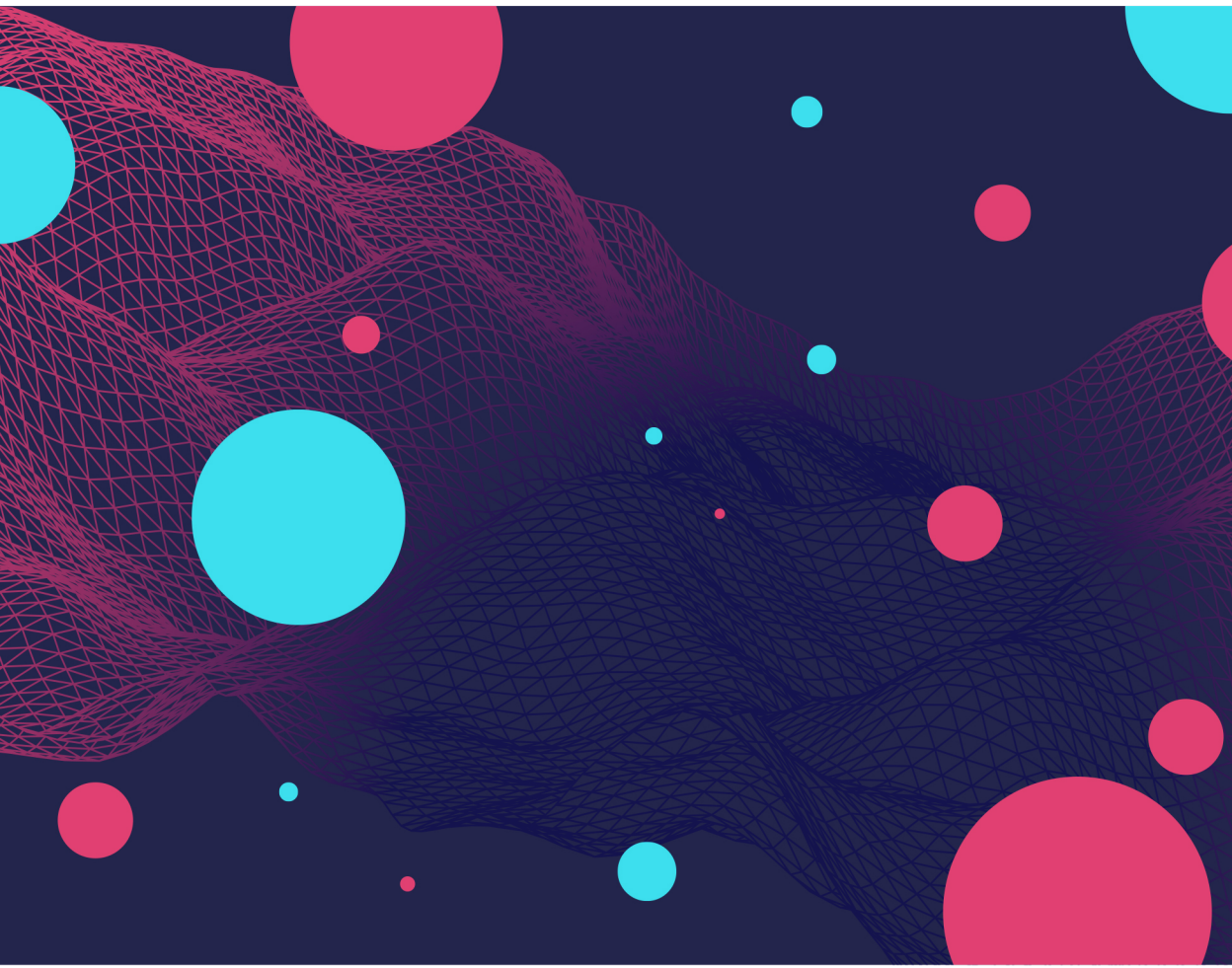
# JAK RADZIĆ SOBIE Z PODATNOŚCIAMI

## na przykładzie Bug Bounty, Otwarte przekierowanie i HTTP Parameter Pollution



PWNING_WEEK 2021

PWN

# 1

# PODSTAWY BUG BOUNTY

Jeżeli jesteś nowy w świecie hackingu, podstawowe zrozumienie tego, jak internet działa i co się dzieje „pod maską”, kiedy wpisujesz adres URL w przeglądarce, będzie Ci bardzo pomocne. Nawigacja po stronach internetowych może wydawać się prosta, lecz wbrew pozorom angażuje ona wiele ukrytych procesów, takich jak przygotowanie żądania HTTP, identyfikacja domeny, do której wysłać żądanie, tłumaczenie domeny na adres IP, przesłanie żądania, renderowanie odpowiedzi i tak dalej.

W tym rozdziale poznasz podstawowe koncepcje i terminologię, takie jak podatności, programy bug bounty, klient, serwer, adres IP oraz HTTP. Zrozumiesz, w jaki sposób wykonywanie niezamierzonych akcji i dostarczanie nieoczekiwanego wejścia lub dostępu do prywatnych informacji

może skutkować podatnościami bezpieczeństwa. Następnie zobaczymy, co się dzieje, kiedy wpisujesz adres URL w pasku adresu swojej przeglądarki, włączając w to, jak wyglądają żądania i odpowiedzi HTTP, jak i również różne metody HTTP. Zakończymy rozdział ze zrozumieniem tego, co oznacza stwierdzenie, że protokół HTTP jest bezstanowy.

## Podatności i Bug Bounty

*Podatność* jest to słabość aplikacji, która pozwala złośliwej osobie wykonać niedozwolone akcje lub uzyskać dostęp do informacji, których normalnie nie powinna być stanie uzyskać.

Podczas nauki i testowania aplikacji pamiętaj o tym, że podatności mogą skutkować działaniami zamierzonymi lub niezamierzonymi. Dla zobrazowania: zmiana parametru / pola ID identyfikatora rekordu w celu uzyskania informacji, do których nie powinieneś mieć dostępu, jest przykładem niezamierzonej akcji.

Załóżmy, że utworzyłeś profil ze swoją nazwą, e-mailem, datą urodzenia i adresem. Strona zachowałaby te informacje jako prywatne i umożliwiłaby do nich dostęp tylko Twoim znajomym. Jeśli jednak aplikacja internetowa pozwala komukolwiek dodawać Cię jako swojego znajomego bez Twojej zgody, byłaby to podatność bezpieczeństwa. Mimo że strona strzeże Twoich prywatnych danych przed nieznajomymi, pozwalając dowolnej osobie na dodanie Cię do swoich znajomych bez Twojej zgody, jednocześnie umożliwia ona każdemu dostęp do informacji o Tobie. Kiedy testujesz stronę internetową, zawsze rozważaj to, jak ktoś mógłby nadużyć obecnej funkcjonalności.

*Bug bounty* to nagroda, którą strona lub firma przyznaje każdemu, kto w etyczny sposób znajdzie podatność i zgłosi ją do administracji bądź firmy. Nagrody są zazwyczaj pieniężne i wahają się od dziesiątek, aż po tysiące dolarów. Inne przykłady nagród to między innymi kryptowaluty, mile lotnicze, punkty nagród bądź inne dobra cyfrowe.

Kiedy firma oferuje nagrody za znalezienie podatności, tworzy wyznaczony do tego *program* – termin, używany w tej książce, by oznaczyć zasady i strukturę ustanowione przez firmy dla ludzi, którzy chcą testować firmy na te podatności. Zwróć uwagę na to, że różni się to od firm, które wykorzystują programy upubliczniania podatności (Vulnerability Disclosure Program – VDP). Programy bug bounty oferują nagrody pieniężne, podczas gdy VDP nie oferuje płatności (aczkolwiek firmy mogą nagradzać rzeczami). VDP to najzwyklejszy sposób dla uczciwych hakerów na zgłaszanie znalezionych podatności, aby dana firma mogła je naprawić. Mimo że nie wszystkie zgłoszenia przedstawione w tej książce zostały nagrodzone, wszystkie pochodzą od hakerów biorących udział w programach bug bounty.

## Klient i serwer

Twoja przeglądarka korzysta z internetu, który jest siecią komputerów wysyłających do siebie wiadomości. Wiadomości te nazywamy *pakietami*. Pakiety zawierają dane, które wysyłasz, i informacje o tym, skąd te dane wychodzą i dokąd zmierzają. Każdy komputer w internecie ma adres do odbierania pakietów. Niektóre komputery akceptują tylko określone typy pakietów, a jeszcze inne akceptują pakiety tylko z ograniczonej listy innych komputerów. Kiedy komputer odbiera pakiet, zależy to tylko od niego, co z nim zrobi i jak odpowie. Na potrzeby tej książki skupimy się tylko na danych zawartych w pakietach (wiadomościach HTTP), a nie na samych pakietach.

Będę się odwoływał do tych komputerów jako do klientów lub serwerów. Komputer inicjujący żądanie jest zazwyczaj nazywany *klientem,* niezależnie od tego, czy żądanie jest zainicjowane przez przeglądarkę, wiersz poleceń lub coś innego. Nazwa *serwery* odnosi się do stron i aplikacji internetowych odbierających żądania.

Ponieważ internet może mieć dowolną liczbę rozmawiających ze sobą komputerów, potrzebujemy wytycznych do tego, jak komputery powinny się komunikować przez internet. Przyjmują one formę dokumentów *RFC* (*Request for Comment* – prośba o komentarze), które definiują standardy tego, w jaki sposób komputery powinny się zachowywać. Na przykład *protokół przesyłania dokumentów hipertekstowych* – w skrócie *HTTP* (*Hypertext Transfer Protocol*) – określa, jak Twoja przeglądarka internetowa komunikuje się ze zdalnym serwerem przy użyciu *protokołu internetowego IP* (*Internet Protocol*). W tym przypadku zarówno klient, jak i serwer muszą się zgodzić na implementację tych samych standardów, dzięki czemu są w stanie zrozumieć wysyłane i odbierane przez siebie pakiety.

## Co się dzieje, kiedy odwiedzasz stronę

Ponieważ w tej książce skupiamy się na żądaniach HTTP, ta sekcja dostarczy Ci wysokopoziomowy wgląd na cały proces, który następuje po wpisaniu adresu URL w przeglądarce.

### *Krok 1: Identyfikacja domeny internetowej*

Kiedy wpiszesz *http://www.google.com/*, Twoja przeglądarka ustala domenę z adresu URL. *Domena* identyfikuje stronę, którą próbujesz odwiedzić, i musi przy tym przestrzegać określonych zasad zdefiniowanych przez RFC. Na przykład domena może zawierać tylko znaki alfanumeryczne i podkreślenia. Jedyny wyjątek stanowią zinternacjonalizowane domeny, które wychodzą poza zakres tej książki. Jeśli chcesz dowiedzieć się więcej na ten

temat, sprawdź dokument RFC 3490, który definiuje ich użycie. W tym przypadku domeną jest *www.google.com*. Domena służy jako jedna z dróg do znalezienia adresu serwera.

### Krok 2: Ustalenie adresu IP

Po ustaleniu domeny, Twoja przeglądarka używa IP do sprawdzenia *adresu IP* powiązanego z tą stroną. Ten proces jest określany jako ustalanie adresu IP i każda domena w internecie musi zostać przetłumaczona na adres IP, aby działała.

Istnieją dwa rodzaje adresów IP, definiowane przez protokół internetowy w wersji 4 (IPv4) i w wersji 6 (IPv6). Adresy IPv4 są skonstruowane z czterech liczb połączonych kropkami, gdzie każdy numer jest z przedziału od 0 do 255. IPv6 jest najnowszą wersją protokołu internetowego. Został stworzony po to, by rozwiązać problem kończących się adresów IPv4. Adresy IPv6 są stworzone z ośmiu grup czterocyfrowych liczb dziesiętnych, oddzielonych dwukropkami, lecz istnieją metody na ich skrócenie. Na przykład 8.8.8.8 jest adresem IPv4, a 2001:4860:4860::8888 jest skróconym adresem IPv6.

Aby znaleźć adres IP, używając domeny, Twój komputer wysyła żądanie do serwerów *DNS* (*Domain Name System*) składających się ze specjalnych serwerów w internecie, które mają rejestr wszystkich domen i odpowiadających im adresów IP. Powyższe adresy IPv4 i IPv6 są serwerami DNS Google.

W tym przypadku serwer DNS, do którego się łączysz, dopasowałby *www.google.com* do adresu IPv4 216.58.201.228 i wysłałby go z powrotem do Twojego komputera. By dowiedzieć się więcej na temat adresu IP konkretnej strony, możesz użyć komendy `dig A site.com` w swoim terminalu i zamienić `site.com` na stronę, którą chcesz sprawdzić.

### Krok 3: Nawiązanie połączenia TCP

Następnie komputer próbuje nawiązać połączenie *TCP* (*Transmission Control Protocol* – protokół kontroli transmisji) z adresem IP przez port 80, ponieważ odwiedziłeś stronę, używając *http://*. Szczegóły na temat TCP nie są istotne oprócz tego, że jest to kolejny protokół, który definiuje sposób, w jaki komputery komunikują się ze sobą. TCP zapewnia dwukierunkową komunikację, dzięki czemu odbiorcy wiadomości mogą zweryfikować kompletność przesyłanej informacji podczas transmisji.

Serwer, do którego wysyłasz żądanie, może być w trakcie obsługi wielu usług (traktuj usługę jako program komputerowy), więc używa *portów* do identyfikacji konkretnych usług, aby odbierać żądania. Możesz je sobie wyobrazić jako drzwi serwera do internetu. Bez portów usługi musiałyby konkurować ze sobą o to, która informacja zostanie wysłana. Oznacza to, że potrzebujemy kolejnego standardu do zdefiniowania tego, jak usługi współpracują ze sobą i upewnić się, że dane dla jednej usługi nie są kradzione

PWNING_WEEK 2021

# Dołącz do uczestników PWNing Week 2021!

**Grupa PWN** zaprasza do udziału w **PWNing Week 2021** – jednym z ważniejszych w Polsce wydarzeń z obszaru cyberbezpieczeństwa. Tegoroczna edycja odbędzie się online i jest rozszerzoną formułą odbywającej się w ubiegłych latach konferencji **Security PWNing**.

**Dowiedz się więcej**

Tegoroczną edycję kierujemy szczególnie do kadry zarządzającej, osób odpowiedzialnych za wybór i zakup rozwiązań z zakresu bezpieczeństwa i monitoringu sieci oraz poszukujących nowych rozwiązań chmurowych.

**PWNing WEEK 2021 to:**

- Security Conference (17-19 listopada)
- strefa EXPO (17-19 listopada)
- warsztaty, szkolenia i inne wydarzenia towarzyszące (15-16 listopada, 22-26 listopada)

**Tematyka PWNing Security Conference:**

- Bezpieczeństwo danych w dobie pandemii i masowej pracy zdalnej
- Rozwiązania chmurowe
- Narzędzia wspomagające bezpieczeństwo danych

**Zarejestruj się**

Konferencję poprowadzi dla Was
**dr Maciej Kawecki**

przez inne. Na przykład port 80 jest standardowym portem do wysyłania i odbierania żądań HTTP. Innym powszechnym portem jest 443, który jest używany do zaszyfrowanych żądań HTTPS. Mimo że port 80 jest standardem dla HTTP i 443 jest standardem dla HTTPS, komunikacja TCP może następować na dowolnym porcie, zależnie od konfiguracji aplikacji przez administratora.

Możesz nawiązać swoje własne połączenie TCP do strony internetowej przez port 80 przy użyciu terminala, wpisując komendę `nc <ADRES IP> 80`. Komenda `nc` używa narzędzia Netcat do utworzenia połączenia sieciowego do odczytywania i pisania wiadomości.

### Krok 4: Wysyłanie zapytania HTTP

Kontynuując przykład ze stroną *http://www.google.com/*, jeśli połączenie w kroku 3 zostało nawiązane pomyślnie, Twoja przeglądarka powinna przygotować i wysłać żądanie HTTP, tak jak w listingu 1.1:

```
❶ GET / HTTP/1.1
❷ Host: www.google.com
❸ Connection: keep-alive
❹ Accept: application/html, */*
❺ User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/72.0.3626.109 Safari/537.36
```

*Listing 1.1. Wysyłanie żądania HTTP*

Przeglądarka przygotowuje żądanie `GET` do ścieżki `/` ❶, która jest główną ścieżką dostępu strony. Zawartość strony jest organizowana ścieżkami, dokładnie tak samo jak foldery i pliki na Twoim komputerze. Gdy wchodzisz w kolejne foldery, ścieżka, którą podążasz, jest oznaczona przez zapisywanie nazw kolejnych folderów i dopisywanie `/` na ich końcu. Kiedy odwiedzasz pierwszą stronę na witrynie internetowej, dostajesz się do głównej ścieżki dostępu, którą jest po prostu `/`. Przeglądarka również sygnalizuje, że używa wersji 1.1 protokołu HTTP. Żądanie `GET` jedynie otrzymuje informacje. Dowiesz się o tym więcej później.

Nagłówek *host* ❷ zawiera dodatkowe informacje, które są wysłane jako część zapytania. HTTP 1.1 potrzebuje ich, by określić, gdzie serwer dla danego adresu IP powinien przesłać żądanie, ponieważ adresy IP mogą hostować kilka domen na raz. Nagłówek *connection* ❸ informuje, żeby utrzymać połączenie otwarte, aby uniknąć potrzeby ciągłego otwierania i zamykania połączeń.

Możesz zobaczyć oczekiwany format odpowiedzi w ❹. W tym przypadku oczekujemy `application/html`, lecz zaakceptujemy każdy format, co zostało oznaczone symbolem (`*/*`). Istnieją tysiące możliwych rodzajów zawartości, ale dla naszych potrzeb najczęściej będziesz wykorzystywał:

`application/html`, `application/json`, `application/octet-stream` oraz `text/plain`. W końcu User-Agent ❺ określa oprogramowanie odpowiedzialne za wysyłanie żądań.

### Krok 5: Odpowiedź serwera

W odpowiedzi na nasze żądanie serwer powinien odpowiedzieć czymś w rodzaju listingu 1.2:

```
❶ HTTP/1.1 200 OK
❷ Content-Type: text/html
<html>
  <head>
    <title>Google.com</title>
  </head>
  <body>
  ❸ --cięcie--
  </body>
</html>
```

*Listing 1.2. Odpowiedź serwera*

Tutaj otrzymaliśmy odpowiedź HTTP z kodem statusu 200 ❶. Kod statusu jest istotny, ponieważ mówi o rezultacie odpowiedzi. Co również jest opisane przez RFC, kody te zazwyczaj mają trzycyfrowy numer, który zaczyna się 2, 3, 4 lub 5. Mimo że nie ma wyznaczonych standardów do używania konkretnych kodów, 2xx najczęściej oznacza sukces.

Ponieważ nie istnieje żaden wymóg co do tego, jak serwer implementuje użycie własnych kodów HTTP, możesz czasem zobaczyć odpowiedź z kodem 200, nawet jeśli dalsza odpowiedź HTTP opisuje błąd. *Treść wiadomości HTTP* jest tekstem powiązanym z żądaniem lub odpowiedzią ❸. W tym przypadku usunęliśmy zawartość i zamieniliśmy ją na `--cięcie--` ze względu na wielkość treści wiadomości od Google. Zawartość odpowiedzi to zazwyczaj HTML dla strony internetowej, ale może być również tekstem JSON dla API, zawartością do pobrania i tak dalej.

Nagłówek Content-Type ❷ informuje przeglądarkę o rodzaju treści. Rodzaj treści określa, w jaki sposób przeglądarka będzie renderować zawartość. Przeglądarki nie zawsze jednak używają wartości zwróconej przez aplikację; zamiast tego przeglądarka wykonuje tzw. *MIME sniffing* przez czytanie pierwszych bitów treści wiadomości, aby wykryć jej rodzaj samemu. Aplikacje mogą wyłączyć tę funkcję przez dołączanie nagłówka *X-Content-Type-Options:nosniff*, który w obecnym przykładzie nie został dołączony.

Kolejny kod statusu, zaczynający się od 3 oznacza przekierowanie, co instruuje Twoją przeglądarkę do wykonania dodatkowego zapytania.

Na przykład, jeśli Google chciałby odesłać Cię na stałe z jednego adresu URL na drugi, użyłby odpowiedzi 301. Dla odróżnienia 302 oznacza tymczasowe przekierowanie.

Gdy otrzymujesz odpowiedź 3xx, Twoja przeglądarka powinna wykonać nowe żądanie HTTP na adres URL wskazany w nagłówku `Location`:

```
HTTP/1.1 301 Found
Location: https://www.google.com/
```

Odpowiedzi zaczynające się od 4 najczęściej wskazują na błąd użytkownika, tak jak kod 403, kiedy żądanie nie będzie zawierało odpowiednich danych do wykonania uwierzytelnienia, mimo przesłania prawidłowego żądania HTTP. Odpowiedzi zaczynające się 5 oznaczają błąd po stronie serwera, przykładem może być kod 503, który oznacza, że serwer nie jest w stanie obsłużyć otrzymanego żądania.

### *Krok 6: Renderowanie odpowiedzi*

Ponieważ serwer wysłał odpowiedź z kodem 200 i rodzajem `text/html`, nasza przeglądarka rozpocznie renderowanie wiadomości, którą otrzymała. Treść odpowiedzi mówi przeglądarce, co powinno zostać wyświetlone użytkownikowi.

Dla naszego przykładu byłby to HTML do budowy strony; kaskadowe arkusze stylów (CSS) do stylizacji i układu; JavaScript do dodatkowej funkcjonalności i mediów, takich jak zdjęć lub filmów. Serwer może również odpowiedzieć inną zawartością, taką jak XML, lecz dla tego przykładu zostaniemy przy podstawach. Rozdział 11 omawia XML w szczegółach.

W przypadku gdy strona internetowa ma zewnętrzną zawartość, taką jak CSS, JavaScript czy media, przeglądarka będzie musiała wykonać dodatkowe żądania HTTP dla wszystkich wymaganych plików przez daną witrynę. W trakcie gdy przeglądarka wysyła wymagane żądania, wciąż będzie analizować odpowiedź i wyświetlać zawartość jako stronę. W tym przypadku wyrenderuje główną stronę Google, *www.google.com*.

Zauważ, że JavaScript jest językiem skryptowym wspieranym przez każdą przeglądarkę. JavaScript pozwala stronom na posiadanie dynamicznej funkcjonalności, włączając w to możliwość aktualizacji elementów na stronie bez potrzeby odświeżania całej strony, sprawdzanie siły Twojego hasła (w przypadku niektórych stron) i wiele innych. Tak jak inne języki programowania, JavaScript ma wbudowane funkcje, może przechowywać wartości w zmiennych i potrafi uruchamiać kod w odpowiedzi na interakcje na stronie. Ma również dostęp do zróżnicowanych interfejsów programowania (API). Interfejsy te pozwalają językowi JavaScript na interakcję z innymi systemami, z czego jednym z najważniejszych jest obiektowy model dokumentu (DOM – Document Object Model).

DOM umożliwia językowi JavaScript dostęp i manipulację kodem HTML i CSS należącymi do strony. Jest to bardzo istotne, ponieważ jeśli intruz jest w stanie uruchomić własny kod JavaScript, będzie on mógł uzyskać dostęp do DOM-u i wykonać akcje na stronie w imieniu ofiary. Rozdział 7 rozwija to zagadnienie dokładniej.

## Żądania HTTP

Zgoda między klientem a serwerem w użyciu wiadomości HTTP włącza definiowanie metod żądań. *Metoda* określa powód żądania przesyłanego przez klienta oraz to, czego oczekuje jako pozytywny rezultat. Dla zobrazowania w listingu 1.1 wysłaliśmy zapytanie `GET` do *http://google.com/*, implikując w ten sposób, że oczekujemy jedynie zawartości *http://google.com/* w odpowiedzi, bez wykonywania dodatkowych akcji. Internet jest zaprojektowany jako interfejs między zdalnymi komputerami, z tego względu metody żądań zostały zbudowane i zaimplementowane po to, by rozpoznawać podejmowane akcje.

Standard HTTP definiuje następujące metody: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `TRACE`, `CONNECT` i `OPTIONS` (`PATCH` było również rozważane, lecz nie jest wystarczająco często implementowane). W momencie pisania tej książki, przeglądarki wysyłają jedynie żądania `GET` i `POST` przy użyciu HTML. Jakiekolwiek żądanie `PUT`, `PATCH` lub `DELETE` może być wywołane jedynie przez JavaScript. Skupimy się na tym później, kiedy będziemy omawiać przykłady podatności aplikacji używających tych metod.

Następna część zawiera zwięzły przegląd rodzajów metod, które znajdziesz w tej książce.

### *Metody żądań*

Metoda `GET` wyszukuje informacje zidentyfikowane przez *URI* żądania (*Uniform Resource Identifier* – ujednolicony identyfikator zasobów). Termin URI jest często synonimicznie używany z URL (*Uniform Resource Locator* – ujednolicony format adresowania zasobów). Technicznie rzecz biorąc, URL jest jednym z rodzajów URI, który określa zasoby, i zawiera sposób na odnalezienie ich przez lokalizację w sieci. Na przykład *http://google.com/<example>/file.txt* oraz *<example>/file.txt* są poprawnymi adresami URI. Jednak tylko *http://google.com/<example>/file.txt* jest poprawnym URL-em, ponieważ wskazuje, jak odnaleźć zasoby przez domenę *http://www.google.com*. Pomijając niuanse, kiedy będziemy w tej książce odnosić się do jakichkolwiek identyfikatorów, będziemy używać adresów *URL*.

Choć nie ma sposobu na wymuszenie tego warunku, żądania `GET` nie powinny modyfikować danych; powinny jedynie otrzymywać dane z serwera i zwracać je jako zawartość treści HTML. Dla zobrazowania na stronach mediów społecznościowych żądanie `GET` powinno jedynie zwracać nazwę Twojego profilu, nie aktualizować go. To zachowanie jest

krytyczne w przypadku ataków *CSRF* (*Cross-Site Request Forgery*) omawianych w rozdziale 4. Odwiedzenie jakiegokolwiek linku bądź adresu URL (o ile nie zostało wykonane przez JavaScript) sprawia, że Twoja przeglądarka wysyła żądanie `GET` do zamierzonego serwera. Zachowanie to jest bardzo istotne w przypadku podatności na otwarte przekierowania, omawianych w rozdziale 2.

Metoda `HEAD` jest identyczna z metodą `GET` z wyjątkiem tego, że serwer nie może przesłać treści wiadomości w odpowiedzi.

Metoda `POST` uruchamia niektóre funkcje ustalone przez połączony serwer. Innymi słowy, często na stronach podejmujesz pewnego rodzaju backendowe czynności, takie jak utworzenie komentarza, rejestracja użytkownika, usunięcie konta i tak dalej. Reakcja serwera na otrzymane od nas żądanie `POST` może być różna. Czasem serwer nie podejmie żadnej akcji, na przykład w przypadku, gdy wysłane żądanie `POST` spowodowało błąd podczas jego przetwarzania. W takiej sytuacji zmiany na serwerze nie zostaną zapisane.

Metoda `PUT` wywołuje funkcje, które odnoszą się do istniejących już zapisów w zdalnej aplikacji bądź witrynie. Na przykład może być użyta podczas aktualizacji profilu, utworzenia posta na blogu bądź czymkolwiek, co już istnieje. Ponownie podjęte akcje mogą być różne, a czasem nie zostanie podjęta żadna.

Metoda `DELETE` wysyła żądanie dotyczące usunięcia zdalnego zasobu zidentyfikowanego przez URI.

Metoda `TRACE` jest kolejną rzadką metodą; jest używana do odesłania wysłanego żądania z powrotem do źródła. Pozwala wykonawcy żądania na zobaczenie tego, co jest odbierane przez serwer i użyć tej informacji do testów bądź zbierania danych diagnostycznych.

Metoda `CONNECT` jest zarezerwowana do użytku przez *proxy*, serwer pośredniczący między żądaniami. Ta metoda rozpoczyna dwukierunkową komunikację z żądanym źródłem. Na przykład metoda `CONNECT` może mieć dostęp do stron, które używają HTTPS przez proxy.

Metoda `OPTIONS` wysyła żądanie do serwera o informację o dostępnych możliwościach komunikacji. Na przykład, używając `OPTIONS`, możesz dowiedzieć się, czy serwer akceptuje zapytania `GET`, `POST`, `PUT`, `DELETE` lub `OPTIONS`. Ta metoda nie wskaże jednak, czy serwer akceptuje zapytania `HEAD` lub `TRACE`. Przeglądarki internetowe automatycznie wysyłają ten rodzaj żądania, gdy zauważą specyficzny rodzaj zawartości, taki jak `application/json`. Metoda `OPTIONS`, należąca do mechanizmu *preflighted requests*, jest omówiona dokładniej w rozdziale 4, gdyż służy jako ochrona przed atakami CSRF.

### Protokół HTTP jest bezstanowy

Żądania HTTP są *bezstanowe*, co oznacza, że każde wysłane żądanie do serwera jest traktowane jako całkowicie nowe. Serwer nie zna historii komunikacji z Twoją przeglądarką, kiedy dobiera żądanie. Dla większości

stron jest to problematyczne, ponieważ chcą one pamiętać kim jesteś. Bez tego musiałbyś wpisywać swoją nazwę użytkownika i hasło za każdym razem, kiedy prześlesz żądanie HTTP. Oznacza to również, że wszystkie dane wymagane do wykonania żądania HTTP musiałyby zostać przeładowane z każdym żądaniem, które klient wyśle do serwera.

Do objaśnienia tego mylącego konceptu rozważ następujący przykład: jeśli Ty i ja prowadzilibyśmy bezstanową konwersację, przed wypowiedzeniem jakiegokolwiek zdania musielibyśmy zacząć od „Nazywam się Peter Yaworski; przed chwilą rozmawialiśmy o hakowaniu". Wtedy, musiałbyś *przeładować* wszystkie informacje dotyczące naszej dyskusji o hakowaniu. Pomyśl tylko co Adam Sandler musiał robić dla Drew Barrymore każdego poranka w *50 Pierwszych Randkach* (jeśli nie widziałeś tego filmu, to powinieneś).

W celu uniknięcia konieczności wysyłania swojej nazwy użytkownika razem z hasłem dla każdego żądania HTTP strony internetowe korzystają z plików cookies lub podstawowej autoryzacji, które omówimy szczegółowo w rozdziale 4.

**UWAGA** *Specyfika tego, w jaki sposób dane są kodowane przy użyciu base64, jest poza zakresem tej książki, prawdopodobnie jednak napotkasz zawartość zakodowaną przez base64 podczas hakowania. Jeśli tak, powinieneś ją rozszyfrować. Zapytanie w Google „base64 decode" powinno dostarczyć Ci wystarczającej liczby metod i narzędzi do wykonania tej czynności.*

## Podsumowanie

Powinieneś teraz już wiedzieć, jak w podstawowy sposób działa komunikacja z serwerami webowymi. Dokładniej mówiąc, dowiedziałeś się tego, co się dzieje, kiedy wpisujesz adres strony w pasek adresu przeglądarki: jak przeglądarka tłumaczy to na domenę, w jaki sposób domena jest kojarzona z adresem IP oraz jak żądanie HTTP jest wysyłane na serwer.

Dowiedziałeś się również, jak Twoja przeglądarka tworzy żądania i renderuje odpowiedzi oraz jak żądania HTTP pozwalają klientom na komunikację z serwerami. Dodatkowo dowiedziałeś się o tym, że podatności wynikają z podejmowania przez kogoś niezamierzonych akcji i otrzymywania informacji w zamyśle niedostępnych dla niego. Poznałeś także programy bug bounty, które wynagradzają hakerów za odkrycia podatności w zabezpieczeniach i zgłaszanie ich do właścicieli stron.

# 2

# OTWARTE PRZEKIEROWANIE

Naszą dyskusję rozpoczniemy podatnościami *otwartego przekierowania* następującego wtedy, kiedy wybrany cel odwiedza witrynę, która odsyła jego przeglądarkę do innego adresu URL, potencjalnie na zupełnie inną witrynę. Otwarte przekierowanie wykorzystuje zaufanie do danej domeny, aby zwabić ofiary na złośliwą stronę.

Atakom phishingowym może również towarzyszyć przekierowanie, aby nabrać użytkowników na to, że strona, na którą wysyłają informacje, jest zaufana, podczas gdy w rzeczywistości ich informacje trafiają na złośliwą stronę. W połączeniu z innymi atakami otwarte przekierowania mogą pozwolić atakującym na dystrybucję oprogramowania ze złośliwej strony lub kradzież tokenów Oauth (ten temat rozwiniemy w rozdziale 17).

Ponieważ otwarte przekierowania jedynie przenoszą użytkowników, są często uważane za mało niebezpieczne i mogą nie przysługiwać nagrody za ich znalezienie. Na przykład program bug bounty od Google traktuje otwarte przekierowania jako zbyt małe niebezpieczeństwo do wypłacenia

wynagrodzenia. The Open Web Application Security Project (OWASP), który stanowi społeczność skupioną wokół bezpieczeństwa aplikacji, prowadzi listę najbardziej krytycznych wad bezpieczeństwa. W edycji z 2017 roku swojej listy top dziesięciu podatności całkowicie usunęli z niej otwarte przekierowania.

Mimo że otwarte przekierowania są mało krytycznymi podatnościami, stanowią świetny materiał do nauki tego, w jaki sposób przeglądarka wykonuje przekierowania. W tym rozdziale nauczysz się, jak wykorzystać otwarte przekierowania i jak zidentyfikować kluczowe parametry, posiłkując się trzema przykładami raportów błędu.

## Jak działają otwarte przekierowania?

Otwarte przekierowania pojawiają się wtedy, gdy deweloper błędnie zaufa wejściu kontrolowanemu przez potencjalnego atakującego i przekierowuje użytkownika na inną stronę, zazwyczaj za pomocą parametrów URL, znaczników odświeżających HTML `<meta>` lub własności lokalizacji okna DOM.

Wiele stron celowo odsyła użytkowników na inne strony, umieszczając docelowy adres URL jako parametr w oryginalnym URL-u. Aplikacja wykorzystuje ten parametr, żeby powiedzieć wyszukiwarce, by wysłała żądanie `GET` do docelowego adresu URL. Na przykład załóżmy, że Google ma możliwość przekierowywania użytkowników do Gmaila przez odwiedzenie następującego adresu URL:

```
https://www.google.com/?redirect_to=https://www.gmail.com
```

W tym przypadku, gdy odwiedzisz ten link, Google otrzyma żądanie HTTP `GET` i użyje parametru `redirect_to`, by zdecydować, gdzie przekierować Twoją przeglądarkę. Po zrobieniu tego serwery Google zwracają odpowiedź HTTP z kodem statusu instruującym Twoją przeglądarkę do przekierowania użytkownika. Zazwyczaj jest to kod 302, ale w niektórych przypadkach może to być 301, 303, 307 lub 308. Wszystkie te kody HTTP mówią Twojej przeglądarce, że strona została znaleziona; jednakże kod również informuje przeglądarkę, by wykonała żądanie `GET` do adresu znalezionego w wartości parametru `redirect_to`, *https://www.google.com/*, który jest oznaczony w nagłówku odpowiedzi HTTP `Location`. Nagłówek `Location` określa, gdzie przekierować żądanie `GET`.

Teraz przypuśćmy, że atakujący zmienił oryginalny link na poniższy:

```
https://www.google.com/?redirect_to=https://www.attacker.com
```

Jeśli Google nie weryfikuje, czy parametr `redirect_to` należy do jednego z jego prawowitych serwerów, gdzie zazwyczaj odsyła użytkowników,

atakujący mógłby podmienić wartość parametru z jego własnym URL-em. W wyniku tego odpowiedź HTTP mogłaby instruować Twoją przeglądarkę do wysłania żądania `GET` na stronę *https://www.<attacker>.com/*. Kiedy atakujący wreszcie przyciągnął Cię na swoją złośliwą stronę, jest w stanie wykonać na Tobie kolejne ataki.

Gdy poszukujesz podatności tego rodzaju, miej na uwadze parametry URL włączające pewne nazwy, takie jak `url=`, `redirect=`, `next=` i tym podobne, mogące wskazywać na URL, do którego użytkownicy będą przekierowywani. Również pamiętaj o tym, że parametry nie będą zawsze oczywiście nazwane; parametry będą się różniły od siebie na różnych stronach, a nawet w obrębie jednej. W niektórych przypadkach parametry mogą być oznaczone tylko jednym znakiem, takim jak `r=` czy `u=`.

W dodatku do ataków bazujących na parametrach, tagi HTML `<meta>` i JavaScript mogą również przekierowywać przeglądarkę. Tagi `<meta>` HTML mówią przeglądarce, aby odświeżyć stronę i wykonać żądanie `GET` na adres URL zdefiniowany w atrybucie `content`. Oto jak może wyglądać jeden z nich:

```
<meta http-equiv="refresh" content="0; url=https://www.google.com/">
```

Atrybut `content` definiuje, jak przeglądarka wykona żądanie HTTP, na dwa sposoby. Po pierwsze, `content` określa, jak długo przeglądarka czeka przed wysłaniem żądania HTTP pod wskazany adres; w tym przypadku 0 sekund. Po drugie, atrybut `content` wyznacza parametr URL strony, do której wysyła żądanie `GET`; w tym przypadku `http://www.google.com`. Atakujący mogą wykorzystać te zachowania w sytuacjach, w których kontrolują atrybut `content` w tagu `<meta>` lub dodać własny tag przez inną podatność.

Atakujący może również użyć kodu JavaScript, by przekierować użytkowników przez modyfikację właściwości `location` okna przez *obiektowy model dokumentu* (*DOM – Document Object Model*). DOM jest interfejsem programowania dla dokumentów HTML i XML, który pozwala deweloperom na modyfikowanie struktury, stylów i zawartości strony internetowej. Ponieważ właściwość `location` wskazuje, gdzie żądanie powinno zostać przekierowane, przeglądarka niezwłocznie zinterpretuje szkodliwy JavaScript i przekieruje do określonego adresu URL. Atakujący może modyfikować własność `location` okna, używając jednego z wybranych skryptów:

```
window.location = https://www.google.com/
window.location.href = https://www.google.com
window.location.replace(https://www.google.com)
```

Zazwyczaj możliwość zmiany wartości `window.location` pojawia się jedynie tam, gdzie atakujący może wykonać kod JavaScript, przez podatność

cross-site scripting (XSS), albo tam, gdzie strona celowo pozwala użytkownikom decydować o adresie URL do przekierowania, podobnie jak w podatności serwisu HackerOne omawianej później w tym rozdziale, na stronie 16.

Kiedy szukasz podatności na otwarte przekierowanie, najczęściej będziesz monitorował swoją historię proxy, szukając żądań `GET` wysłanych do strony, którą testujesz.

## Otwarte przekierowanie przy instalacji motywu Shopify

**Poziom trudności:** Niski

**URL:** *https://apps.shopify.com/services/google/themes/preview/supply--blue?domain_name=<dowolnadomena>*

**Źródło:** *https://www.hackerone.com/reports/101962/*

**Data zgłoszenia:** 25 listopada 2015

**Nagroda:** 500 $

Pierwszy przykład podatności na otwarte przekierowania, który poznasz, został znaleziony na platformie Shopify pozwalającej użytkownikom tworzyć sklepy internetowe. Shopify zezwala administratorom na dostosowywanie wyglądu swoich sklepów przez zmianę ich motywów. Jako część tej funkcjonalności, Shopify zaoferowało możliwość sprawdzenia wyglądu motywu przez przekierowanie właściciela sklepu na adres URL. Przekierowujący URL prezentował się następująco:

```
https://apps.shopify.com/services/google/themes/preview/
supply--blue?domain_name=attacker.com
```

Parametr `domain_name` na końcu URL-a przekierowywał domenę sklepu użytkownika i dodawał `/admin` na końcu adresu. Shopify spodziewał się, że parametr `domain_name` zawsze będzie adresem sklepu użytkownika i nie weryfikował, czy należy do serwisu. W rezultacie atakujący mógł wykorzystać parametr do przekierowania ofiary na stronę *http:/<attacker>.com/admin/*, gdzie mógł podjąć się kolejnych ataków.

### Wnioski

Nie wszystkie podatności są skomplikowane. W przypadku tego przekierowania, wystarczyło podmienić parametr `domain_name` na zewnętrzną stronę, by przenieść użytkownika poza Shopify.

## Otwarte przekierowanie przy logowaniu do Shopify

**Poziom trudności:** Niski

**URL:** *https://mystore.myshopify.com/account/login/*

**Źródło:** *https://www.hackerone.com/reports/103772/*

**Data zgłoszenia:** 6 grudnia 2015

**Nagroda:** 500 $

Drugi przykład otwartego przekierowania jest podobny do pierwszego, z wyjątkiem tego, że zamiast parametru URL docelową domenę określa wartość parametru na końcu subdomeny Shopify. Normalnie ta funkcjonalność byłaby używana do odsyłania użytkownika na określoną stronę w obrębie jednego sklepu. Jednakże atakujący wciąż mogą manipulować tymi adresami, aby przekierować przeglądarkę z dala od subdomeny Shopify, na witrynę atakującego przez dodanie znaków, które zmienią znaczenie URL.

W przypadku tego błędu, po zalogowaniu do Shopify, serwis używał parametru `checkout_url` do przekierowania użytkownika. Dla zobrazowania powiedzmy, że ofiara odwiedza następujący adres:

```
http://mystore.myshopify.com/account/login?checkout_url=.attacker.com
```

Zostałaby przekierowana na URL *http://mystore.myshopify.com.<attacker>.com/*, który nie jest domeną Shopify.

Ponieważ URL kończy się *.<attacker>.com*, a wyszukiwarki DNS używają nazwy domeny wysuniętej najdalej na prawo w adresie, przekierowanie zostaje wykonane na domenę *<attacker>.com*. Zatem gdy *http://mystore.myshopify.com.<attacker>.com/* zostaje wysłany do wyszukiwarki DNS, zostanie dopasowany do *<attacker>.com*, którego Shopify nie ma, a nie *myshopify.com*, tak jak Shopify miał w zamiarze. Mimo że atakujący nie byłby w stanie wysłać ofiary na dowolną stronę, mógłby go wysłać na inną domenę przez dodanie specjalnych znaków, takich jak kropka, do wartości, którymi może manipulować.

### *Wnioski*

Jeśli tylko możesz kontrolować końcowy fragment adresu URL używanego przez stronę, dodanie specjalnych znaków URL może zmienić znaczenie danego adresu i przekierować użytkownika na inną domenę. Powiedzmy, że możesz kontrolować jedynie wartość parametru `checkout_url` oraz zauważysz, że parametr jest dodawany do gotowego adresu URL, takiego jak *http://mystore.myshopify.com/*. Spróbuj dodawać specjalne znaki URL, takie jak kropka lub @, w celu przetestowania możliwości kontroli lokalizacji przekierowania.

## Przekierowanie pośrednie na HackerOne

**Poziom trudności:** Niski
**URL:** N/A
**Źródło:** *https://www.hackerone.com/reports/111968/*
**Data zgłoszenia:** 20 stycznia 2016
**Nagroda:** 500 $

Niektóre strony internetowe próbują się chronić przed otwartymi przekierowaniami przez implementację *stron pośredniczących*, które wyświetlają się przed ukazaniem oczekiwanej zawartości. Za każdym razem, gdy przekierowujesz użytkownika na URL, możesz wyświetlić pośredniczącą stronę z wiadomością wyjaśniającą użytkownikowi to, że opuszcza domenę, na której się znajduje. W rezultacie, jeżeli przekierowująca strona pokazuje fałszywy login lub próbuje udawać zaufaną stronę, użytkownik będzie wiedział, że jest przekierowywany. To podejście stosuje serwis HackerOne w przypadku większości linków ze swojej strony; na przykład dla linków w zgłoszeniach błędów.

Możesz używać stron pośredniczących, by uniknąć podatności na przekierowania, jednakże komplikacje, które wynikają z tego, w jaki sposób strona wchodzi w interakcję z innymi, może doprowadzić do uszkodzenia linków. HackerOne używa Zendesk, serwisu obsługi klienta w systemie ticketowym, na subdomenie *https://support.hackerone.com/*. Poprzednio, jeśli po *hackerone.com* wpisałeś */zendesk_session*, przeglądarka przekierowywała Cię z platformy HackerOne na ich platformę Zendesk bez stron pośredniczących, ponieważ adresy URL zawierające domenę *hackerone.com* były zaufanymi linkami. (Obecnie HackerOne przekierowuje z *https://support.hackerone.com* na *docs.hackerone.com*, no chyba że przesyłasz wniosek o pomoc przez URL */hc/en-us/requests/new*.) Niestety jednak każdy mógł utworzyć swoje konto Zendesk i przekazać je do parametru `/redirect_to_account?state=`. Utworzone konto Zendesk mogło następnie przekierowywać na inną stronę nienależącą do Zendesk bądź HackerOne. Ponieważ Zendesk zezwalał na przekierowania między kontami bez pośredniczących stron, użytkownik mógł zostać wysłany na niezaufaną stronę bez ostrzeżenia. Jako rozwiązanie HackerOne zaczął identyfikować linki zawierające `zendesk_session` jako zewnętrzne, a tym samym wyświetlać ostrzeżenie po ich kliknięciu.

W celu potwierdzenia tej podatności, haker Mahmoud Jamal stworzył konto na platformie Zendesk z subdomeną *http://compayn.zendesk.com*. Następnie dodał następujący kod JavaScript do pliku nagłówka, używając edytora motywów Zendesk, który pozwalał administratorom na dostosowywanie wyglądu ich strony:

```
<script>document.location.href = <<http://evil.com>>;</script>
```

Używając tego kodu, Jamal instruował przeglądarkę, by odwiedziła *http://evil.com*. Tag `<script>` oznacza kod JavaScript, a `document` odnosi się do całego dokumentu HTML, który Zendesk zwraca jako informację dla strony internetowej. Kropki i nazwy następujące po `document` są jego właściwościami. Właściwości zawierają informację i wartości, które albo opisują obiekt, albo mogą być manipulowane, by zmienić obiekt. Możesz zatem użyć właściwości `location` do kontroli strony internetowej, którą Twoja przeglądarka wyświetla i użyć podwłaściwości `href` (która jest właściwością `location`) do przekierowania przeglądarki do zdefiniowanej strony. Odwiedzenie następującego linku przekierowywało ofiary do subdomeny Zendesk należącej do Jamala, która zawierała skrypt przekierowujący ich przeglądarki na *http://evil.com*:

```
https://hackerone.com/zendesk_session?locale_id=1&return_to=https:
//support.hackerone.com/ping/redirect_to_account?state=compayn:/
```

Ponieważ ten link zawiera domenę *hackerone.com*, pośrednicząca strona nie wyświetlała się, a użytkownik nie wiedział, że strona, którą odwiedza, jest niebezpieczna. Co ciekawe, Jamal wcześniej zgłosił problem z brakującą stroną pośredniczącą do Zendeska, lecz zgłoszenie zostało zignorowane. Instynktownie nie zaprzestał poszukiwań i w końcu odkrył atak przez JavaScript, który przekonał HackerOne do wypłacenia nagrody.

### *Wnioski*

Gdy poszukujesz podatności, miej na uwadze fakt, iż zewnętrzne serwisy, których używają strony, otwierają nowe miejsca na poszukiwania. Podatność na stronie HackerOne istniała tylko dzięki ich współpracy z Zendesk-iem.

Dodatkowo, gdy znajdziesz błąd, mogą zdarzyć się przypadki, gdzie Twoje uwagi dotyczące bezpieczeństwa nie będą zrozumiane przez osobę reagującą na Twoje zgłoszenie. Z tego powodu przedyskutujemy zgłoszenia znalezionych błędów w rozdziale 19, który uszczegóławia to, co powinieneś dołączyć do raportu, jak budować współpracę z firmami i wszelkie pozostałe informacje. Jeśli popracujesz nad opisem znalezionej podatności w raporcie, Twoje starania pomogą w płynnym rozwiązaniu.

Oprócz tego pojawią się sytuacje, gdy firmy nie zgodzą się z Tobą. W takich przypadkach kontynuuj poszukiwania, tak jak to zrobił Jamal, i sprawdź, czy możesz udowodnić niebezpieczeństwo, łącząc kilka podatności naraz, by zademonstrować ich wpływ.

## Podsumowanie

Otwarte przekierowania pozwalają atakującym na odesłanie użytkowników na niebezpieczną stronę bez ich wiedzy. Znajdowanie ich – co już powinieneś wiedzieć – wymaga wnikliwej obserwacji. Parametry do przekierowań

mogą być łatwe do znalezienia, jeśli mają nazwy takie jak `redirect_to=`, `domain_name=` czy `checkout_url=`. W pozostałych jednak przypadkach będą to mniej oczywiste nazwy: `r=`, `u=` i tak dalej.

Podatność w postaci otwartego przekierowania opiera się na wykorzystaniu zaufania ofiary do serwisu, który zna, podczas gdy jest nabierana na odwiedzenie strony atakującego. Kiedy zauważysz potencjalnie podatne parametry, upewnij się, że przetestowałeś je dokładnie i użyłeś w tym celu znaków specjalnych.

Przekierowanie pośrednie HackerOne'a pokazuje istotę rozpoznawania narzędzi i serwisów, których strony używają, podczas poszukiwań podatności. Pamiętaj, że czasami będziesz musiał być cierpliwy i wyczerpująco opisać zarówno podatność, jak i jej wpływ na testowane strony, by namówić firmę na zaakceptowanie Twojego znaleziska i przyznanie nagrody.

# 3

# HTTP PARAMETER POLLUTION

*HTTP Parameter Pollution* (*HPP*) jest procesem manipulacji tego, w jaki sposób witryna traktuje parametry, które dostaje podczas żądań HTTP. Ta podatność następuje, gdy atakujący wstrzykuje dodatkowe parametry do żądania, a docelowa strona akceptuje je, prowadząc tym samym do nieoczekiwanych wyników. Błędy HPP mogą wystąpić po stronie serwera lub klienta. Jeśli błąd występuje po stronie klienta, którą jest zazwyczaj Twoja przeglądarka, możesz zobaczyć efekt swoich testów. W wielu przypadkach podatności HPP zależą od tego, w jaki sposób kod po stronie serwera używa wartości przekazanych jako parametry, które są kontrolowane przez atakującego. Z tego powodu szukanie tego rodzaju podatności może wymagać nieco więcej pracy niż w przypadku pozostałych.

W tym rozdziale zaczniemy od poznania ogólnych różnic między HPP po stronie serwera a HPP po stronie klienta. Potem przedstawię trzy przykłady oparte na znanych serwisach społecznościowych, aby zilustrować użycie

HPP do wstrzyknięcia parametrów na docelowe strony. Dokładniej mówiąc, dowiesz się, jak wykonywać testy na HPP oraz poznasz miejsca, w których deweloperzy często popełniają pomyłki. Jak zobaczysz, znajdowanie podatności HPP wymaga cierpliwości i kombinowania, ale może być warte starań.

## HPP po stronie serwera

W przypadku HPP po stronie serwera, do serwera wysyła się nieoczekiwane informacje w zamiarze otrzymania od niego nieoczekiwanych rezultatów. Kiedy wykonujesz żądanie do strony internetowej, serwer tej strony przetwarza żądanie i zwraca odpowiedź, tak jak omawialiśmy to w rozdziale 1. W niektórych przypadkach serwery nie zwracają zwyczajnie strony internetowej, ale również uruchamiają kod, bazując na informacji, którą otrzymali z odebranego URL-a. Kod ten jest uruchamiany tylko na serwerze, jest więc niewidoczny dla ciebie; możesz zobaczyć wysłaną informację i zwrócony rezultat, lecz pośredniczący kod jest nieosiągalny. W związku z tym możesz jedynie wywnioskować to, co się dzieje. Ponieważ nie widzisz tego, jak kod serwera funkcjonuje, HPP po stronie serwera zależy od Twojej zdolności do identyfikacji potencjalnie podatnych parametrów i eksperymentowania z nimi.

Spójrzmy na przykład: HPP po stronie serwera mógłby wystąpić wtedy, gdy Twój bank inicjowałby przelew za pomocą swojej strony przez akceptację parametrów URL, które są przetwarzane na jego serwerach. Wyobraź sobie, że mógłbyś zrobić przelew przez wpisanie trzech wartości w trzech parametrach URL: `from`, `to` i `amount`. Parametry te określają kolejno numer konta, z którego ma zostać wykonany przelew, numer konta docelowego i kwotę pieniędzy do przesłania. Adres URL z tymi parametrami, który przesyła 5000 $ z konta o numerze 12345 na konto 67890, wyglądałby następująco:

```
https://www.bank.com/transfer?from=12345&to=67890&amount=5000
```

Możliwe jest, że bank zakładałby, że otrzyma tylko jeden parametr `from`. Co jednak jeśli otrzyma dwa, tak jak tutaj:

```
https://www.bank.com/transfer?from=12345&to=67890&amount=5000&from=ABCDEF
```

Ten URL jest umyślnie zbudowany tak samo jak poprzedni z tą różnicą, że zawiera dodatkowy parametr `from`, który określa inne konto nadawcy, `ABCDEF`. W tej sytuacji atakujący mógłby wysłać dodatkowy parametr w nadziei, że aplikacja zweryfikuje transfer, używając pierwszego parametru `from`, ale pobierze pieniądze, używając drugiego. Zatem atakujący byłby w stanie wykonać transfer pieniędzy z konta, do którego nie ma dostępu, jeżeli tylko bank akceptowałby ostatni parametr `from`, który otrzymał.

Zamiast wysyłać 5000 $ z konta 12345 na 67890, kod serwera użyłby drugiego parametru i wysłał pieniądze z konta ABCDEF na 67890.

Kiedy serwer dostaje kilkukrotne parametry z tą samą nazwą, może odpowiedzieć na wiele sposobów. Na przykład PHP i Apache używają ostatniego wystąpienia, Apache Tomcat używa pierwszego, ASP i IIS używają wszystkich i tak dalej. Dwóch badaczy, Luca Carettoni i Stefano di Paola, udostępnili dokładną prezentację wielu różnic między technologiami serwerów podczas konferencji AppSec EU 09; informacja ta jest teraz dostępna na stronie OWASP na *https://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf* (zob. slajd 9). W wyniku tego nie ma jednego gwarantowanego procesu do obsługi wielokrotnych parametrów z tą samą nazwą, więc znalezienie podatności HPP wymaga nieco kombinowania, by przekonać się o tym, jak działa strona, którą sprawdzasz.

Przykład z bankiem używa parametrów, które są oczywiste. Czasami jednak podatności HPP objawiają się jako rezultat ukrytego zachowania po stronie serwera, wynikającego z kodu, który nie jest bezpośrednio widoczny. Powiedzmy na przykład, że Twój bank zadecydował ulepszyć sposób, w jaki przetwarza przelewy, i zmienia kod w backendzie tak, aby nie używał parametru `from` z URL-a. Tym razem bank użyje dwóch parametrów, jednego do konta, na które ma zostać wykonany przelew, a drugiego do kwoty do przelania. Przykładowy link mógłby wyglądać tak:

```
https://www.bank.com/transfer?to=67890&amount=5000
```

Zazwyczaj kod na serwerze byłby dla nas zagadką, lecz ze względu na potrzeby tego przykładu wiemy, że kod Ruby serwera (nadzwyczaj brzydki i bezużyteczny) wygląda następująco:

```
user.account = 12345
def prepare_transfer(❶params)
 ❷ params << user.account
 ❸ transfer_money(params) #user.account (12345) becomes params[2]
end
def transfer_money(params)
 ❹ to = params[0]
 ❺ amount = params[1]
 ❻ from = params[2]
   transfer(to,amount,from)
end
```

Ten kod tworzy dwie funkcje, `prepare_transfer` i `transfer_money`. Funkcja `prepare_transfer` używa tablicy o nazwie `params` ❶, która zawiera parametry `to` i `amount` pochodzące z URL-a. Tablicę stanowiłoby `[67890,5000]`, gdzie jej wartości są wciśnięte między nawiasy, a każda

wartość jest oddzielona przecinkiem. Pierwsza linia funkcji ❷ dodaje informacje o koncie użytkownika, które zostały zdefiniowane wcześniej w kodzie, na końcu tablicy. Dostajemy zatem tablicę `params [67890,5000,12345]`, która następnie jest przekazana do `transfer_money` ❸. Zauważ to, że w odróżnieniu od parametrów tablice nie mają nazw powiązanych z ich wartościami, więc kod podlega tablicy zawierającej zawsze każdą wartość w następującej kolejności: konto odbiorcy jako pierwsze, kwota przelewu druga i konto, z którego wychodzi przelew, jako następne. W `transfer_money` kolejność wartości staje się oczywista z racji, że funkcja przypisuje każdą wartość tablicy do zmiennej. Ze względu na to, że tablice są numerowane od 0, `params[0]` przyjmuje pierwszą wartość w tablicy, którą w tym przypadku jest `67890` i przypisuje ją do zmiennej `to` ❹. Pozostałe wartości są również przypisywane zmiennym w wierszach ❺ i ❻. Wtedy nazwy zmiennych są przekazane do funkcji `transfer`, niepokazanej w tym fragmencie, która przyjmuje wartości i wykonuje przelew.

W najlepszej sytuacji parametry URL byłyby zawsze formatowane w sposób, jakiego kod oczekuje. Jednakże atakujący mógłby zmienić wynik tego procesu przez podanie w `from` wartości dla `params`, tak jak z następującym URL-em:

```
https://www.bank.com/transfer?to=67890&amount=5000&from=ABCDEF
```

W tej sytuacji parametr `from` jest również włączony do tablicy `params` przekazanej funkcji `prepare_transfer`; co za tym idzie, wartościami tablicy mogłyby być `[67890,5000,ABCDEF]`, a podanie konta użytkownika w ❷ dawałoby rezultat w postaci `[67890,5000,ABCDEF,12345]`. W wyniku tego w funkcji `transfer_money` wywołanej w `prepare_tranfer` wartość `from` stałaby się trzecim parametrem, zamieniając oczekiwaną wartość `12345` na wartość atakującego `ABCDEF` ❹.

## HPP po stronie klienta

Podatności HPP po stronie klienta pozwalają atakującym na wstrzyknięcie dodatkowych parametrów do adresu URL, w celu wywołania efektów po stronie użytkownika (*strona klienta* jest częstym sposobem odnoszenia się do akcji, które dzieją się na twoim komputerze, często przez przeglądarkę, a nie po stronie serwera).

Luca Carettoni i Stefano di Paola w swojej prezentacji zamieścili przykład tego zachowania, używając wymyślonego adresu URL *http://host/page.php?par=123%26action=edit* i następującego kodu na serwerze:

```
❶ <? $val=htmlspecialchars($_GET['par'],ENT_QUOTES); ?>
❷ <a href="/page.php?action=view&par='.<?=$val?>.'">View Me!</a>
```

Ten kod generuje nowy adres URL, bazując na wartości `par` parametru wpisanego przez użytkownika. W tym przykładzie atakujący przekazuje wartość `123%26action=edit` jako wartość dla `par` w celu wygenerowania dodatkowego, nieoczekiwanego parametru. Wartością dla `&` zakodowaną przez URL jest `%26`, co oznacza, że kiedy URL jest przetwarzany, `%26` jest interpretowane jako `&`. Ta wartość dodaje dodatkowy parametr do wygenerowanego `href`-a, bez sprecyzowania parametru w URL-u. Używając parametru `123&action=edit` zamiast `%26`, `&` byłoby zinterpretowane jako oddzielenie dwóch parametrów, lecz ze względu na to, że strona używa tylko jednego parametru `par` w swoim kodzie, parametr `action` zostałby usunięty. Wartość `%26` omija ten problem, zapewniając, że akcja nie jest rozpoznawana jako osobny parametr, a więc `123%26action=edit` staje się wartością `par`.

Następnie `par` (z zakodowanym `&` jako `%26`) jest przekazany do funkcji `htmlspecialchars` ❶. Funkcja `htmlspecialchars` konwertuje znaki specjalne, takie jak `%26`, do ich zakodowanych w HTML-u wartości, zamieniając `%26` na `&` (jednostka w HTML-u, która reprezentuje `&`), gdzie ten znak może mieć specjalne znaczenie. Zmieniona wartość jest następnie przechowywana w `$val`. W następnej kolejności generowany jest nowy link przez dodanie `$val` do wartości `href` w ❷. Zatem wygenerowany link to `<a href="/page.php?action=view&par=123&action=edit">`. W konsekwencji atakujący zdołał dodać nieoczekiwany `action=edit` do URL-a `href`, który mógłby doprowadzić do podatności w zależności od tego, jak aplikacja traktuje przemycony parametr `action`.

Następne 3 przykłady wyjaśniają szczegółowo oba błędy HPP, zarówno po stronie serwera, jak i klienta, znalezione na HackerOne i Twitterze. Wszystkie z tych przykładów wymagają manipulowania parametrami URL. Miej na uwadze fakt, że żaden z przykładów nie używa tej samej metody lub nie współdzieli głównej przyczyny istnienia błędu, podkreślając istotę dokładnego testowania w poszukiwaniach podatności HPP.

## Przyciski do udostępniania na HackerOne

**Poziom trudności:** Niski
**URL:** *https://hackerone.com/blog/introducing-signal-and-impact/*
**Źródło:** *https://hackerone.com/reports/105953/*
**Data zgłoszenia:** 18 grudnia 2015
**Nagroda:** 500 $

Jedną z metod na znajdowanie podatności HPP jest szukanie linków, które łączą się z zewnętrznymi serwisami. Blog na HackerOne robi to, dodając linki do udostępniania zawartości w popularnych serwisach społecznościowych, takich jak Twitter lub Facebook. Po kliknięciu linki te generują gotową treść dla użytkownika do publikacji na swoim profilu. Opublikowana treść zawiera odnośnik URL do oryginalnego posta.

Pewien haker odkrył podatność, która pozwoliła mu na dołączenie parametru do adresu URL z postem na blogu HackerOne. Dodany parametr był umieszczany w udostępnionym linku, przez co treść mogła odsyłać w inne miejsce niż zamierzony post na blogu.

Przykład użyty w tym zgłoszeniu podatności korzystał z URL-a *https://hackerone.com/blog/introducing-signal*, a następnie na jego końcu wystarczyło dodać *&u=https://vk.com/durov*. Na stronie bloga, kiedy HackerOne renderował link do udostępnienia na Facebooku, link prezentował się następująco:

```
https://www.facebook.com/sharer.php?u=https://hackerone.com/blog/
introducing-signal?&u=https://vk.com/durov
```

Jeśli użytkownicy HackerOne kliknęli zainfekowany link, chcąc udostępnić post, ostatni parametr `u` miał pierwszeństwo nad pierwszym. Co za tym idzie, post na Facebooku użyłby ostatniego parametru `u`. Wtedy użytkownicy Facebooka, którzy kliknęli w link, zostaliby odesłani na *https://vk.com/durov* zamiast na HackerOne.

Co więcej, w przypadku postów na Twitterze, HackerOne dołącza domyślny tekst do tweeta, który promuje post. Atakujący mógł również manipulować tym tekstem, dołączając do adresu `&text=`, tak jak tutaj:

```
https://hackerone.com/blog/introducing-signal?&u=https://vk.com/
durov&text=another_site:https://vk.com/durov
```

Kiedy użytkownik kliknął w poprzedzający link, wyświetliło się przed nim okno pop-up z tweetem zawierającym tekst "another_site: https://vk.com/durov", zamiast tekstu promującego blog HackerOne.

### *Wnioski*

Wypatruj podatności w sytuacjach, gdy strona akceptuje zawartość, łączy się z innym serwisem (takim jak media społecznościowe) i bazuje na obecnym adresie URL w celu wygenerowania zawartości do publikacji.

W takich przypadkach możliwe jest, że przesyłana zawartość nie podlega żadnym kontrolom bezpieczeństwa, co może prowadzić do podatności HPP.

## Anulowanie subskrypcji powiadomień na Twitterze

**Poziom trudności:** Niski

**URL:** *https://www.twitter.com/*

**Źródło:** *https://blog.mert.ninja/twitter-hpp-vulnerability/*

**Data zgłoszenia:** 23 sierpnia 2015

**Nagroda:** 700 $

W niektórych przypadkach szukanie podatności HPP z powodzeniem wymaga niemałej wytrwałości. W sierpniu 2015 roku haker Mert Tasci, podczas anulowania subskrypcji powiadomień z Twittera, zauważył interesujący URL (który na potrzeby książki został skrócony):

```
https://twitter.com/i/u?iid=F6542&uid=1134885524&nid=22+26&
sig=647192e86e28fb6691db2502c5ef6cf3xxx
```

Zwróć uwagę na parametr `UID`. Okazuje się, że jest on numerem ID użytkownika, który jest obecnie zalogowany. Po dostrzeżeniu `UID` Tasci zrobił to, co na jego miejscu zrobiłaby większość hakerów – spróbował zmienić obecny `UID` na inny, należący do innego użytkownika, jednak nic się nie stało. Twitter zwrócił błąd.

Podczas gdy inni w tym miejscu poddaliby się, zdeterminowany Tasci kontynuował; spróbował on dodać drugi parametr `UID` w taki sposób, że adres URL wyglądał następująco (ponownie, skrócona wersja):

```
https://twitter.com/i/u?iid=F6542&uid=2321301342&uid=1134885524&nid=
22+26&sig=647192e86e28fb6691db2502c5ef6cf3xxx
```

Sukces! Udało mu się anulować subskrypcję powiadomień e-mail u innego użytkownika. Twitter był podatny na HPP anulowania subskrypcji użytkowników. Powód, dla którego warto zapamiętać tą podatność, wyjaśnił mi FileDescriptor. Chodzi o parametr `SIG`. Jak się okazuje, Twitter generuje wartość `SIG` na podstawie wartości `UID`. Kiedy użytkownik klika w link do anulowania subskrypcji, Twitter weryfikuje, czy link nie został zmodyfikowany, sprawdzając wartości `SIG` i `UID`. W przypadku pierwszej próby Tasciego zmiana `UID` w celu anulowania subskrypcji u innego użytkownika zawiodła, ponieważ podpis nie pasował do tego, którego Twitter oczekiwał. Jednakże, dodając drugi `UID`, Tasciemu udało się oszukać Twittera, który sprawdzał podpis, używając pierwszego parametru `UID`, a wykonywał właściwą akcję przy użyciu tego drugiego.

### *Wnioski*

Wysiłek Tasciego pokazuje, jak ważna jest wytrwałość i wiedza. Jeżeli odpuściłby po początkowej próbie zmiany `UID` na inną wartość lub nie wiedziałby o podatnościach typu HPP, nie otrzymałby nagrody 700 $.

Ty również miej na oku parametry z liczbami, które są automatycznie zwiększane, takimi jak `UID`, które znajdują się w żądaniach HTTP: wiele

podatności wymaga manipulowania wartościami parametrów takimi jak te, aby spowodować w aplikacji nieoczekiwane zachowania. Przedyskutujemy ten temat dokładnie w rozdziale 16.

## Web Intents Twittera

**Poziom trudności:** Niski

**URL:** *https://www.twitter.com/*

**Źródło:** *https://ericrafaloff.com/parameter-tampering-attack-on-twitter-web-intents/*

**Data zgłoszenia:** Listopad 2015

**Nagroda:** Nieujawniona

W niektórych przypadkach podatność HPP może sygnalizować inne problemy i doprowadzić do znalezienia dodatkowych błędów. Tak właśnie się stało z funkcją Twittera – Web Intents. Dostarcza ona okna pop-up do interakcji z tweetami użytkownika, odpowiedziami, retweetami, polubieniami i pozwala robić to z poziomu innych stron niż Twitter. Web Intent uwalnia użytkowników od potrzeby uwierzytelniania nowej aplikacji w celach interakcji na Twitterze. Rysunek 3.1 pokazuje, jak jedno z tych okien wygląda.


*Rysunek 3.1. Wczesna wersja funkcjonalności Web Intents, która pozwala użytkownikom na interakcję z zawartością Twittera bez opuszczania strony. W tym przykładzie użytkownicy mogą polubić tweeta Jacka*

Testując tą funkcję, haker Eric Rafaloff odkrył, że wszystkie cztery rodzaje interakcji – obserwowanie użytkownika, polubienie tweeta, retweetowanie i tweetowanie – były podatne na HPP. Twitter przygotowywał okna za pomocą żądania `GET` z parametrami URL, takimi jak w przykładzie:

```
https://twitter.com/intent/intentType?parameter_name=parameterValue
```

Ten adres zawierał *intentType* i jedną lub więcej dodatkowych par parametrów i ich wartości – na przykład nazwę użytkownika i ID Tweeta. Twitter używał tych parametrów do utworzenia okna pop-up z możliwością polubienia tweeta bądź obserwowania użytkownika. Rafaloff odkrył problem, kiedy utworzył URL z dwoma parametrami `screen_name` zamiast domyślnie jednego, tak jak tutaj:

```
https://twitter.com/intent/follow?screen_name=twitter&screen_name=ericrtest3
```

Podczas generowania przycisku Follow, Twitter wykonałby żądanie, korzystając z drugiej wartości parametru `screen_name` – `ericrtest3` – zamiast pierwszej. Co za tym idzie, użytkownik próbujący zaobserwować oficjalny profil Twittera tak naprawdę obserwował testowe konto Rafaloffa. Odwiedzając URL, który przygotował Rafaloff, kod na serwerze Twittera generował poniższy formularz HTML, używając obu parametrów `screen_name`:

```
❶ <form class="follow" id="follow_btn_form" action="/intent/
follow?screen_name=ericrtest3" method="post">
    <input type="hidden" name="authenticity_token" value="...">
❷ <input type="hidden" name="screen_name" value="twitter">
❸ <input type="hidden" name="profile_id" value="783214">
    <button class="button" type="submit">
      <b></b><strong>Follow</strong>
    </button>
</form>
```

Twitter używał informacji z pierwszego parametru `screen_name`, która dotyczy oficjalnego konta Twittera. W rezultacie użytkownik widział poprawny profil użytkownika, który zamierzał zaobserwować, ponieważ w ❷ i ❸ został użyty pierwszy parametr `screen_name`. Jednakże po kliknięciu przycisku ofiara zaobserwowałaby konto `ericrtest3`, ponieważ atrybut `action` w tagu formularza używał drugiej wartości `screen_name` ❶ przekazanej do URL-a.

W przypadku pozostałych interakcji, takich jak polubienie, Rafaloff zauważył, że wciąż może dodać parametr `screen_name` z podobnymi skutkami. Rozważmy jako przykład taki URL:

```
https://twitter.com/intent/like?tweet_i.d=661625230297821184 5
&screen _name=ericrtest3
```

Domyślnie okno potrzebowałoby tylko parametru `twitter_id`; Rafaloff jednakże umieścił na końcu URL-a parametr `screen_name`. Próbując

polubić tego tweeta, ofiara zobaczyłaby prawidłowy profil użytkownika, lecz znajdujący się obok przycisk Follow dotyczyłby niezwiązanego z postem użytkownika – `ericrtest3`.

### *Wnioski*

Podatność w funkcji Web Intents Twittera jest podobna do poprzedniej podatności dotyczącej `UID`. Nic dziwnego zatem, że jeśli strona ma wady, takie jak HPP, to może być to oznaką istnienia szerszego problemu w systemie. Czasami, kiedy znajdziesz podobną podatność, warto jest poświęcić czas na zbadanie całej platformy, by spróbować wykorzystać wadę w miejscach opartych na podobnych mechanizmach.

## Podsumowanie

Ryzyko stwarzane przez HPP jest zależne od akcji, które podejmuje backend aplikacji oraz od zainfekowanych parametrów, które zostały użyte.

Odnalezienie podatności HPP wymaga bardziej dokładnego testowania niż w przypadku pozostałych podatności z tego względu, że nie mamy dostępu do kodu, na którym serwer operuje po otrzymaniu żądania HTTP. Oznacza to, że możemy tylko domyślać się, jak serwer używa parametrów, które od nas otrzymał.

Metodą prób i błędów jesteś w stanie odkryć, w jakich sytuacjach pojawiają się podatności HPP. Często linki do mediów społecznościowych są dobrym miejscem do rozpoczęcia testów, pamiętaj jednak, by nie przestawać próbować i myśleć o HPP podczas testowania parametrów takich jak ID.