Wappalyzer bądź BuiltWith, lub przeglądając źródło strony i szukając atrybutów `ng-`. Jak już wspomniałem, starsze wersje AngularJS implementowały Sandboxa, lecz wersja, której używał Uber, była podatna na ucieczkę. W tym przypadku zatem podatność CSTI oznaczała możliwość wykonania XSS.

Używając poniższego JavaScriptu w adresie URL, Kettle dokonał ucieczki z Sandboxa AngularJS i uruchomił funkcję `alert`:

```
https://developer.uber.com/docs/deep-linking?q=wrtz{{(_="".sub).call.
call({}[$="constructor"].getOwnPropertyDescriptor(_.__proto__,$).
value,0,"alert❶")()}}zzzz
```

Dekonstrukcja tego payloadu wychodzi poza zakres tej książki, biorąc pod uwagę publikacje licznych obejść Sandboxa AngularJS oraz usunięcia ich w wersji 1.6. W każdym bądź razie końcowym rezultatem payloadu `alert❶` jest okno pop-up. Ten dowód pokazał Uberowi, że atakujący mógł wykorzystać tą podatność CSTI do wykonania XSS-a, co mogło skutkować zagrożeniem kont deweloperów i powiązanych aplikacji.

Wnioski

Po ustaleniu, czy witryna korzysta z template engine'a po stronie klienta, rozpocznij testy od przesłania prostych ładunków, korzystając ze standardowej składni dla danego engine'a, takiej jak `{{7*7}}` dla AngularJS, i sprawdzania renderowanych rezultatów. Jeśli ładunek został wykonany, sprawdź wersję AngularJS, której strona używa, wpisując *Angular.version* w przeglądarkowej konsoli dla deweloperów. Jeśli wersja jest wyższa niż 1.6, możesz przesłać payload z wyżej wymienionych źródeł bez omijania Sandboxa. Jeśli jest niższa niż 1.6, będziesz musiał znaleźć obejście dostosowane do wersji AngularJS, której aplikacja używa.

Template Injection w Uberze przez Flask i Jinja2

Poziom trudności: Średni

URL: *https://riders.uber.com/*

Źródło: *https://hackerone.com/reports/125980/*

Data zgłoszenia: 25 marca 2016

Nagroda: 10 000 $

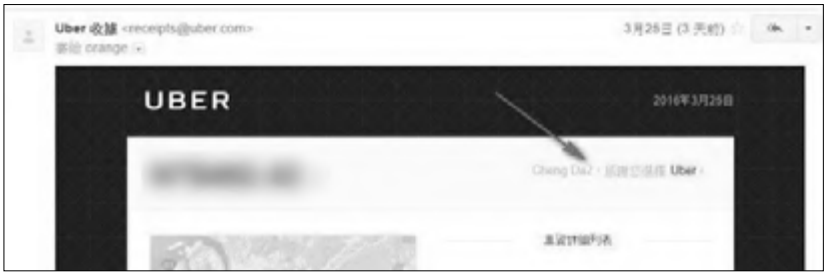
Podczas hakowania ważna jest identyfikacja technologii używanej przez daną firmę. Kiedy Uber uruchomił swój publiczny program bug bounty na portalu HackerOne, dołączył do niego „mapę skarbów" pod adresem *https://eng.uber.com/bug-bounty/* (ulepszona wersja mapy została opublikowana w sierpniu 2017 roku na stronie *https://medium.com/uber-security-privacy/*

uber-bug-bounty-treasure-map-17192af85c1a/). Mapa zidentyfikowała szereg wrażliwych funkcjonalności obsługiwanych przez Ubera, dołączając do tego oprogramowanie, którego używała każda z nich.

Na tej mapie Uber ujawnił, że witryna *riders.uber.com* została zbudowana przy użyciu Node.js Express i Backbone.js, z których żadne nie wyróżnia się jako potencjalne zagrożenie atakiem SSTI. Jednak strony *vault.uber.com* i *partners.uber.com* zostały napisane przy użyciu Flask oraz Jinja2. Jinja2 to template engine, który w przypadku niepoprawnej konfiguracji pozwala na zdalne wykonanie kodu. Mimo że witryna *riders.uber.com* nie używała Jinja2, jeśli dostarczała dane wejściowe do subdomeny *valut* bądź *partners*, a te strony ufały wejściu, które otrzymują, atakujący mógł być w stanie wykonać atak SSTI.

Orange Tsai, haker, który odkrył tę podatność, w celu rozpoczęcia testów na podatność SSTI wpisał `{{1+1}}` jako swoją nazwę. Szukał jakiejkolwiek interakcji, która odbywała się między subdomenami.

W swoim artykule Orange wyjaśnia, że jakakolwiek zmiana w profilu na *riders.uber.com* skutkowała wiadomością e-mail z powiadomieniem dla właściciela profilu o zmianach (jest to częsta praktyka bezpieczeństwa). Zmieniając swoją nazwę na `{{1+1}}`, otrzymał wiadomość z `2` w swojej nazwie, tak jak pokazuje to rysunek 8.1.



Rysunek 8.1. E-mail, który otrzymał Orange, wykonywał kod wstrzyknięty do nazwy profilu

Widząc to zachowanie, hakerowi od razu zaświeciła się czerwona lampka, ponieważ Uber obliczył jego wyrażenie i umieścił wynik jako nazwę profilu. Orange następnie próbował przesłać kod w Pythonie `{% for c in [1,2,3]%} {{c,c,c}} {% endfor %}` w celu sprawdzenia działania odkrytej podatności na bardziej złożonych operacjach. Ten kod iteruje przez tablicę `[1,2,3]` i wypisuje każdy numer trzy razy. E-mail na rysunku 8.2 pokazuje nazwę Orange'a wyświetlaną w postaci dziewięciu cyfr, które są rezultatem wykonania pętli `for`, potwierdzając tym samym jego odkrycie.

Jinja2 również implementuje Sandboxa, co ogranicza możliwość wykonywania dostarczonego kodu, lecz w niektórych sytuacjach można go ominąć. W tym przypadku Orange był w stanie to zrobić od razu.