQL DB 9.6.0 or later
| fingerprint-strings:
|   SMBProgNeg:
|     SFATAL
|     VFATAL
|     C0A000
|     Munsupported frontend protocol 65363.19778: server supports 2.0 to 3.0
|     Fpostmaster.c
|     L2109
|_    RProcessStartupPacket
8000/tcp open  http-alt   WSGIServer/0.2 CPython/3.8.7
| fingerprint-strings:
|   FourOhFourRequest:
```

```
|     HTTP/1.1 404 Not Found
|     Date: Tue, 25 May 2021 19:04:36 GMT
|     Server: WSGIServer/0.2 CPython/3.8.7
|     Content-Type: text/html
|     Content-Length: 77
|     Vary: Origin
|     X-Frame-Options: SAMEORIGIN
|     <h1>Not Found</h1><p>The requested resource was not found on this server.</p>
|   GetRequest:
|     HTTP/1.1 404 Not Found
|     Date: Tue, 25 May 2021 19:04:31 GMT
|     Server: WSGIServer/0.2 CPython/3.8.7
|     Content-Type: text/html
|     Content-Length: 77
|     Vary: Origin
|     X-Frame-Options: SAMEORIGIN
|     <h1>Not Found</h1><p>The requested resource was not found on this server.</p>
```

Uzyskany wynik pokazuje, że interfejs ma kilka otwartych portów, m.in. 1025, 5432, 8000, 8080, 8087 i 8888. Dodatkowo narzędzie Nmap dostarczyło informacji, że port numer 1025 jest wykorzystywany przez usługę pocztową SMTP, port numer 5432 przez bazę PostgreSQL, a pozostałe porty są używane do odbierania zapytań HTTP. Skanowanie ujawniło również, że jest wykorzystywane oprogramowanie serwerowe CPython, WSGIServer i OpenResty.

Zwróć uwagę na odpowiedź wysłaną z portu 8080, którego nagłówki sugerują istnienie interfejsu API:

```
Content-Type: application/json and "error": "Invalid Token" }.
```

W następnym kroku wykonaj pełne skanowanie, aby sprawdzić, czy za nietypowym portem nie kryje się jakaś usługa:

```
$ nmap -p- 192.168.50.35
Nmap scan report for 192.168.50.35
Host is up (0.00068s latency).
Not shown: 65527 closed ports
PORT      STATE SERVICE
1025/tcp  open  NFS-or-IIS
5432/tcp  open  postgresql
8000/tcp  open  http-alt
8025/tcp  open  ca-audit-da
8080/tcp  open  http-proxy
8087/tcp  open  simplifymedia
8888/tcp  open  sun-answerbook
27017/tcp open  mongod
```

Pełne skanowanie ujawniło serwer MailHog i bazę danych MongoDB wykorzystujące nietypowe porty o numerach odpowiednio 8025 i 27017. Te informacje mogą Ci się przydać w następnych ćwiczeniach, gdy będziesz eksplorował interfejs API.

Wyniki wstępnego skanowania zawierają informację o aplikacji internetowej wykorzystującej port numer 8080, co prowadzi do następnego logicznego kroku: ręcznej analizy. Sprawdź wszystkie porty, tj. 8000, 8025, 8080, 8087 i 8888, z których nadeszły odpowiedzi na zapytania HTTP wysłane przez narzędzie Nmap. W tym celu wpisz w przeglądarce następujące adresy:

http://192.168.50.35:8000

http://192.168.50.35:8025

http://192.168.50.35:8080

http://192.168.50.35:8087

http://192.168.50.35:8888

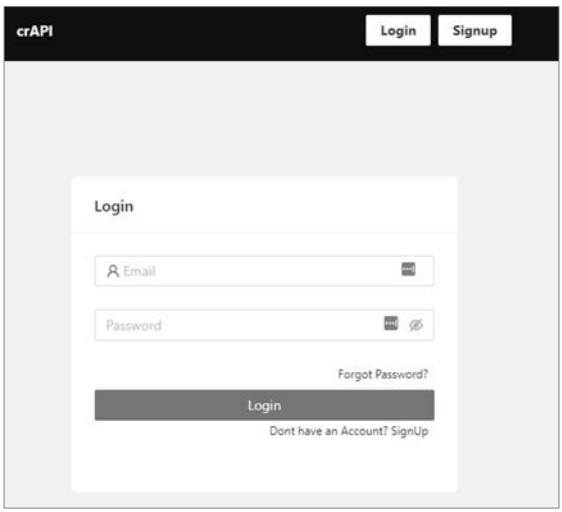
Pod adresem zawierającym port numer 8000 znajduje się strona, na której jest widoczny jedynie komunikat *The requested resource was not found on this server* (żądany zasób nie został znaleziony na tym serwerze).

Port numer 8025 jest wykorzystywany przez serwer MailHog. Pod tym adresem znajduje się strona z wiadomością e-mail *Welcome to crAPI* (witaj w crAPI). Zajmiemy się nią w jednym z następnych ćwiczeń.

Po otwarciu adresu z portem numer 8080 pojawia się komunikat `{ "error": "Invalid Token" }`, taki sam jak uzyskany podczas pierwszego skanowania.

Przy próbie otwarcia adresu z portem numer 8087 pojawia się błąd *404 Page not found*.

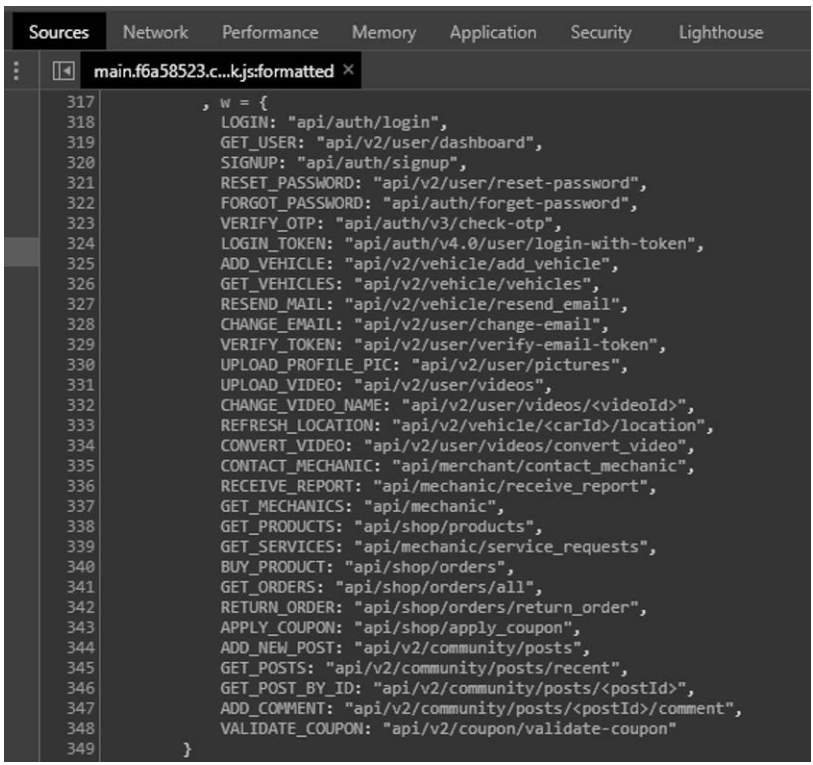
Wreszcie pod adresem z portem numer 8888 znajduje się strona logowania do aplikacji crAPI, pokazana na rysunku 6.22.



Rysunek 6.22. Główna strona aplikacji crAPI

Na podstawie błędów i informacji dotyczących autoryzacji możesz założyć, że większy użytek zrobisz z otwartych portów jako uwierzytelniony użytkownik.

Teraz zbadaj za pomocą narzędzi Chrome DevTools użyte na stronie pliki JavaScript. Aby je wyświetlić, kliknij zakładkę *Network* i odśwież stronę. Kliknij prawym przyciskiem myszy interesujący Cię plik i wybierz polecenie *Open in Sources panel* (otwórz w panelu Sources). Otwórz w ten sposób plik */static/js/main.f6a58523.chunk.js* i poszukaj w nim punktów końcowych interfejsu API (patrz rysunek 6.23).



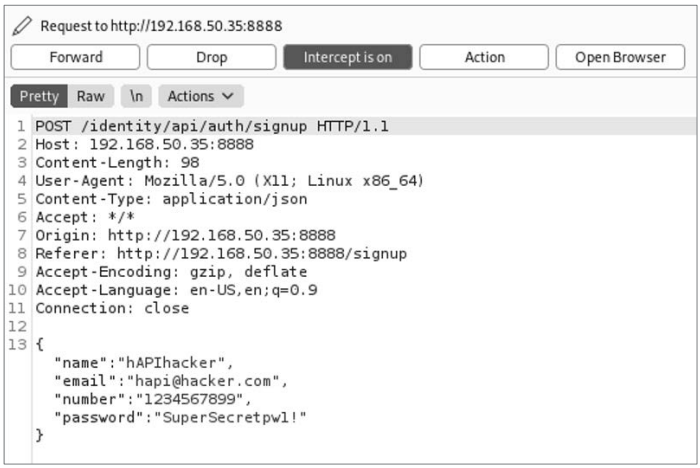
Rysunek 6.23. Główny plik JavaScript aplikacji crAPI

Gratulacje! Przeprowadziłeś aktywny rekonesans za pomocą narzędzi Chrome DevTools i odkryłeś swój pierwszy interfejs API. Dodatkowo, przeszukując w zwykły sposób plik źródłowy, znalazłeś wiele unikatowych punktów końcowych.

Gdy przyjrzysz się plikowi, zauważysz interfejsy API wykorzystywane w procesie rejestracji użytkownika. Dobrym kolejnym krokiem byłoby przechwycenie wykorzystywanych w tym procesie zapytań, aby zobaczyć interfejs API w akcji. W tym celu kliknij na stronie aplikacji crAPI przycisk *Signup*, wpisz swoje imię, nazwisko, adres e-mail, telefon i hasło. Zanim klikniesz przycisk *Signup* (zarejestruj) w dolnej części okna, otwórz program Burp Suite, a w rozszerzeniu FoxyProxy przeglądarki

wybierz serwer pośredniczący Hackz, aby przechwycić komunikację. Teraz możesz kliknąć przycisk *Signup*.

Jak pokazuje rysunek 6.24, aplikacja crAPI wysyła podczas rejestracji użytkownika zapytanie `POST` do punktu */identity/api/auth/signup*. To zapytanie, przechwycone w programie Burp Suite, potwierdza istnienie interfejsu API. Dodatkowo jest to informacja z pierwszej ręki o przeznaczeniu punktu końcowego.



Rysunek 6.24. Przechwycone za pomocą programu Burp Suite zapytanie wysłane podczas rejestracji użytkownika

Dobra robota! Nie tylko odkryłeś interfejs API, ale także zbadałeś interakcję z nim. W następnym ćwiczeniu poznasz jego funkcjonalności i zidentyfikujesz słabe punkty. Zachęcam Cię do przetestowania innych narzędzi na tym przykładzie. Potrafisz wykryć interfejs w inny sposób?

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

X

Y

Z


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

TA KSIĄŻKA TO ŻYŁA ZŁOTA DLA HAKERÓW INTERFEJSÓW API!

Chris Roberts, vCISO

CZY JUŻ WIESZ, JAK ATAKUJE PRAWDZIWY WRÓG?

Jeśli najcenniejszym zasobem świata są dane, to interfejsy API można porównać do cyfrowych rurociągów przesyłających cenny towar pomiędzy kontrahentami. Ta technologia umożliwia wymianę danych między różnymi aplikacjami, zmieniła sposób projektowania oprogramowania. API mają jednak poważną wadę: podatność na ataki. Bardzo często hakerzy traktują je jak otwarte drzwi do cennych danych. A to najkrótsza droga do katastrofy.

Ta książka stanowi przyspieszony kurs testowania bezpieczeństwa API aplikacji internetowych. Dzięki niej przygotujesz się do testowania interfejsów, wyszukiwania błędów i zwiększania bezpieczeństwa własnoręcznie napisanych interfejsów. Dowiesz się, jak REST API działają w środowisku produkcyjnym i jakie problemy wiążą się z ich bezpieczeństwem. Zbudujesz nowoczesne środowisko testowe złożone z programów: Burp Suite, Postman, Kiterunner i OWASP Amass, przydatnych do rekonesansu, analizy punktów końcowych i zakłócania interfejsów. Następnie nauczysz się przeprowadzać ataki na procesy uwierzytelniania, luki w procedurach biznesowych i typowe słabe punkty interfejsów. Dowiesz się też, jak tworzyć skrypty międzyinterfejsowe, a także jak prowadzić masowe przypisania i wstrzykiwanie danych.

Dzięki książce nauczysz się:

- identyfikować użytkowników i punkty końcowe API
- wykrywać nadmierną ekspozycję danych
- atakować proces uwierzytelniania
- wstrzykiwać zapytania NoSQL
- przeprowadzać inżynierię wsteczną interfejsu API
- wykrywać błędy w procedurach biznesowych

Corey Ball zajmuje się testami penetracyjnymi w Moss Adams. Od ponad dziesięciu lat specjalizuje się w bezpieczeństwie cybernetycznym w lotnictwie, energetyce, finansach i administracji państwowej. Posiada certyfikaty branżowe: OSCP, CCISO, CEH, CISA, CISM, CRISC i CGEIT.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-8322-408-4
9 788383 224084
Cena: 89,00 zł
no starch press