

BEZPIECZEŃSTWO APLIKACJI

- CSP 2.0
- Bit-Flipping
- Padding Oracle
- Mechanizm Service Workers



PODATNOŚCI W MECHANIZMACH UPLOADU

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych



MICHał SAJDak

SEKURAK/OFFLINE: WITAMY, PONOWNIE

Ponad pół roku od wydania pierwszego **sekurak-zina**, który pobrało około 50 000 osób – udostępniamy kolejny numer.

Tematyka pozostaje bez zmian: cały czas poruszamy się w obszarze aplikacji webowych. Tym razem przygotowaliśmy kilka bardziej zaawansowanych tekstów podzielonych na dwa obszary (ofensywa/defensywa). W artykułach znajdziecie wiedzę przydatną dla programistów, testerów, pentesterów, administratorów oraz wszystkich fanów problematyki bezpieczeństwa webowego.

Dla żądnego praktyki: w ciągu ostatnich kilku miesięcy w serwisie [rozwal.to](#), umieściliśmy rozbudowane i zaawansowane zadania – m.in. [serię zadań sylwestrowych](#).

Sekurak/Offline udostępniamy bezpłatnie i zachęcamy do dzielenia się nim ze znajomymi. Do jego pobrania nie jest wymagany adres e-mail, ale [możecie zapisać się](#), aby otrzymywać powiadomienia o kolejnych numerach.

Macie pytania? Prośby? Chcielibyście podzielić się uwagami? Czekamy na kontakt od Was (sekurak@sekurak.pl).

REDAKTOR NACZELNY

Michał Sajdak

WSPÓŁPRACA/TEKSTY

Michał Bentkowski

Marcin Bury

Rafał Janicki

Bartosz Jerzman

Adrian Michalczyk

REDAKCJA JĘZYKOWA

Julia Wilk

KOREKTA

Katarzyna Sajdak

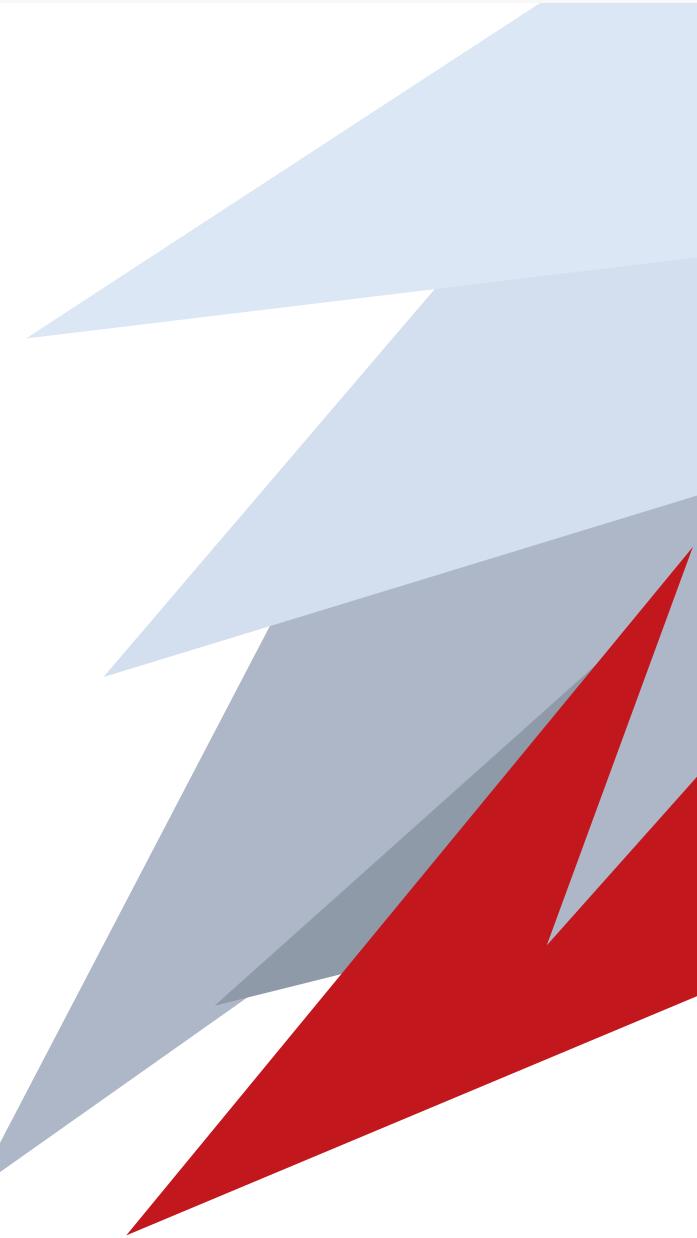
SKŁAD

Havok

Treści zamieszone w Sekurak/Offline służą wyłącznie celom informacyjnym oraz edukacyjnym. Nie ponosimy odpowiedzialności za ewentualne niezgodne z prawem wykorzystanie materiałów dostępnych w zinie oraz ewentualne szkody czy inne straty poniesione w wyniku wykorzystania tych materiałów. Stanowczo odradzamy działanie niezgodne z prawem czy dobrymi obyczajami.

Wszelkie prawa zastrzeżone. Kopiowanie dozwolone (a nawet wskazane) – tylko w formie niezmienionej i w całości.

Spis treści



Mechanizm Service Workers

Mechanizm Service Workers został wprowadzony w najnowszych wersjach przeglądarek internetowych, by rozwiązać problem, z którym świat aplikacji webowych boryka się od dawna – mianowicie: jak aplikacja powinna się zachowywać w przypadku utraty połączenia z Internetem. Niniejszy artykuł ma na celu przybliżenie zasad działania Service Workers oraz wskazanie największych zagrożeń związanych z tym mechanizmem, którymi w szczególności powinni być zainteresowani twórcy stron internetowych.

Service Workers nie jest pierwszym mechanizmem próbującym rozwiązać problem z dostępem offline do aplikacji webowych. Kilka lat temu zdefiniowano mechanizm AppCache, który pozwala developerom zdefiniować listę plików podlegających cache'owaniu. Aby skorzystać z tego rozwiązania, należało w elemencie `html` dodać atrybut `manifest` wskazujący na plik manifestu.

Wyglądało to następująco:

```
<!doctype html>
<html manifest="manifest.appcache">
...
</html>
```

Natomiast sam plik `manifest.appcache` mógł na przykład zawierać następującą treść:

```
CACHE MANIFEST
/styl.css
/main.js
/obrazek.png
```

W tym przypadku, plik manifestu instruuje przeglądarkę, że cache'owaniu powinny podlegać trzy konkretne pliki.

Praktyka pokazała jednak, że rozwiązanie AppCache jest mało elastyczne; sposób definiowania pliku manifestu jest częściowo niezgodny z „duchem Web”, nie wspominając o niektórych dziwnych zachowaniach implementacji, które sprawiały, że AppCache nie spełniał dobrze swojej roli w aplikacjach, będących czymś więcej niż pojedynczą stroną korzystającą z szeregu skryptów JavaScript. Wszystkie te problemy świetnie opisuje w swoim artykule [Jake Archibald](#).

Aby dać twórcom więcej swobody w implementacji cache'u w aplikacjach webowych, zdefiniowano mechanizm Service Workers. Dzięki niemu zyskali oni możliwość kontroli pobierania wszystkich zasobów z domeny, dla której są one zainstalowane z poziomu JavaScriptu.

JAK DZIAŁAJĄ SERVICE WORKERS

Aktualnie, Service Workers są domyślnie wspierane w Chromie, zaś w przypadku przeglądarki Firefox niezbędne jest włączenie flagi `dom.serviceWorkers.enabled` w `about:config`.

Jeśli będziemy chcieli używać Service Workers w swojej aplikacji, musi ona działać w protokole HTTPS. Ze względu na potęgowe możliwości, jakie daje ten mechanizm, udostępnianie go przez HTTP groziłoby ryzykiem ataku *man-in-the-middle* umożliwiającego przejęcie stałej kontroli nad domeną w obrębie przeglądarki używanej przez użytkownika w momencie ataku. Wyjątkiem jest localhost – w którym to obostrzenie nie obowiązuje, co pozwala na przeprowadzanie testów.

Skrypt Service Worker wykonuje się w JavaScript w kontekście niezależnym od głównej strony, co oznacza, że nie ma z niego dostępu do drzewa DOM. Aby zdefiniować w aplikacji mechanizm Service Worker, należy w skrypcie strony umieścić następujący kod:

Listing 1. Definicja podstawowego Service Workera.

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js').then(function(r) {
    console.log('ServiceWorker zarejestrowany.')
  }).catch(function(e) {
    console.log('Ups! Błąd przy rejestracji ServiceWorkera! '+e)
  });
}
```

W opisanym wyżej kodzie rejestrujemy Service Worker, definiując jego ścieżkę na `/sw.js`. Wszystkie akcje Service Worker wykonywane są w ramach zdarzeń (eventów): `install`, `activate`, `fetch`. tzn. w pliku `/sw.js`.

Oto przykładowy kod:

Listing 2. Podstawowa definicja Service Workera.

```
self.addEventListener('install', function(ev) {
  // Zdarzenie wywoływane po zarejestrowaniu Service Workera
```

Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



```
});  
  
self.addEventListener('activate', function(ev) {  
    // Zdarzenie wywoływanie po aktualizacji pliku Service Workera  
});  
  
self.addEventListener('fetch', function(ev) {  
    // Zdarzenie wywoływanie podczas próby pobrania zasobu  
});
```

Typowo, w zdarzeniu `install` wypełniamy cache jakimś początkowymi danymi, w zdarzeniu `activate` bierzemy pod uwagę możliwe zmiany w cache'u (czyli np. przestajemy cache'ować jeden z zasobów), zaś zdarzenie `fetch` jest wywoływanie w przypadku pobrania jakichkolwiek zasobów. Co ważne – zdarzenie `fetch` jest wywoływanie **zawsze** przy próbie pobrania zasobu – bez znaczenia, czy jesteśmy offline, czy online. Co więcej: jeżeli – na przykład – plik `index.html` zawiera odwołanie do pliku obrazka z innej domeny, to zapytanie też przechodzi przez Service Workera! Takie zachowanie powinno od razu zapalić lampkę ostrzegawczą w głowie każdego testera bezpieczeństwa i naturalnym wydaje się pytanie: „co się stanie, jeżeli złośliwy użytkownik będzie mógł zdefiniować własnego Service Workera?”. Odpowiedź poznamy poniżej.

PRZYKŁAD

Aby zrozumieć lepiej działanie Service Workerów, posłużymy się prostym przykładem.

1. Utwórzmy dwa pliki: `index.html` i `sw.js`.

a. Zawartość `index.html`

```
<!doctype html>  
<html>  
  <head>  
    <script>  
      navigator.serviceWorker.register('/sw.js').catch(e=>console.error('Ups! ' + e))  
    </script>  
  </head>  
  <body>  
    Tutaj nic nie ma.  
  </body>  
</html>
```

b. Zawartość `sw.js`

```
self.addEventListener('fetch', function(ev) {  
  if (ev.request.url.endsWith('.worker')) {  
    ev.respondWith(new Response('<strong>Ten URL istnieje!</strong>',  
    {headers:  
      {"Content-type": "text/html"}  
    }));  
  }  
});
```

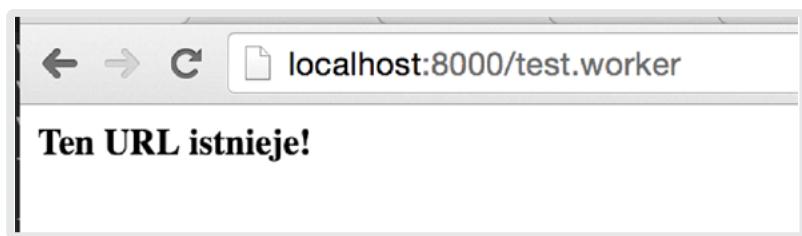
2. Następnie, musimy oba pliki uruchomić na serwerze – proponuję w tym celu używać Pythona i jego prosty serwer WWW, wywołując polecenie:

```
python -mSimpleHTTPServer
```

Wówczas na porcie 8000 zacznie nasłuchiwać serwer listujący pliki z obecnego katalogu.

3. Wróćmy jednak do plików, które zdefiniowaliśmy. Mamy `index.html`, którego jedyną rolą jest zarejestrowanie Service Workera spod adresu `/sw.js` – skrypt z Service Workerem musi zawsze znajdować się w tym samym originie, co skrypt go wywołujący. Sam Service Worker z kolei działa dość prosto. Sprawdzi, czy adres URL żądania kończy rozszerzeniem „.worker” – a jeżeli tak jest – zwraca w odpowiedzi pogrubiony tekst: **Ten URL istnieje!**.

4. Gdy wejdziemy z poziomu przeglądarki najpierw pod `http://localhost:8000`, a następnie pod adres `http://localhost:8000/cokolwiek.worker` – zobaczymy, że oczywiście dostaniemy odpowiedź z Service Workera (rysunek 1).



Rysunek 1. Odwołanie się do adresu `test.worker`.

Ten bardzo prosty przykład pokazuje dobrze, że dzięki Service Workerowi kontrolujemy treść odpowiedzi HTTP wykonywanych do dowolnych zasobów w ramach

Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



jednej domeny, co umożliwia także definiowanie odpowiedzi do zasobów, które nie istnieją (np. plik test.worker nie istnieje w ogóle na serwerze).

W ramach eksperymentu możemy spróbować usunąć z serwera plik sw.js, zrestartować przeglądarkę i tym razem sprawdzić, co się stanie po odwołaniu do innego zasobu, np. <http://localhost:8000/test2.worker>. Okaże się, że zostanie zwrócona treść zdefiniowana w Service Workerze – mimo tego że sam Service Worker już nie istnieje.

Wniosek: gdy chcemy zrezygnować z używania Service Worker w swojej aplikacji webowej, musimy pamiętać o tym, by go wyrejestrować.

Jak zatem wyłączyć Service Workera?

W Chrome pod adresem chrome://inspect/#service-workers możemy zobaczyć listę działających Service Workerów dla wszystkich domen (rysunek 2). Intuicyjnie może wydawać się, że po kliknięciu na przycisk „Terminate”, Service Worker zostanie zatrzymany. Okazuje się, że nie; <http://localhost:8000/test3.worker> nadal zwraca wynik z Service Workera, bowiem zostaje on na nowo uruchomiony przy próbie dostępu do zasobu z naszej domeny.

The screenshot shows the DevTools interface with the 'Service workers' tab selected. It lists three registered Service Workers:

- Worker pid:18624 http://localhost:8000/sw.js
 - [inspect](#)
 - [terminate](#)
- Worker pid:18629 https://plus.google.com/serviceworker.js
 - [inspect](#)
 - [terminate](#)
- Worker pid:18629 https://www.google.pl/_/chrome/newtab-serviceworker.js
 - [inspect](#)
 - [terminate](#)

Rysunek 2. Lista Service Workerów.

Przeglądarka Chrome ma jeszcze jeden specjalny URL pokazujący listę Service Workerów, mianowicie: chrome://inspect/#service-workers (rysunek 3). Na tym ekranie pojawia się przycisk „Unregister” i dopiero po jego kliknięciu, Service Worker zostanie wyrejestrowany, a zdefiniowane wcześniej URL-e przestaną działać.

The screenshot shows the DevTools interface with the 'ServiceWorker' tab selected. It lists a single registered Service Worker:

- Scope: <http://localhost:8000>
- Registration ID: 17
- Active worker:
- Installation Status: ACTIVATED
- Running Status: STOPPED
- Script: <http://localhost:8000/sw.js>
- Version ID: 7136
- Renderer process ID: -1
- Renderer thread ID: -1
- DevTools agent route ID: -2
- Log:

At the bottom, there are 'Unregister' and 'Start' buttons.

Rysunek 3. Inna lista Service Workerów, z opcją „Unregister”.

PROBLEMY BEZPIECZEŃSTWA

Pokazaliśmy powyżej przykład Service Workera, który podstawał własną odpowiedź do zasobów, których URL kończy się na „.worker”. Pokazaliśmy też, że z punktu widzenia użytkownika końcowego, raz zdefiniowany Service Worker może być trudny do usunięcia.

W świetle tych faktów możemy wyciągnąć dwa wnioski:

- » jeżeli znajdziemy w danej domenie błąd XSS (Cross Site Scripting), który pozwoli na zdefiniowanie własnego Service Workera, będziemy mogli w tej domenie podmienić każdą podstronę; innymi słowy: możemy przeprowadzić prawdziwie permanentnego XSS-a na wszystkich podstronach;
- » nawet jeśli atakowany użytkownik zorientuje się, że coś jest nie w porządku, i tak może mieć problem z usunięciem złośliwego Service Workera.

W większości przypadków wprowadzenie złośliwego Service Workera nie będzie proste. Wymagane jest bowiem, żeby skrypt Service Workera zwracał nagłówek Content-Type: application/javascript. W większości aplikacji webowych, nie mamy możliwości wstrzygnięcia w tego typu zasoby. Wyjątkiem jest sytuacja, w której aplikacja udostępnia interfejs JSONP. Wtedy zazwyczaj mamy kontrolę

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



nad parametrem *callback*, w którym bardzo często możemy umieścić dowolny kod JavaScript.

Na przykład: https://example.com/jsonp?callback=<złośliwy_kod>.

Wskazówka: jeżeli używamy na swoich stronach JSONP, warto zadbać o ograniczenia do parametru *callback*, np. tylko znaki alfanumeryczne i kropka, maksymalna długość: 20 znaków.

Wydaje się, że przeglądarki internetowe mogłyby robić więcej, jeśli chodzi o ochronę użytkowników przed złośliwym użyciem Service Workerów. Dotychczas w nowoprowadzonych mechanizmach, jak np. CORS, stosowana była zasada opt-in, tzn. jawne określenie w nagłówku HTTP zgody na pobieranie zasobów z naszej domeny przez inne domeny.

W przypadku Service Workerów nie ma możliwości, aby w jawnym sposób zadeklarować, że tylko jeden URL może być używany jako skrypt. Chrome wysyła w pytaniu dodatkowy nagłówek Service-Worker: *script*, więc ewentualnym zaatakowaniem po stronie serwera mogłyby być odrzucanie wszystkich żądań z tym nagłówkiem (jeśli nie używamy u siebie w ogóle Service Workerów).

PODSUMOWANIE

Mechanizm Service Workers umożliwia programistom aplikacji webowych definiowanie sposobu pobierania zasobów (np. gdy aplikacja znajduje się w trybie offline). Ze względu na jego potężne możliwości, błąd typu XSS może zostać wykorzystany do permanentnej podmiany zawartości wszystkich podstron.

Aby zabezpieczyć się przed atakiem z wykorzystaniem mechanizmu Service Workers, należy zadbać o to, by użytkownik nie miał zbyt dużych możliwości umieszczania treści bezpośrednio w skryptach JavaScript. Dodatkowym środkiem bezpieczeństwa może być zmiana konfiguracji serwera polegająca na odrzucaniu żądań z nagłówkiem Service-Worker: *script*.

Michał Bentkowski. Realizuje testy penetracyjne oraz audyty bezpieczeństwa w firmie [Securitum](#). Autor w serwisie sekurak.pl. Aktywnie (i z sukcesem) uczestniczy w znanych programach bug bounty.



Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

W 2014 roku na Sekuraku (oraz w [pierwszym e-zinie](#)) pisaliśmy [czym jest Content Security Policy](#). Była to wtedy młoda technologia utrudniająca eksploatację ataków XSS. Od tego czasu standard mocno się rozwinął. Czy obecna wersja CSP 2.0 jest remedium na nieznane luki XSS? Na jakie problemy można natknąć się podczas wdrożeń CSP? Na te pytania postaram się odpowiedzieć w kolejnych sekcjach poniższego artykułu.

SŁOWEM WSTĘPU

CSP zainteresował mnie kilka lat temu, gdy zacząłem zgłębiać bezpieczeństwo technologii skupionych wokół HTML5. Z nagłówkami Content-Security-Policy miałem styczność jako web developer oraz pentester. W zależności od roli projektowej i doświadczenia, CSP było moim przyjacielem lub wrogiem. Sądzę, że warto zaznajomić się z tą technologią i wyciągnąć z niej to, co najlepsze, by zabezpieczyć użytkowników aplikacji webowych, ale także – aby nie utknąć na zbyt dużych restrykcjach w czasie testów.

W sieci istnieje wiele poradników dotyczących CSP. Niestety te, na które trafiłem, zawsze miały wspólną wadę – nie odwoływały się do rekomendowanej wersji standardu (obecnie – drugiej). Autorzy poradników często mieszają mechanizmy proponowane przez W3C z niestandardową implementacją przeglądarek oraz nie wyjaśniają różnic. Z tego powodu projekty z CSP, z którymi miałem do czynienia, posiadały bardzo liberalne zasady nieznacznie podnoszące poziom bezpieczeństwa. Obserwowałem też wdrożenia CSP, które całkowicie nie działały lub powodowały popuszczenie aplikacji internetowej.

Aby pomóc innym programistom i testerom, zdecydowałem się napisać artykuł, którego źródłem są wyłącznie dokumentacje W3C oraz moje zawodowe doświadczenie.

O CZYM JEST TEN ARTYKUŁ

Artykuł pisany jest w kontekście zwiększenia bezpieczeństwa („utwardzania”/hardeningu) aplikacji internetowych. Testowanie, omijanie i atakowanie CSP będzie tematem osobnej publikacji.

Poniższa publikacja (pisana na początku 2016 roku) omawia technologię Content Security Policy (CSP) w wersji 1.0 oraz 2.0 według oficjalnej dokumentacji W3C:

- » [Content Security Policy 1.0 W3C Candidate Recommendation 15 November 2012](#)
(wyłącznie do celów informacyjnych).
- » [Content Security Policy Level 2, W3C Candidate Recommendation, 21 July 2015.](#)

Ten materiał jest efektem wniosków z analizy fragmentów unormowanych oraz nie-normatywnych dokumentacji W3C. W obu przypadkach analizowałem tylko stabilne, rekomendowane standardy (o statusie CR), które *nie powinny* zostać zmienione.

W momencie pisania tekstu, zostały rozpoczęte prace nad CSP 3. Ze względu na to, że niektóre elementy CSP 3 zostały już zaimplementowane w przeglądarkach, w artykule również się do nich odnioszę. Z uwagi na możliwe zmiany w dokumentacji i rzetelność opracowania, referencje do trzeciej wersji CSP będą w niniejszym opracowaniu wyraźnie zaznaczone.

W poniższym artykule zagadnienia dotyczące CSP 3 (oraz wyższych) będą poruszane tylko na poziomie ogólnym.

CZYM JEST CSP?

W chwili tworzenia aplikacji internetowej z reguły wiadomo, jakie zasoby trafiają do przeglądarek końcowych użytkowników. Programiści nie powinni mieć większego problemu z określeniem, z jakiego miejsca zostaną pobrane pliki HTML, skrypty, style czy multimedia.

Można by więc pokusić się o wskazanie dozwolonych źródeł (domen, ścieżek, protokołów) dla różnych grup zasobów. Gdyby przeglądarka respektowała takie wartości, to użytkownicy aplikacji (oraz agresorzy), nie mogliby spowodować załadowania (złośliwych) zasobów (skryptów, appletów...) z domeny niewskazanej wprost przez programistę.

Content Security Policy (CSP) jest właśnie koncepcją wykorzystującą listy dozwolonych źródeł, z których mogą zostać załadowane zasoby strony. Programista tworzy białą listę hostów i ścieżek, którą dodaje do nagłówka odpowiedzi strony internetowej. Nowoczesna przeglądarka respektuje tę listę i – w razie naruszenia polityki – blokuje ładowanie oraz wykonywanie niechcianego zasobu.

Włączenie polityk CSP w znacznym stopniu utrudnia przeprowadzenie udanych ataków XSS, UI Redressing (Clickjacking), złośliwego wykorzystania ramek czy

Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



wstrzyknień CSS. Gdy agresor będzie próbował dodać szkodliwe elementy do strony (np. przez nieznaną podatność), restrykcje CSP zatrzymają żądanie o wrogi zasób (np. skrypt). Nawet gdy w aplikacji zostanie odnaleziona podatność XSS, potencjalny atakujący może mieć bardzo utrudnione zadanie w jej wykorzystaniu, ponieważ ładunki („payloady”) podsylane ofiarom nie załadują szkodliwych skryptów w miejscu wstrzyknięcia.

PIERWSZE KROKI Z CSP

CSP można włączyć na dwa sposoby:

- » przez specjalnie przygotowany nagłówek HTTP – Content-Security-Policy: \$polityki-csp,
- » przez dodanie znacznika meta w sekcji <head> kodu HTML – <meta http-equiv="Content-Security-Policy" content="\$polityki-csp">.

Zalecany jest pierwszy wariant – CSP włączone przez znacznik <meta> (dla bezpieczeństwa) nie interpretuje wszystkich dyrektyw.

Domyślną polityką, od której zaczyna się budowanie nagłówka, jest: **blokuj wszystko**. Programista modyfikując wartość CSP, rozluźnia obostrzenia dla konkretnych grup zasobów (np. osobno dla skryptów, obrazków itp.). Gdy nowoczesna przeglądarka, wspierająca użyte dyrektyny, odbierze wraz z treścią strony nagłówek:

Content-Security-Policy: default-src 'self' cdn.example.com; img-src img.example.com; to podczas przetwarzania kodu HTML (a potem podczas działania strony) powodującego wysłanie żądań o skrypty, style czy fonty, pozwoli wykonać je wyłącznie do:
» tej samej domeny, co załadowana strona ('self'),
» do domeny cdn.example.com (korzystając z protokołu HTTP oraz HTTPS).

W powyższym przykładzie obrazki będą mogły zostać załadowane tylko spod adresu `http(s)://img.example.com/*`. Nie będzie możliwości wczytania zasobów z innych hostów. Wskazana polityka CSP zablokuje również wywoływanie skryptów umieszczonych między znacznikami <script></script>. Nie wykonają się również akcje z atrybutów takich jak onclick czy oninput, gdyż wykonanie tzw. „skryptów inline” musiałoby zostać dopuszczone przez dodanie 'unsafe-inline':

Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline'

Jak widać, użycie CSP jest bardzo proste i intuicyjne. Zanim zapoznamy się z elementami, które możemy chronić, najpierw przyjrzyjmy się standardom stojącym za technologią oraz ich praktycznemu wsparciu w obecnych przeglądarkach.

Jeśli jeszcze nie czujesz się pewnie z wiedzą o technologii CSP oraz o jej głównych założeniach, przed dalszą lekturą polecam zapoznanie się z artykułem Rafała ‘Blade’ Janickiego „Czym jest Content Security Policy?” z 2014 roku.

BAŁAGAN STANDARDÓW I IMPLEMENTACJI

Content Security Policy powstało z inicjatywy pracowników Google oraz Mozilli. Od samego początku przeglądarek tych dwóch firm najszybciej wprowadzały nowe rozszerzenia CSP, oczywiście każda w innym czasie. Po kilku latach firma Microsoft – jakby pod naciskiem konkurencji – również podjęła pierwsze próby implementacji CSP w swoich produktach. Ostatecznie pomysł na nowy standard trafił pod skrzydła grupy roboczej W3C.

Z początkiem 2016 roku CSP w wersji 1.0 zostało **całkowicie zastąpione** przez wersję drugą. Można również zapoznać się ze szkicem (W3C draft) **trzeciej wersji standardu**, uzupełnianej o nowe mechanizmy zabezpieczeń.

Content Security Policy w pierwszej wersji zostało bardzo dobrze przyjęte przez inżynierów bezpieczeństwa. Nieco gorzej wypadło w oczach web developerów, ponieważ w niektórych przypadkach, CSP okazało się bardziej skomplikowane niż przypuszczało. Dlatego też szybko rozpoczęto prace nad usprawnieniami, oznaczonymi numerem 1.1. Standard powoli się rozrastał, aż W3C zdecydowało się zmienić numerację na Level 2, zamrozić proces zmian i kontynuować pracę nad wersją trzecią.

Mogłoby się wydawać, że prace nad CSP przebiegały bardzo sprawnie – przeglądarki szybko implementowały kolejne dyrektywy CSP, a grupy robocze płynnie tworzyły nowe mechanizmy zabezpieczeń. Problem pojawił się w innym miejscu – po stronie użytkowej, kiedy programiści chcieli się nauczyć, jak korzystać z CSP oraz przekazać tę wiedzę dalej. Pojawiły się więc poradniki oraz benchmarki, do których – niestety – wkradło się wiele niespójności.

Do niedawna jeszcze mieliśmy niemały bałagan w samej standaryzacji i stopniu jej implementacji; na szczęście proces zaczyna się porządkować, niemniej nadal można trafić na wdrożenia CSP pisane w duchu wsparcia:

- » niestandardyzowanych wersji nagłówków (np. X-Content-Security-Policy) lub niestandardowych dyrektyw (np. allow zamiast default-src),

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



- » CSP 1.0,
- » wybranych elementów CSP 1.1,
- » wcześniejszej (rozwijanej) wersji CSP 2.0,
- » oficjalnej (stabilnej) wersji CSP 2.0,
- » wybranych elementów CSP 3.0.

JAK UNIKNAĆ PROBLEMÓW

Aby uniknąć problemów w przyszłości, przed rozpoczęciem definiowania polityk CSP należy:

- » jasno ustalić wymagania dotyczące wersji wspieranych przeglądarek przez klientów aplikacji webowych,
- » najważniejsze polityki opierać na stabilnej wersji standardu – obecnie „W3C Level 2 Candidate Recommendation”,
- » przeanalizować, czy dyrektywy z nowszych szkiców W3C mogą zwiększyć bezpieczeństwo klientów końcowych; jeśli tak – warto je wprowadzić, ale z zastrzeżeniem, że w przyszłości ich wsparcie może się zmienić,
- » wykonać testy wsparcia najważniejszych dyrektyw dla kluczowych przeglądarek.

Polecam z rezerwą traktować tzw. „benchmarki wsparcia” oraz „generatory CSP”, które można znaleźć w sieci. Porównania i narzędzia szybko się dezaktualizują i mogą wprowadzić w błąd. Bardzo ważne jest testowanie wdrożonej polityki na różnych przeglądarkach i podgląd oficjalnej dokumentacji.

Obecnie jednym z najlepszych narzędzi online wspierających proces tworzenia CSP, jest [Report-URI CSP Builder](#). Pamiętajmy jednak, że tego rodzaju narzędzia mogą się zdezaktywować lub posiadać błędy.

WSPARCIE W PRZEGŁĄDARKACH

Wsparcie nowych technologii WWW w przeglądarkach można sprawdzić w serwisie „Can I use”. Pod adresem <http://caniuse.com/#search=CSP> znajdziemy podstawowe informacje o stopniu implementacji CSP w różnych przeglądarkach.

Wersje CSP 1.0 oraz 2.0 nie powinny się już zmieniać, więc <http://caniuse.com> jest dość dobrym (i często aktualizowanym) źródłem wiedzy. Trzeba tylko pamiętać o adnotacjach i wyszukiwać informacji o pełnym wsparciu i ze szczególną rezerwą podchodzić do przeglądarek z „częściowym wsparciem” – na przykład IE 10/11

w praktyce implementuje tylko jedną dyrektywę CSP 1.0 (sandbox), w dodatku przez niestandardowy nagłówek. Dość często Internet Explorer okazuje się więc niemiłą niespodzianką.

Pamiętając o tego rodzaju problemach, przyjrzyjmy się ogólnie, w jakich przeglądarkach możemy liczyć na wsparcie Content Security Policy.

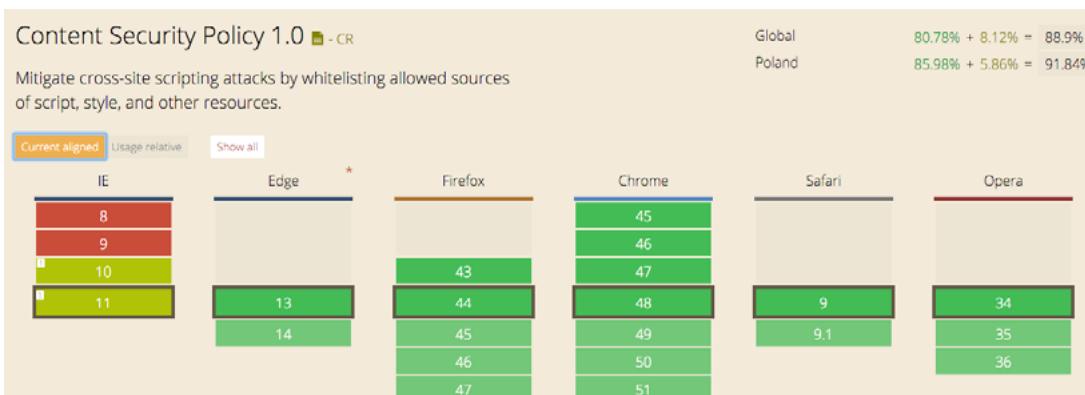
Wsparcie CSP 1.0

Obecnie poziom wsparcia pierwszej wersji standardu jest bardzo wysoki. Kompletna implementacja standardu W3C została wdrożona w:

- » Internet Explorer Edge (12+),
- » Firefox (23+),
- » Chrome (25+),
- » Safari (7+),
- » a także w mobilnych wersjach: Safari, Android Browser/Web View/Chrome.

Problemem jest wyłącznie **Internet Explorer 10/11**, który obsługiwany jest tylko przez nagłówek 'X-Content-Security-Policy' i respektuje wyłącznie dyrektywę sandbox. Inne dyrektywy, takie jak chociażby script-src czy style-src, nie są interpretowane przez Internet Explorer w wersjach poniżej „Edge”.

Można założyć, że Internet Explorer w wersji 10 oraz 11 (i starszych) jest przeglądarką niewspierającą CSP.



Rysunek 1. CSP 1.0 wspierane przez około 90% przeglądarek internautów.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

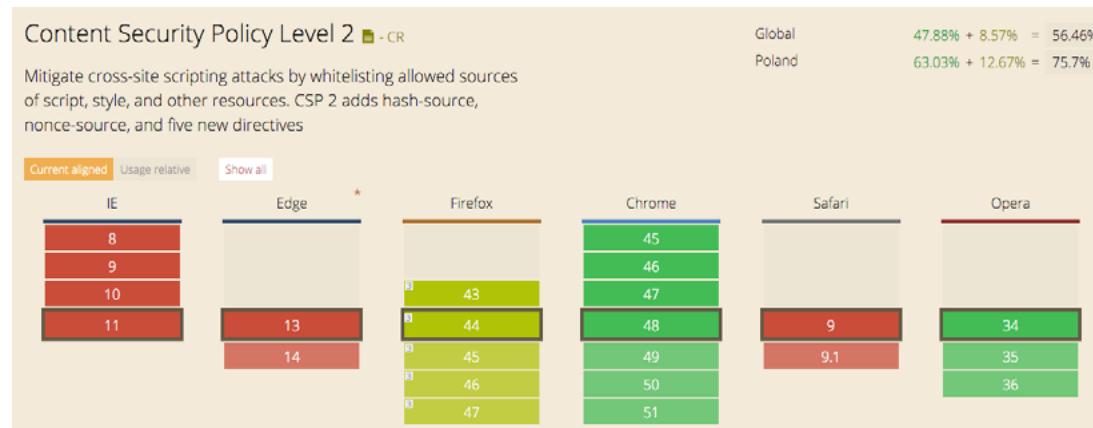
Mechanizm HTTP Public Key Pinning



Wsparcie CSP 2.0

Na początku 2016 roku, kompleksowym wsparciem CSP 2.0 może pochwalić się tylko Google Chrome i przeglądarki oparte na jej silniku (Opera, Opera Mobile, Android WebView, Android Chrome). Nieźle radzi sobie również Firefox, który od wersji 36 ma problemy tylko z interpretacją plugin-types oraz child-src. Za kilka miesięcy z pewnością „dogoni” przeglądarkę Google, zapewniając stuprocentowe wsparcie CSP 2.0.

Internet Explorer, Internet Explorer Edge, Safari oraz wersje mobilne tych przeglądarek na początku 2016 roku nie wspierały CSP 2.0.



Rysunek 2. CSP 2.0 wspierane przez ponad połowę przeglądarek internautów (w Polsce – 75%).

Wsparcie CSP 3

W celu szybszego wdrożenia wersji drugiej, W3C zdecydowało się na wyrzucenie niektórych dyrektyw CSP do kolejnej wersji standardu. Część z nich została już zaimplementowana przez twórców przeglądarek, więc możemy mówić o pierwszych wdrożeniach CSP Level 3.

Trudno jednak obecnie wyciągać wnioski o poziomie wsparcia CSP 3.0 w przeglądarkach, ponieważ standard ciągle ewoluje. Niektóre z nowości są bardzo ciekawe i można pokusić się o ich wdrożenie – ale należy pamiętać, że z dnia na dzień może zmienić się sposób ich działania.

Więcej o nowinkach CSP Level 3 w sekcji: „Co się kończy, coś zaczyna – CSP 3.0?”.

SKŁADNIA POLITYK

Content-Security-Policy jest nagłówkiem odpowiedzi serwera, którego wartością są polityki stopniowo rozluźniające zasadę „blokuj wszystko”.

Polityka CSP (CSP policy) składa się z dwóch elementów:

1. dyrektywy (directive),
2. list źródeł (source list).

Ogólny schemat polityki (dyrektywy) wygląda następująco:

```
directiveX: source1 source2 sourceN;
```

Poniższy zapis definiuje więc trzy polityki:

```
Content-Security-Policy: script-src 'self'; img-src 'self'; style-src 'self' cdn.com;
```

Dyrektyny rozdziela się średnikami, białe znaki nie mają znaczenia. Powyżej widzimy restrykcje dla skryptów, obrazków oraz stylów. Nie zmieniono definicji domyślnego zachowania, więc inne elementy w takim przypadku (np. fonty) byłyby blokowane.

Dyrektyny

Dyrektywami są słowa kluczowe, które opisują reguły dostępu do zasobu. Gdy przeglądarka nie wspiera danej dyrektywy, pominie ją i zacznie przetwarzać następną (czyli w przypadku braku wsparcia, przeglądarka nie przestaje nagle interpretować CSP).

Poniżej znajduje się lista dyrektyw oraz informacja, od której wersji standardu dana dyrektywa powinna być wspierana:

- » base-uri (CSP 2.0),
- » child-src (CSP 2.0, zastępuje frame-src),
- » connect-src (CSP 1.0),
- » default-src (CSP 1.0),
- » font-src (CSP 1.0),
- » form-action (CSP 2.0),
- » frame-ancestors (CSP 2.0),
- » frame-src (tylko CSP 1.0, obecnie przestarzałe),
- » img-src (CSP 1.0),
- » media-src (CSP 1.0),
- » object-src (CSP 1.0),
- » plugin-types (CSP 2.0),

Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



- » report-uri (CSP 1.0),
- » sandbox (CSP 1.0),
- » script-src (CSP 1.0),
- » style-src (CSP 1.0).

List nr 1. Oficjalna lista dyrektyw CSP 1.0 oraz CSP 2.0.

Istnieje też kilka bardzo ciekawych dyrektyw, które ostatecznie wypadły z CSP 2.0, ale mogą trafić do trzeciej wersji CSP:

- » upgrade-insecure-requests,
- » block-all-mixed-content,
- » manifest-src,
- » referrer,
- » reflected-xss.

List nr 2. Wybrane dyrektywy, które mogą wejść w skład nowszych wersji CSP.

W sekcji „Coś się kończy, coś zaczyna – CSP 3.0?” przyjrzymy się tym dyrektywom dokładniej.

Lista źródeł dyrektywy

Na listę źródeł dyrektywy może trafić:

- » **host:**
 - » niepoprzedzony protokołem (oznacza wtedy dopasowanie z HTTP oraz HTTPS) – example.com;
 - » poprzedzony protokołem – https://example.com;
 - » wraz z definicją portu – example.com:8080;
 - » schemat, domena i/lub port może zawierać znak wieloznaczny * (ale w części hosta musi znajdować się skrajnie z lewej strony) – https://*.example.com:*
- » **host wraz ze ścieżką** (od CSP 2.0):
 - » http://example.com/js/plik.js – dopasowanie do pliku plik.js znajdującego się pod adresem http://example.com/js/plik.js;
 - » example.com/js – pozwala ładować tylko pojedynczy plik js w domenie example.com (nie dotyczy katalogu);
 - » example.com/js/ – pasuje do całego katalogu js/ w domenie example.com
 - » 'none' (zabroni);

- » 'self' (pozwolenie dla tego samego hosta, zgodnie z zasadą *Same Origin Policy*);
- » 'unsafe-inline' (pozwala na skrypty, style, obrazki definiowane bezpośrednio w kodzie HTML);
- » 'unsafe-eval' (pozwala na stosowanie niebezpiecznych funkcji typu eval);
- » nonce-XXXX – o tym w sekcji „Funkcje zaawansowane > nonce” (CSP 2.0);
- » sha256-XXXX, sha364-XXXX lub sha512-XXXX (od CSP 2.0) – o tym w sekcji „Funkcje zaawansowane > Inline Hash”;
- » specyficzna fraza dla danej dyrektywy (dokładny opis w szczegółach dyrektywy).

List nr 3. Wartości, które mogą trafić na listę źródeł dyrektyw.

Warto wspomnieć o kilku najczęstszych błędach w implementacji dyrektyw:

- » przy politykach default-src 'self'; img-src cdn.com; obrazki zostaną załadowane tylko z domeny cdn.com (a nie z tej samej domeny oraz z domeny cdn.com);
- » odwołanie do obrazka nie zadziała, gdy dozwolonym hostem będzie localhost;
- » gdy przy polityce img-src localhost użyjemy , to obrazek wyświetli się tylko podczas odwiedzenia adresu http://localhost (nie zadziała w przypadku http://127.0.0.1);
- » zapis hostu lub ścieżki w cudzysłowie jest błędny – przy img-src 'localhost' nie będziemy w stanie wyświetlać żadnych obrazków.

W sekcji standardu [CSP 4.2.2 \(Matching Source Expressions\)](#), w kroku 4.8, zawarto ostrzeżenie, że adresy IPv4 oraz IPv6 obecnie nie są wspierane, ale może się to zmienić w przyszłości. W praktyce jednak nie stwierdziłem problemów z podawaniem adresów IP jako hosta.

SZCZEGÓŁOWY OPIS DYREKTYW

default-src

Domyślana polityka CSP blokująca dostęp do niezdefiniowanego źródła może zostać zmieniona przy pomocy dyrektywy default-src.

Gdy mamy wpływ na CSP w trakcie tworzenia aplikacji webowej (np. jako programista), dobrym pomysłem jest rozpoczęcie od pojedynczej, restrykcyjnej dyrektywy default-src ('none' lub 'self'), a następnie „rozluźnianie” obostrzeń przez uszczegóławianie źródeł skryptów, obrazków itp. Dla zachowania czytelności,

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatność w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



warto też dodawać default-src jako pierwszą politykę, nawet, gdy jej wartością miałoby być słowo kluczowe 'none'.

Należy zaznaczyć, że po definicji default-src, wartości domyślne ustawiane są tylko na **wybrane i niezdefiniowane dyrektywy**.

Poniżej lista dyrektyw, które mogą przybrać wartości domyślne (według CSP 2.0):

- » child-src (tylko w CSP 2.0! zastępuje frame-src),
- » connect-src,
- » font-src,
- » frame-src (przestarzały),
- » img-src,
- » media-src,
- » object-src,
- » script-src,
- » style-src.

Lista nr 4. Dyrektywy, które mogą przyjąć wartości domyślne.

Pamiętaj, że default-src nie wpływa na wszystkie dyrektywy oraz na dyrektywy już zdefiniowane!

base-uri (CSP 2.0; nie przyjmuje wartości default-src)

Ta dyrektywa odnosi się do ograniczeń nakładanych na element <base>. Tag „base” w HTML pozwala określić:

- » bazowy adres URL dla linków względnych,
- » oraz domyślny kontekst nawigacji (np. target=_blank otwierający odnośnik w nowym oknie).

Dodanie lub manipulacja wartości znacznika „base” przez atakującego może skutkować załadowaniem złośliwych zasobów (np. skryptów) z obcych źródeł. Definiując wartość base-uri, możemy znacznie utrudnić ataki wykorzystujące ten element.

child-src (CSP 2.0; może przyjąć wartość domyelną)

Dotyczy dodatkowych (zagnieżdżonych) kontekstów przeglądarki (**nested browser context**), czyli elementów takich jak ramki frame/iframe oraz „web workerzy” HTML5 (konkretnie są to restrykcje konstruktora Worker oraz SharedWorker).

Zastępuje frame-src z CSP 1.0.

connect-src (CSP 1.0; może przyjąć wartość domyelną)

Pozwala określić, co może być celem żądań asynchronicznych XMLHttpRequest send(), konstruktorów WebSocket, EventSource lub parametrów funkcji sendBeacon() technologii Beacon.

Dyrektyna ta wpłynie między innymi na takie wywołania w kodzie, jak:

- » (new XMLHttpRequest()).open('GET', 'http://example.com', true);
- » new WebSocket('wss://example.com');
- » new EventSource('https://evil.com');

font-src (CSP 1.0; może przyjąć wartość domyelną)

Tu możemy określić ograniczenia dla fontów webowych, czyli np. źródeł, które pojawiają się w definicji @font-face w stylach CSS.

form-action (CSP 2.0; nie przyjmuje wartości default-src)

Obostrzenia dla atrybutu „action” formularza HTML.

frame-ancestors (CSP 2.0; nie przyjmuje wartości default-src)

Restrykcje zagnieżdżania zasobu w innych miejscach. Dotyczy to ramkowania strony przez takie elementy, jak ramki frame/iframe czy elementy <object>, <embed>, <applet> i podobne.

Obecnie jest to jedno z najlepszych zabezpieczeń przeciwko atakom UI Redressing (w szczególności Clickjacking) oraz innym zagrożeniom, których etapem jest wyświetlenie atakowanej strony wewnątrz ramki.

Domniemaną wartością frame-ancestors jest * (niezależnie od default-src, który nie wpływa na tę dyrektywę). Oznacza to, że strony domyślnie mogą być zagnieżdżane/ramkowane w innych miejscach.

- » Dyrektywa frame-ancestors w CSP 2.0 zastępuje nie do końca ustandaryzowany nagłówek X-Frame-Options, który potrafił sprawiać problemy w mocno rozproszonych środowiskach.
- » Gdy przeglądarka otrzyma dwa nagłówki: Content-Security-Policy z polityką frame-ancestors oraz X-Frame-Options, wykorzystana będzie wyłącznie wartość z CSP.

Na liście dozwolonych źródeł frame-ancestors może się znaleźć:

- » 'none' – strona nie pozwala na ramkowanie/zagnieżdżanie (odpowiednik X-Frame-Options: deny);

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



- » 'self' – strona pozwala na ramkowanie/zagnieżdżanie tylko wtedy, gdy robią to elementy spod tego samego protokołu, adresu i portu (odpowiednik X-Frame-Options: sameorigin);
- » host – wówczas strona pozwoli na zagnieżdżenie swojej zawartości przez wskazanego hosta – stosuje się tutaj standardowy zapis hosta CSP (patrz: lista nr 3), np. frame-ancestors trusted.ads-partner.com (nie było takiej możliwości w X-Frame-Options).

frame-src (CSP 1.0; przestarzałe w CSP 2.0; może przyjąć wartość domyślną)

Starsza, prostsza wersja child-src. Działa podobnie, ale wyłącznie dla ramek iframe/frame. Dyrektywa child-src działa na większą grupę elementów i powinna być używana zamiast frame-src, która oznaczona jest jako „deprecated” i prawdopodobnie zniknie w trzeciej wersji standardu.

img-src (CSP 1.0; może przyjąć wartość domyślną)

Lista dozwolonych źródeł obrazków, które mogą być podane w:

- » znaczniku ,
- » znaczniku <input type=image>,
- » atrybutu poster elementu <video>,
- » atrybutu href elementu <link> służącym m.in. do ustawiania ikony strony,
- » wartości funkcji url(), image(), image-set() (i podobnych) w CSS,
- » innych źródłach obrazków, które w przyszłości będą obsługiwane przez przeglądarki.

media-src (CSP 1.0; może przyjąć wartość domyślną)

Ograniczenia dla dozwolonych źródeł multimediiów (filmy, dźwięki, ścieżki...) wskazywanych przez atrybut src wewnętrz tagów <video>, <audio>, <source> oraz <track>.

object-src (CSP 1.0; może przyjąć wartość domyślną)

Restrykcje dla źródeł, z których mogą być pobierane obiekty obsługiwane przez pluginy przeglądarki. Dotyczy to między innymi apletów Flash, Silverlight oraz Java.

Content Security Policy w tej dyrektywie sprawdza:

- » znaczniki <object> oraz <embed>, które mogłyby zostać użyte do stworzenia zagnieżdzonego kontekstu przeglądarki (tak jak robią to ramki);

- » źródło atrybutów <object data> oraz <embed src>,
- » źródło atrybutów <applet code> oraz <applet archive>.

plugin-types (CSP 2.0; nie przyjmuje wartości default-src)

Za pomocą tej dyrektywy możemy kontrolować rodzaj pluginów uruchamianych przez przeglądarkę do obsługi zasobów takich jak aplety Flash czy Silverlight, zagnieżdżanych przy pomocy znaczników <object> oraz <embed>.

W odróżnieniu od object-src wskazującego skąd obiekt może być załadowany, plugin-types służy do wskazania dozwolonych typów MIME. Gdy wartością dyrektywy będzie:

```
Content-Security-Policy: plugin-types application/pdf  
application/x-shockwave-flash
```

wtedy na odwiedzanej stronie przeglądarka będzie mogła uruchomić wyłącznie plugin do obsługi plików PDF oraz apletów Flash, ale nie będzie mogła obsługiwać apletów Javy oraz Silverlight.

Po włączeniu restrykcji należy upewnić się, że deklaracje obiektów w kodzie HTML mają ściśle zdefiniowany typ MIME – w przeciwnym wypadku nie zostaną załadowane, niezależnie od zawartości:

```
<object data='resource' type='application/pdf'></object>
```

Ze względu na nietypowy format listy źródeł domyślnej polityki default-src nie wpływa na plugin-types. Na liście typów MIME nie mogą również pojawić się znaczniki wieloznaczne *.

report-uri (CSP 1.0; nie przyjmuje wartości default-src)

Jest to specjalna dyrektywa, która wskazuje, pod jaki adres powinien zostać wysłany raport w przypadku naruszenia zasad CSP. Wartością dyrektywy powinien być adres URI.

Dyrektyna report-uri jest ignorowana, jeśli CSP zostało włączone przez znaczniki <meta> (a nie nagłówki HTTP).

Więcej o raportowaniu w sekcji „Funkcje zaawansowane > Raportowanie”.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



sandbox (CSP 1.0; nie przyjmuje wartości default-src)

Użycie tego słowa kluczowego w CSP spowoduje wyświetlenie całego zasobu (odwiedzanej strony) w trybie piaskownicy HTML5. Tryb ten, pierwotnie wprowadzony jako mechanizm ochrony ramek, pozwala na wyłączenie niektórych mechanizmów przeglądarki.

Gdy zasób jest wyświetlany w trybie „sandbox”, wtedy nakładane są nieiego wszystkie poniższe obostrzenia:

- » wysyłanie formularzy jest wyłączone,
- » konteksty przeglądania (np. ramki) zawsze są limitowane ścisłą zasadą *Same Origin Policy*,
- » wykonywanie skryptów zostaje wyłączone,
- » zmiana nawigacji jest zabroniona (window top navigation),
- » zabrania się otwierania wyskakujących okienek (pop-up),
- » interfejs do blokowania wskaźnika myszy zostaje wyłączony.

Lista nr 5. Restrykcje trybu sandbox w HTML5.

Wszystkie powyższe ograniczenia zostaną wprowadzone, gdy słowo kluczowe *sandbox* trafi na listę CSP:

Content-Security-Policy: sandbox;

Możliwe jest rozluźnienie niektórych obostrzeń trybu piaskownicy. Aby to zrobić, wystarczy na listę dozwolonych źródeł dyrektywy *sandbox* dodać wybrane słowa kluczowe z poniższej listy (dowolny zestaw oddzielony spacjami, bez cudzysłowów):

- » allow-forms
- » allow-same-origin
- » allow-scripts
- » allow-top-navigation
- » allow-popups
- » allow-pointer-lock

W poniższym przykładzie:

Content-Security-Policy: sandbox allow-scripts allow-popups;

załadowana strona będzie mogła wykonywać kod Javascript oraz tworzyć okna pop-up. Niezależnie od tego, czy logika wysyłania formularzy zostanie zaimplementowana przez kod HTML czy Javascript, ich wysłanie zostanie zabronione (nawet przy włączonym Javascript). Na stronie nie będzie również możliwości zmiany adresu ramkijącej (nadzędnej) strony (zmiana adresu kontekstu nadzędnego przez window.top.location.href).

Szczegółowe informacje o HTML5 Sandboxing można przeczytać na [stronach W3C](#).

script-src (CSP 1.0; może przyjąć wartość domyślną)

Sztandarowa dyrektywa CSP, która włącza restrykcje dla skryptów na stronie. Na liście źródeł tej dyrektywy mogą pojawić się:

- » dozwolone hosty (CSP 1.0), a nawet ścieżki (CSP 2.0);
- » słowa kluczowe 'none' (zabronić) lub 'self' (zezwól tylko z tego samego źródła);
- » słowo kluczowe 'unsafe-inline', które pozwoli wykonywać skrypty „inline”;
- » słowo kluczowe 'unsafe-eval', które pozwoli wykonywać (ewaluować) w sposób dynamiczny kod Javascript (wpływa na wywołania funkcji takich jak eval(), setTimeout(), setInterval() oraz na konstruktor funkcji Function).

Użycie *script-src* bardzo utrudnia udane wykorzystanie błędów XSS. Lista dozwolonych hostów utrudni atakującemu dołączenie złośliwego skryptu ze swojej domeny. Brak wartości 'unsafe-inline' utrudni wykonanie skryptów „inline”, które są najpopularniejszą metodą dystrybucji ładunku XSS. Brak wartości 'unsafe-eval' utrudni zaś eksploatację błędów DOM XSS.

Zawsze staraj się tworzyć polityki w taki sposób, aby unikać słów kluczowych 'unsafe-inline' oraz 'unsafe-eval'. CSP zostało stworzone głównie w celu wprowadzenia dodatkowej ochrony przed atakami XSS – nie rezygnuj z tej możliwości!

Istnieje możliwość używania skryptów oraz stylów „inline” **bez dodawania 'unsafe-inline'** obniżającego bezpieczeństwo strony. Do tego celu powstały mechanizmy „nonce” oraz „Inline Hash” CSP 2.0. Oba zostały opisane w sekcji „Funkcje zaawansowane”.

style-src (CSP 1.0; może przyjąć wartość domyślną)

Jest to bardzo często używana dyrektywa, która działa analogicznie do *script-src*, tylko dotyczy kaskadowych arkuszy stylów – czyli plików CSS, oraz stylów definiowanych wprost w kodzie HTML („inline”).

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



FUNKCJE ZAAWANSOWANE

Raportowanie (CSP 1.0)

Po dodaniu nagłówka Content-Security-Policy przeglądarki od razu zaczyną respektować wdrożone zasady i chronić użytkowników aplikacji webowej.

Niestety, o przeoczenia w CSP nietrudno – chociażby podczas odwoływanego się do bibliotek hostowanych w sieciach CDN tylko na niektórych podstronach serwisu. W przypadku błędów, przeglądarki zazwyczaj zablokują zbyt dużo elementów na stronie, utrudniając korzystanie z serwisu. Czym aplikacja większa i architektura bardziej rozproszona, tym bardziej zbyt wysokie restrykcje mogą utrudnić, a nawet całkowicie uniemożliwić korzystanie z serwisu.

Remedium na tego rodzaju problemy jest tryb raportowania, który włącza się przez osobny nagłówek HTTP w celach testowych, jeszcze przed głównym wdrożeniem CSP:

Content-Security-Policy-Report-Only: politykiCSP;

Przeglądarka, odbierając powyższy nagłówek, będzie przetwarzala podane polityki, ale w momencie ich naruszenia nie zablokuje zasobu, zwracając błąd w konsoli Javascript (rysunek 3.).

Refused to execute inline event handler because it violates the following Content Security Policy directive: "script-src https://code.jquery.com/ui/1.11.4/jquery-ui.js". Either the 'unsafe-inline' keyword, a hash ('sha256-...'), or a nonce ('nonce-...') is required to enable inline execution. test.php:10

Rysunek 3. Naruszenie CSP widoczne w konsoli Javascript spowodowane wstrzyknięciem ładunku XSS do kodu strony.

Aby rozszerzyć funkcje raportowania, warto użyć dyrektywy report-uri, dzięki czemu podczas naruszenia restrykcji, pod wskazany adres zostanie wysłany raport o tym zajściu. Raport jest zwykłym obiektem JSON, który należy odebrać we właściwym serwisie webowym i np. zapisać w celu dalszej analizy.

Przykładowo – podczas naruszenia poniższych zasad:

Content-Security-Policy: default-src 'self'; report-uri http://example.org/csp-report.cgi

spowodowanych przez próbę załadowania obrazka <http://evil.example.com/image.png>, raport wysłany przez przeglądarkę przybierze postać przedstawioną na listingu 1.

Listing 1. Przykład raportu wysyłanego do serwisu webowego podczas naruszenia polityki CSP.

```
{  
  "csp-report": {  
    "document-uri": "http://example.org/page.html",  
    "referrer": "http://evil.example.com/haxor.html",  
    "blocked-uri": "http://evil.example.com/image.png",  
    "violated-directive": "default-src 'self'",  
    "effective-directive": "img-src",  
    "original-policy": "default-src 'self'; report-uri http://example.org/csp-report.cgi"  
  }  
}
```

Jednoczesne wysłanie nagłówka Content-Security-Policy i Content-Security-Policy-Report-Only może czasem sprawić problemy. Gdy zasób zostanie zablokowany w „CSP”, nie będzie już interpretowany w „Report-Only”. Gdy w nagłówku CSP nie dodamy dyrektywy report-only (obecnej w trybie raportowania), raport z błędem nie zostanie wysłany!

Dostaniemy więc szczegółowe informacje o źródle problemu, polityce, która spowodowała blokadę, a nawet całą zawartość (bardzo przydatne, gdy nagłówek jest generowany dynamicznie przez serwer).

Dyrektywę report-uri można włączyć zarówno w nagłówku Content-Security-Policy, jak i Content-Security-Policy-Report-Only.

Tryb raportowania nie ogranicza się wyłącznie do debugowania poprawności mechanizmu CSP podczas rozwoju oprogramowania czy wdrożenia. Poniżej podaję dwa przykłady, które pokazują, w jaki sposób można połączyć standardowy tryb „CSP”, tryb „Report-Only” oraz report-uri.

CSP jako IDS

Użycie dyrektywy report-uri połączone z systemem składowania raportów ich automatycznej analizy może posłużyć jako prosty system wykrywania włamań (głównie w kontekście nieznanych luk XSS). Raport, który zawiera naruszenia w rodzaju „próba wykonania skryptu inline” lub „ewaluacji funkcji Javascript”, może okazać się bardzo cennym sygnałem dla zespołu developerskiego.

Dyrektywę report-uri można wprowadzić niezależnie od tego, czy w projekcie istnieje możliwość użycia standardowego CSP (w trybie blokowania) czy też nie. Jeżeli programiści nie mogą jeszcze wdrożyć CSP (ze względu na ryzyko błędnego

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



działania strony), to nic nie stoi na przeszkodzie, aby inżynier bezpieczeństwa zaproponował restrykcyjne polityki działające w trybie report-only.

Bezstresowe wdrożenie CSP

Największą skuteczność wdrażania CSP uzyskujemy wtedy, gdy zaczynamy nowy projekt z bardzo rygorystycznymi zasadami stopniowo rozluźnianymi w trakcie rozwoju oprogramowania. Niestety nie zawsze mamy taką możliwość.

Gdy celem hardeningu z wykorzystaniem CSP jest duży serwis, którego działanie nie może zostać zakłócone, dobrym pomysłem jest rozpoczęcie od bardzo pobieżliwych polityk. Potem przy pomocy Content-Security-Policy-Report-Only wprowadzamy ostrzejsze restrykcje, analizujemy raporty potencjalnych błędów i wprowadzamy poprawki. Gdy raporty przestaną napływać, możemy zastąpić wartość CSP zawartością trybu reportowania i zacząć całą operację od nowa, testując jeszcze większe obostrzenia.

Po kilku iteracjach powinniśmy mieć dopracowane reguły w nagłówku CSP, które nie spowodują problemów na stronie.

Nonce (CSP 2.0)

Content Security Policy pokazuje największą siłę, gdy blokuje wywołanie kodu „inline”. Niestety, niekiedy przepisanie takich wstawek kodu może okazać się bardzo skomplikowane. Dlatego też w CSP 2.0 (pierwotnie w CSP 1.1) wprowadzono mechanizmy „nonce” (lub „Inline Hash”), pozwalające na działanie wybranych skryptów (stylów) osadzonych bezpośrednio w kodzie strony.

Mechanizm „nonce” wprowadza się w dwóch krokach.

Po pierwsze należy dodać go do dyrektywy skryptów (stylów) przez słowo kluczowe 'nonce-\$RANDOM' (w pojedynczych cudzysłach), zastępując wartość \$RANDOM trudnym do przewidzenia tokenem (analogicznie jak w przypadku tokenów anty-CSRF).

Następnie tę samą wartość umieszcza się w atrybutie nonce znacznika `<script>` lub `<style>`, których wykonanie chcemy dopuścić.

Pamiętaj, aby zapewnić silną pseudolosowość tokena „nonce”. Nigdy nie wybieraj stałej wartości! Atakujący, który będzie w stanie przewidzieć wartość „nonce”, ominie zabezpieczenia XSS oferowane przez Content Security Policy.

Kilka przykładów zastosowania dyrektywy „nonce” przedstawiono na listingu 2.

Listing 2. Przykład zastosowania dyrektywy „nonce” z CSP 2.0.

```
Content-Security-Policy: default: 'self'; script-src 'self' example.com  
'nonce-Nc3n83cnSAd3wc3Sasdfn939hc3'  
  
<script>  
alert("Skrypt zostanie zablokowany, ponieważ polityka skryptów nie zawiera  
'unsafe-inline'.")  
</script>  
  
<script nonce="EDNnf03nceI0fn39fn3e9h3sdaf">  
alert("Zablokowane, ponieważ nonce polityki skryptów ma inną wartość")  
</script>  
  
<script nonce="Nc3n83cnSAd3wc3Sasdfn939hc3">  
alert("Skrypt zostanie wykonany, ponieważ wartość nonce deklaracji skryptu zgadza  
się z wartością w polityce CSP")  
</script>  
  
<script src="https://example.com/ZEZWOLONY-ze-wzgledu-na-script-src.js"></script>  
<script nonce="EDNnf03nceI0fn39fn3e9h3sdaf"  
src="https://elsewhere.com/ZABLOKOWANY-ze-wzgledu-na-bledny-nonce.js"></script>  
<script nonce="Nc3n83cnSAd3wc3Sasdfn939hc3"  
src="https://elsewhere.com/DOZWOLONY-z-zewnetrznego-zrodla-ze-wzgledu-na-  
zgodosc-nonce.js"></script>
```

Warto zaznaczyć, że mechanizm „nonce” nie musi służyć tylko do tzw. walki ze skryptami i stylami „inline”. Ostatni przykład z listy powyżej pokazuje, w jaki sposób, odwołując się do pojedynczych skryptów z CDN, można utrzymać silną, zwięzłą politykę CSP.

Mechanizm „nonce” może zostać użyty także do oznaczenia skryptów i stylów ładowanych z miejsc niewyszczególnionych wśród hostów i ścieżek CSP.

Inline Hash (CSP 2.0)

Jest to kolejny pomysł (obok „nonce”) na wskazanie zaufanego skryptu lub stylu „inline”. W tym wypadku – przy pomocy jednokierunkowych funkcji skrótu.

Gdy do listy dozwolonych źródeł skryptów/stylów dodamy wartość 'sha256-\$base64hashInline' (pojedynczy cudzysłów), przeglądarka dopuści wykonanie kodu umieszczonego między tagami `<script></script>` (`<style></style>`), jeśli tylko hash SHA256 zawartości znaczników będzie równy wartości `$base64hashInline` (skrót musi być zakodowany algorytmem `base64`).

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



- Obsługiwane algorytmy jednokierunkowych funkcji skrótu to:
- » sha256,
 - » sha384,
 - » sha512.

Listing 3. przedstawia sposób użycia mechanizmu Inline Hash:

Listing 3. Przykład zastosowania Inline Hash.

```
<!-- Przykład CSP dla skrótu sha512 ciągu:
alert('Hello, world.');
-->
Content-Security-Policy: default-src 'self' script-src 'self' 'sha512-
Q2bFTOhEALkN8h0ms2FKTDLy7eugP2zFZ1T8LCvX42Fp3WoNr3bjZSAHeOsHrbV1Fu9/
A0EzCinRE7Af1ofPrw=='

<!-- Zasady zdefiniowane w CSP dopuszczą wykonanie poniższego kodu HTML: -->
<script>alert('Hello, world.');//</script>

<!-- Następne trzy wstawki zostaną zablokowane: -->
<script>alert(/XSS/);</script>

<script>
alert('Hello, world.');
</script>

<script> alert('Hello, world.');//</script>
```

Pamiętaj – podczas wyliczania skrótu przy użyciu jednokierunkowych funkcji skrótu białe znaki mają znaczenie.

Do wyliczenia skrótu ze wskazanego ciągu znaków można użyć poniższych poleceń:

```
echo -n "alert('Hello, world.');
| openssl dgst -sha256 -binary | openssl enc -base64

echo -n "alert('Hello, world.');
| openssl dgst -sha256 -binary | openssl enc -base64 -A > hash.txt
```

Drugim sposobem dostarczenia wartości skrótu do CSP jest przeczytanie opisu błędu naruszenia zasad CSP dla skryptu „inline” i skopiowanie proponowanej wartości do polityki, tak jak pokazano na poniższym rysunku:



Rysunek 4. Wartość jednokierunkowej funkcji skrótu dla skryptu "inline".

„COŚ SIĘ KOŃCZY, COŚ ZACZYNA...” – CSP 3.0?

Poniżej wymieniam kilka dyrektyw, które zostały zaimplementowane w niektórych przeglądarkach i mogą trafić jako element standardu CSP Level 3:

- » upgrade-insecure-requests – włącz mechanizm podmieniający wszystkie odwołania do zasobów HTTP do ich wersji HTTPS;
- » block-all-mixed-content – zabroni wczytywania zasobów przez niezaufane medium (HTTP), gdy cała strona została załadowana po HTTPS;
- » manifest-src – restrykcje dotyczące pliku manifestu mechanizmu „HTML5 Offline Web Applications”, o którym pisałem w artykule [o bezpieczeństwie HTML5](#) (przy braku definicji respektuje default-src);
- » referrer – zwiększa kontrolę nad danymi pojawiającymi się w nagłówku Referer, który może spowodować wysłanie wrażliwych danych do obcej domeny. Obecnie proponowane wartości tej dyrektywy to:
 - » no-referrer – nigdy nie wysyłaj nagłówka,
 - » no-referrer-when-downgrade – nie wysyłaj nagłówka, gdy nawigacja przechodzi ze strony HTTPS na HTTP (podczas nawigacji HTTPS>HTTPS, HTTP>HTTP oraz HTTP>HTTPS nagłówek będzie dodawany),
 - » origin – dodawaj tylko nazwę hosta w nagłówku referrer (chroni przed wyciekiem informacji znajdujących się w ścieżce oraz tzw. „query stringu”),
 - » origin-when-cross-origin – gdy nawigacja będzie w ramach tej samej domeny (niezależnie od protokołu), wtedy referrer będzie zawierał pełny URL; gdy nawigacja będzie się odbywać między różnymi domenami, referrer będzie zawierał tylko nazwę hosta,
 - » unsafe-url – wysyłaj pełny URL w nagłówku referrer (czyli brak restrykcji);
- » reflected-xss – celem dyrektywy jest zastąpienie niestandardowego nagłówka X-XSS-Protection, który instruuje przeglądarkę o włączeniu mechanizmów „Anti-Reflected-XSS” (czyli „XSS Auditor” w Google Chrome czy jego odpowiednik w Internet Explorer); możliwe wartości to:
 - » allow (wyłącz zabezpieczenie anti-xss),
 - » block (blokuj wykonywanie wykrytego skryptu),
 - » filter (postaraj się wyłącznie filtrować złośliwy skrypt).

Wysunięty Content Security Policy directive: "script-src test.php:10 unsafe-inline" keyword, a hash ('sha256-qznLcsR0x4GACP2dm0UCKCzCG+HiZ1guq6ZZDob/Tng='), on.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



CZY WARTO STOSOWAĆ CSP?

Content Security Policy jest szybko rozwijającą się, łatwą w zrozumieniu technologią, która skutecznie utrudnia życie crackerom. Pokazuje największą moc wówczas, gdy jest rozwijana w projektach od samego początku – rygorystyczne polityki znacznie zwiększą bezpieczeństwo, a wyższa jakość kodu zmniejsza ogólną liczbę błędów.

Nie ma jednak róży bez kolców. Pobieżne zrozumienie standardu lub zgubienie się w szczegółach dokumentacji, może unieruchomić działanie serwisu. Wdrożone zasady należy przetestować w najważniejszych przeglądarkach – najlepiej przy pomocy CSP w trybie raportowania.

CSP nigdy nie może zastąpić bezpiecznego kodu – nowe ograniczenia pomagają zmniejszać skutki ataków (takich jak XSS), ale nie są mechanizmami do zapobiegania im!

Aby uniknąć problemów z błędami Cross Site Scripting (XSS), programiści nie mogą poświęcić poprawnego kodowania znaków oraz mechanizmów walidacji na rzecz CSP!

Dużym problemem podczas wdrożenia CSP są również biblioteki Javascript implementujące tzw. „client-side MVC/MVVM”, które przestają działać przy rygorystycznych politykach. Z popularnych frameworków jedynie [AngularJS](#) od dawna może pochwalić się wsparciem CSP, jednak jest to raczej wyjątek potwierdzający regułę. Miejmy nadzieję, że konkurencja szybko pójdzie za przykładem frameworka firmy Google.

PODSUMOWANIE

Wiedza o Content Security Policy szybko rozpowszechnia się wśród programistów i testerów. Niestety pobieżne zrozumienie CSP może skutkować wprowadzaniem liberalnych polityk, tylko nieznacznie podnoszących poziom bezpieczeństwa. Zachęcam do dokładnej analizy tego standardu i wdrożenia go do projektów webowych, aby skutecznie utrudnić życie potencjalnym agresorom.

Content Security Policy staje się uniwersalnym narzędziem konfiguracyjnym załatwianiem bezpieczeństwa strony w przeglądarkach użytkowników. Z nadzieją czekam na kolejne wersje standardu oraz wdrożenia.

Adrian 'Vizzdoom' Michalczyk. Interdyscyplinarny bezpiecznik i geek.

Zakochany w dobrej fabule, roleplay'u i fotografii...

Strona domowa autora: <http://adrian.michalczyk.website/>



[Mechanizm Service Workers](#)

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Wirtualna poprawka (ang. *virtual patch*) to wdrażanie w warstwie zabezpieczeń mechanizmu umożliwiającego wykrycie i zablokowanie złośliwego kodu, zanim dotrze do chronionego zasobu.

Artykuł koncentruje się na atakach na aplikacje webowe i wykorzystaniu ModSecurity jako narzędzia do wprowadzania wirtualnych poprawek. ModSecurity to oprogramowanie typu Web Application Firewall (WAF), który działa jako reverse proxy dla protokołu http i jest oprogramowaniem opensource, z możliwością budowy własnych reguł – zatem jest to narzędzie, które umożliwia realizację wirtualnego "patchowania".

Z artykułu dowiesz się:

- » co to jest wirtualne "patchowanie",
- » jak działa ModSecurity,
- » jak zainstalować i skonfigurować ModSecurity,
- » jak "czytać" i pisać własne reguły,
- » fazy procedury wirtualnego „patchowania”,
- » piszemy wirtualne poprawki pod różne podatności,
- » jak zobrazować dane z logów: Splunk i ModSecurity.

PO CO STOSUJE SIĘ WIRTUALNE POPRAWKI?

Wyobraźmy sobie sytuację, gdy ujawniony zostaje kod exploitu, który jest w stanie zaatakować naszą aplikację. Informacja taka mogłaby na przykład pochodzić od zespołu CERT, z własnych systemów honeypot lub z serwisów publikujących informacje o zagrożeniach sieciowych. Producent danego oprogramowania nie przygotował jeszcze nowej wersji oprogramowania, odpornej na działanie exploita. Mamy tu do czynienia z klasycznym przykładem exploitu typu 0-day, gdzie moment ujawnienia podatności jest różny od wydania oficjalnej łatki. W sytuacji, gdy procedury aktualizacji oprogramowania wymagają przeprowadzenia żmudnych testów, czas niezbędny do zaaplikowania tej poprawki może się znacznie wydłużyć. Zespół odpowiedzialny za zagadnienia ochrony aplikacji webowych powinien w takiej sytuacji skorzystać z możliwości, jakie oferują wirtualne poprawki, ponieważ cechują się one:

- » szybkim czasem reakcji,
- » brakiem konieczności modyfikacji kodu aplikacji i związanego z tym ryzyka,
- » prostotą pisania reguł,

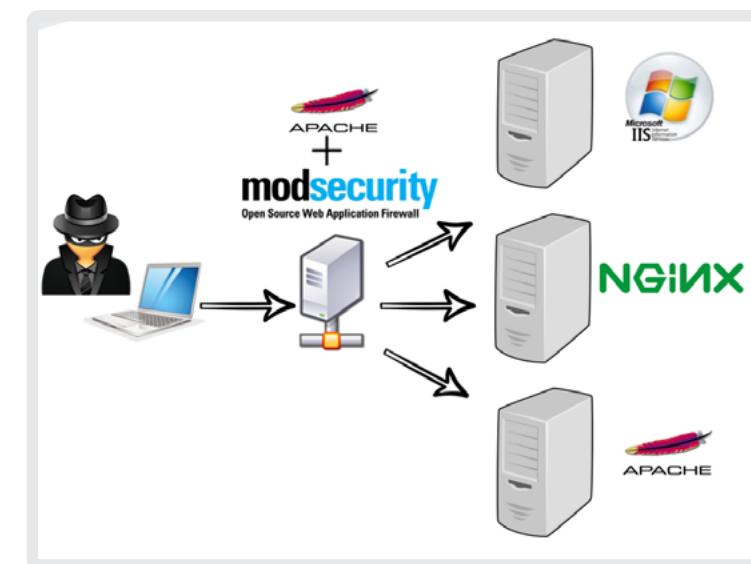
- » skalowalnością, dzięki ochronie szeregu serwerów WWW w jednym miejscu,
- » ochroną systemów, które nie mogą być restartowane w ramach aktualizacji oprogramowania,
- » zapewnianiem bezpieczeństwa starszym aplikacjom, które nie są już rozwijane.

Jak widać, wirtualne poprawki mają wiele zalet, jednak należy mieć stale na względzie, że bezpieczeństwo informatyczne to proces wielopłaszczyznowy, nie należy więc polegać wyłącznie na wirtualnym patchowaniu jako alternatywie dla aktualizacji oprogramowania w celu redukcji kosztów.

JAK DZIAŁA MODSECURITY?

ModSecurity jako moduł serwera WWW (tu Apache, ale możliwa jest też integracja z NGINX i IIS) jest w stanie przechwytywać i analizować zapytania/odpowiedzi, pracując w dwóch trybach z punktu widzenia architektury bezpieczeństwa:

- » single-server (host-based) – chroniąc pojedynczy serwer, na którym ten moduł jest doinstalowany,
- » dedykowany serwer reverse-proxy protokołu HTTP, czyli serwer, który przyjmuje wszystkie zapytania HTTP, przetwarza je i następnie przesyła do końcowych serwerów aplikacji webowej.



Rysunek 1. Poglądowy schemat działania reverse proxy z ModSecurity.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



ModSecurity może również działać w dwóch podstawowych trybach z punktu widzenia powstrzymywania ataków:

- » w trybie detekcji, czyli wykrywania ataków i ich logowania (optymalny tryb w czasie rozpoznania aplikacji i dostosowywania zbioru reguł tak, by nie blokowały normalnego ruchu, który w pewnych okolicznościach może zostać zidentyfikowany jako atak i zablokowany),
- » w trybie prevencji, czyli blokowania ataków i zapobiegania wyciekom danych.

ModSecurity „rozumie” protokół HTTP, a poszczególne zapytania i odpowiedzi łączy w odpowiadające sobie transakcje, dlatego też jest w stanie filtrować złośliwą aktywność na różnych etapach ataku. W przetwarzaniu reguł można rozróżnić pięć faz:
1. Request headers (phase:1) – faza po wczytaniu nagłówków zapytania,
2. Request body (phase:2) – faza po wczytaniu całego zapytania (większość reguł jest przetwarzana w tej fazie),
3. Response headers (phase:3) – faza przed wysłaniem nagłówków odpowiedzi,
4. Response body (phase:4) – faza przed wysłaniem ciała odpowiedzi (w ten sposób można powstrzymać wyciek danych),
5. Logging (phase:5) – faza po wysłaniu odpowiedzi, przed zalogowaniem (wyłącznie możliwość kontroli procesu logowania zdarzeń).

W konfiguracji ModSecurity definiuje się domyślną akcję, jaką nasz WAF będzie realizował w przypadku, gdy reguła zostanie dopasowana do danego zapytania lub odpowiedzi HTTP, przykładowo:

```
SecDefaultAction „phase:2, deny, log, status:403”
```

Co należy interpretować: gdy zostanie wczytane zapytanie w fazie drugiej (nie tylko HEADER, ale też BODY) i zostanie dopasowane do którejś z reguł SecRule, ModSecurity odrzuci takie zapytanie, zapisze zdarzenie do logu, a klientowi zwróci odpowiedź z kodem 403 – *Forbidden*. Zastosowanie akcji *drop* zamiast *deny* spowoduje wysłanie pakietu TCP z flagą *F/N*, co może być bardziej skuteczne w przypadku ataków *Denial of Service*.

Ingerencja w ruch HTTP w czasie każdej z wyżej wymienionych faz, pozwala na bardzo elastyczne sterowanie akcjami w ramach implementowania własnych reguł. Jeśli chcemy, aby dana reguła została sprawdzona w innej fazie niż określona w domyślnej akcji, to dopisujemy numer fazy (*phase*) np.:

```
SecRule REMOTE_ADDR „^192\.168\.” „phase:1, pass, log, msg: 'Podejrzany ruch IP'”
```

Powyższa reguła zostanie sprawdzona w pierwszej fazie, zapytanie będzie przekazane do następnej fazy, ale jeśli będzie wysłane z adresu IP rozpoczynającego się od „192.168” to zdarzenie to zostanie zapisane w logu.

ModSecurity przetwarza każde zapytanie w cyklu pięciu faz. W ramach każdej z nich reguły są sprawdzane w kolejności, w jakiej są wpisane w pliku konfiguracyjnym reguły, a ModSecurity wybiera tylko te reguły, które odpowiadają danej fazie. Zapytanie, które zostało przepuszczone w ramach fazy 1, może zostać odrzucone na późniejszych etapach – np. po analizie parametrów zapytania POST, co odpowiada fazie 2.

Oprócz tworzenia własnych reguł, ModSecurity posiada możliwość dołączania do konfiguracji predefiniowanych reguł przygotowanych na znane ataki webowe. Reguły te można pobrać ze stron:

- » <https://github.com/SpiderLabs/owasp-modsecurity-crs> [darmowe]
- » <https://ssl.trustwave.com/web-application-firewall> [płatne]
- » <https://www.atomicorp.com/amember/cart/index/index?c=6> [płatne]

INSTALACJA I KONFIGURACJA MODSECURITY

1. Instalacja

ModSecurity można zainstalować jako moduł serwera WWW (tu przykład dla serwera Apache w dystrybucji Debian):

Listing 1. Instalacja ModSecurity za pomocą apt. Uwaga: może być wymagane włączenie modułu.

```
$ sudo apt-get install libapache2-mod-security2
Czytanie list pakietów... Gotowe
Budowanie drzewa zależności
Odczyt informacji o stanie... Gotowe
Zostań zainstalowane następujące dodatkowe pakiety:
  modsecurity-crs
Sugerowane pakiety:
  lua geoip-database-contrib
Zostań zainstalowane następujące NOWE pakiety:
  libapache2-mod-security2 modsecurity-crs
$ sudo a2enmod mod-security
```

2. Konfiguracja

Pliki konfiguracyjne umieszczone są w trzech lokalizacjach:

- » </etc/modsecurity/modsecurity.conf>

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



- » /etc/apache2/mods-enabled/security2.conf
- » /usr/share/modsecurity-crs/

Podstawowa konfiguracja silnika ModSecurity znajduje się w pliku *modsecurity.conf*, a wraz z modułem zostaje doinstalowany dodatkowy pakiet reguł Core Rule Set.

Proces konfiguracji warto rozpoczęć od zalecanych ustawień, które znajdują się w pliku z rozszerzeniem *recommended*:

```
$ sudo mv /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf
```

W pliku tym decydujemy o podstawowych parametrach pracy ModSecurity, np. trybie detekcji lub prevencji:

```
SecRuleEngine DetectionOnly | On | Off
```

Zbiór reguł **ModSecurity CRS** zlokalizowany jest w */usr/share/modsecurity-crs/*

```
$ ls /usr/share/modsecurity-crs/  
activated_rules experimental_rules modsecurity_crs_10_setup.conf slr_rules  
base_rules lua optional_rules util
```

Reguły te dodajemy wpisami do pliku konfiguracyjnego modułu *security2.conf*

```
<IfModule security2_module>  
IncludeOptional /etc/modsecurity/*.conf  
IncludeOptional "/usr/share/modsecurity-crs/*.conf"  
IncludeOptional "/usr/share/modsecurity-crs/activated_rules/*.conf"  
</IfModule >
```

Reguły z katalogu *base_rules* należy przejrzeć i dołączyć do konfiguracji przez tworzenie linków symbolicznych w katalogu *activated_rules*:

Listing 2. Lista reguł ModSecurity Core Rule Set.

```
$ cd /usr/share/modsecurity-crs/activated_rules  
$ ls ..../base_rules/  
modsecurity_35_bad_robots.data  
modsecurity_35_scanners.data  
modsecurity_40_generic_attacks.data  
modsecurity_50_outbound.data  
modsecurity_50_outbound_malware.data
```

```
modsecurity_crs_20_protocolViolations.conf  
modsecurity_crs_21_protocolAnomalies.conf  
modsecurity_crs_23_requestLimits.conf  
modsecurity_crs_30_httpPolicy.conf  
modsecurity_crs_35_badRobots.conf  
modsecurity_crs_40_genericAttacks.conf  
modsecurity_crs_41_sqlInjectionAttacks.conf  
modsecurity_crs_41_xssAttacks.conf  
modsecurity_crs_42_tightSecurity.conf  
modsecurity_crs_45_trojans.conf  
modsecurity_crs_47_commonExceptions.conf  
modsecurity_crs_48_localExceptions.conf.example  
modsecurity_crs_49_inboundBlocking.conf  
modsecurity_crs_50_outbound.conf  
modsecurity_crs_59_outboundBlocking.conf  
modsecurity_crs_60_correlation.conf
```

```
$ sudo ln -s ..../base_rules/modsecurity_crs_40_generic_attacks.conf
```

Po dołączeniu wszystkich niezbędnych plików konfiguracyjnych – w tym plików reguł – wymagany jest restart serwera:

```
$ sudo service apache2 restart
```

3. Inne czynności związane z obsługą reguł

Aktualizacja reguł ModSecurity-CRS odbywa się poprzez *apt-get upgrade* lub aktualizację pojedynczego pakietu, w tym wypadku:

```
$ sudo apt-get install modsecurity-crs --only-upgrade
```

W zależności od specyfiki aplikacji, część reguł może generować fałszywe alarmy i reguły te powinny być **wyłączane** albo zmodyfikowane. W celu wyłączania reguł warto skorzystać z ich numerów *id*. Możemy wyłączać je globalnie, dla konkretnych wirtualnych hostów lub lokalizacji w aplikacji.

```
<LocationMatch "/wordpress/wp-login.php">  
  <IfModule security2_module>  
    SecRuleRemoveById 981134  
  </IfModule>  
</LocationMatch>
```

Aby pisać **własne reguły**, należy przygotować osobny plik, który nie będzie nadpisywany w czasie aktualizacji pakietów.

```
$ sudo vi /etc/modsecurity/modsecurity_custom_rules.conf
```

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



"CZYTANIE" I PISANIE WŁASNYCH REGUŁ

Czytanie i pisanie reguł ModSecurity jest bardzo proste. Silnik WAF oferuje również kilka rozwiązań, które pomagają elastycznie pisać reguły tak, by mogły skutecznie powstrzymywać różne rodzaje ataków webowych, są to:

- » zmienne i kolekcje,
- » transakcje,
- » łańcuchy reguł,
- » wyrażenia regularne,
- » transformacje,
- » dopasowywanie wzorców (z plików).

Poniższe przykłady powinny jednoznacznie przedstawić, w jaki sposób należy stosować wymienione funkcje w czasie definiowania reguł:

1. Składnia reguły

SecRule Target Operator [Transformation Action]

- » **Target** – określa część zapytania lub odpowiedzi, która jest analizowana, np. `REQUEST_HEADERS:User_Agent`, `REQUEST_URI`, `REMOTE_ADDR`.
- » **Operator** – określa metodę, jaką jest używana do porównania zmiennej z wartościami w `Target`, domyślnie jest to `@rx`, czyli zastosowanie wyrażeń regularnych.
- » **Transformation** – określa, jakimi transformacjami powinno być znormalizowane zapytanie lub odpowiedź, np. odkodowanie base64, zamiana znaków na małe litery itp.
- » **Action** – pole opcjonalne, jeśli chcemy użyć innego działania niż określone w `SecDefaultAction`.
- » Zmienne i kolekcje, czyli pojedyncze wartości lub ich zbiory, które chcemy analizować (pole `Target`).

Przykłady: `ARGS` – kolekcja zawierająca wszystkie parametry zapytania GET i POST, np.

SecRule ARGS:username|ARGS:password „admin”

Reguła sprawdza, czy w parametrach zapytania, takich jak `username` lub `password`, przekazywane są wartości „`admin`”.

Inne często stosowane kolekcje: `ENV`, `FILES`, `GEO`, `IP`, `REQUEST_*`, `SESSION`, `USER`.

2. Transakcje

Istnieje specjalny rodzaj kolekcji `TX` pozwalający na definiowanie własnych zmiennych i na przechowywanie w nich danych. Często stosowane jest to do określania poziomu detekcji dla danego typu zagrożeń, tu chcemy wykryć podejrzane manipulacje parametrem `id`:

```
SecRule ARGS:id "@gt 5" "pass, t:none,t:length, tx.anomaly_score+=5"
SecRule ARGS:id "!@rx ^[0-9]*$" "pass, t:none,t:length, tx.anomaly_score+=5"
SecRule TX:anomaly_score „@gt 5” „deny,log,msg: 'Podejrzaną wartość parametru id' - znaki inne niż cyfry i więcej znaków niż 5”
```

3. Łańcuchy reguł

W wielu przypadkach – gdy chcemy budować reguły, w których musi zostać spełnione kilka warunków jednocześnie, należy użyć operatora `chain`. Ostatnia reguła w łańcuchu nie posiada już operatora `chain`. Dla przejrzystości zapisu, warto stosować wcięcia tekstu dla połączonych ze sobą reguł.

```
SecRule REQUEST_URI „/admin” „chain, deny, msg: 'Próba logowania do panelu administratora z nieprzeznaczonych do tego adresów IP, po godzinie 16’”
SecRule REMOTE_ADDR „!@rx ^192\.168\.1\.[10-20]” „chain”
SecRule TIME_HOUR „@gt 16”
```

4. Wyrażenia regularne

ModSecurity domyślnie stosuje wyrażenia regularne (@rx) w celu dopasowania reguł. Implementacja tych wyrażeń odpowiada składni języka Perl. Inne sposoby dopasowania reguł do zawartości zapytań/odpowiedzi HTTP to stosowanie dopasowania łańcuchów znaków (bardziej precyzyjne, ale mniej uniwersalne rozwiązanie), a do dyspozycji twórcy przewidzieli następujące operatory: `@beginsWith`, `@contains`, `@containsWord`, `@endsWith`, `@streq`, `@within`.

```
SecRule REMOTE_HOST „@endsWith attacker.pwn” „deny, msg: 'Blokowanie zapytań od hostów z domeną attacker.pwn’”
```

5. Transformacje

Specyfika protokołu HTTP powoduje, że te same efekty można osiągnąć, używając różnych wartości w zapytaniach. Fakt ten jest podstawą stosowania przez atakujących różnego rodzaju technik omijających filtry. Najprostsze sposoby, to przesłanie wartości parametrów zapisanych wielkimi literami np. `SCRIPT` zamiast `script`.

Bardziej zaawansowane techniki to dodawanie znaków ucieczki „\”, kodowanie base64 lub URL, dodawanie białych znaków lub komentarzy, np. `SEL/**/ECT`.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



W celu normalizacji wartości z zapytań HTTP, w ModSecurity stosowane są funkcje transformujące definiowane w polu *Action* i poprzedzone operatorem „*t*:”. W tym polu można zastosować kilka takich funkcji transformujących zapytanie.

```
SecRule ARGS "@rx (?i)^php://" \ "phase:2,t:none, t:base64Decode, t:lowercase,msg:'Wstrzygnięcie zakodowanego w base64 PHP streams'"
```

Użycie *t:none* czyści wcześniejsze funkcje transformujące.

6. Dopasowanie wzorców

Dla zwiększenia wydajności przetwarzania reguł, których zapytanie musi być porównane z dużą liczbą wzorców, warto stosować *pattern matching* – operator @*pm* lub @*pmFromFile* (wzorce zapisane są w osobnym pliku).

```
SecRule REQUEST_HEADERS:User_Agent „@pmFromFile /usr/share/modsecurity_extras/known_web_scanners.txt” „deny, msg: 'blokowanie webowych skanerów podatności na podstawie wartości User_Agent z nagłówka'"
```

FAZY PROCEDURY WIRTUALNYCH POPRAWEK

Na podstawie zasad określonych przez OWASP wyróżnia się sześć faz procedury wirtualnego „patchowania”, są to:

1. przygotowanie,
2. identyfikacja,
3. analiza,
4. przygotowania wirtualnej poprawki,
5. wdrożenie/testy,
6. monitorowanie.

W poszczególnych fazach realizowane są poniższe czynności.

1. Przygotowanie

Faza przygotowania obejmuje następujące zagadnienia:

- » budowa infrastruktury umożliwiającej wdrażanie wirtualnych poprawek,
- » uzyskanie zgody na natychmiastowe stosowanie wirtualnych poprawek poza reżimem testów, będących częścią procesu aktualizacji oprogramowania. Testy takie są zwykle rozciągnięte w czasie,
- » monitorowanie list mailingowych oprogramowania podlegającego ochronie,

- » zapewnienie dostępu do pełnych informacji o zapytaniach i odpowiedziach ruchu HTTP – podstawowe dane zawarte w *access.log* stanowią niewystarczające źródło do ochrony aplikacji webowych.

2. Identyfikacja

Faza identyfikacji związana jest z poszukiwaniem i uzyskaniem informacji o podatności występującej w używanej aplikacji webowej. Rozróżnia się dwa podejścia:

- » **proaktywne**, gdy organizacja poszukuje podatności za pomocą testów penetracyjnych i/lub przeglądów bezpieczeństwa kodów źródłowych,
- » **reaktywne**, gdy organizacja o podatności dowiaduje się przez bezpośredni kontakt z producentem, publiczne ujawnienie podatności (*public disclosure*) i najbardziej krytycznie – przez wykrycie incydentu bezpieczeństwa z trwającego ataku.

3. Analiza

W czasie analizy należy:

- » określić, czy wirtualna poprawka jest odpowiednim remedium na daną podatność,
- » jednoznacznie zidentyfikować podatność po nazwie (CVE lub nadać własną nazwę wewnętrzna w organizacji),
- » określić poziom krytyczności dla tej podatności, ponieważ przydziel sił i środków powinien być zorientowany na ryzyko, jakie generuje podatność,
- » określić, jakie wersje oprogramowania są podatne i jakie warunki muszą być spełnione, aby podatność była „exploitowalna”, np. czy atakujący musi mieć dostęp zdalny/lokalny, być uwierzytelniony/nieuwierzytelniony,
- » zgromadzić listę payloadów i kodu exploitów typu Proof of Concept,
- » uzupełnić informacje o podatności w systemie śledzenia błędów, np. Jira, Threadfix.

4. Przygotowania wirtualnej poprawki

Wirtualna poprawka musi spełniać **dwie podstawowe reguły**:

- » brak **false positives** czyli fałszywych alarmów,
- » brak **false negatives**, czyli niewykrycia rzeczywistych ataków.

Wirtualne poprawki mogą być budowane ręcznie zgodnie z dwoma modelami:

- » **positive security** – białe listy dopuszczonych parametrów komunikacji (zbiór znaków, długość ciągów znakowych) i blokowanie wszystkich odstępstw od tych reguł,
- » **negative security** – czarne listy zabronionych parametrów, np. użycie znaków

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



specjalnych składni języka SQL, takich jak `< ', " , - , # >`. Takie podejście jest bardziej zbliżone do przygotowania sygnatur pod charakterystyczne ataki webowe.

Wirtualne poprawki mogą być również generowane automatycznie na podstawie raportów o podatnościach w formacie XML za pomocą:

- » skryptu ze zbioru reguł **OWASP ModSecurity Core Rule Set** ([arachni2modsec.pl](#), [zap2modsec.pl](#)),
- » narzędzi do śledzenia informacji o błędach np. **Threadfix**,
- » bezpośrednio, z wykorzystaniem firewalli aplikacyjnych, np. poprzez integrację skanera podatności [Arachni i WAF ModSecurity](#),
- » **usługi** do wirtualnych poprawek **Virtual Patching On-Demand Service**, gdzie można importować dane z 11 najczęściej stosowanych narzędzi typu DAST.

5. Wdrożenia i testy

Po zaimplementowaniu reguły ModSecurity mającej wirtualnie załatać podatność w aplikacji webowej, należy przetestować, czy WAF poprawnie loguje (ale jeszcze nie blokuje, aby uniknąć fałszywych alarmów) ataki za pomocą znanych payloadów oraz wykorzystując techniki omijania filtrów WAF/IDS. Stosowane mogą być następujące narzędzia:

- » przeglądarka internetowa,
- » narzędzia wiersza poleceń: **wget**, **curl**,
- » HTTP proxy, np. **Burp Suite**, **OWASP ZAP**,
- » ModSecurity AuditViewer – aplikacja JAVA do analizy **audit log ModSecurity**.

6. Monitorowanie

Ostatnia faza cyklu wirtualnej poprawki to nadzór. W ramach tej fazy należy zaktualizować informacje w systemie śledzenia błędów, a w szczególności zgłosić potrzebę załatwania podatności w kodzie źródłowym. Po wprowadzeniu nowej, poprawionej wersji oprogramowania można wycofać wirtualną poprawkę z WAF lub zachować ją w celu lepszej agregacji informacji bezpieczeństwa o próbach wykorzystania tej podatności i przygotowania odpowiednich raportów na temat skuteczności działań obronnych – za pomocą procedury wirtualnego „patchowania”.

WIRTUALNE POPRAWKI POD RÓŻNE PODATNOŚCI

Poniżej zostaną przedstawione scenariusze przygotowania wirtualnych poprawek dla podatnych aplikacji.

Scenariusz #1

Podatność

Atak Blind SQLi przez kontrolę parametru, względem którego sortowane są kolumny w zapytaniu bazodanowym (wyrażenie ORDER BY).

Podatność ta nagminnie występuje w wielu różnych pluginach do Wordpressa.

Lokalizacja podatności

Podatny jest parametr `orderby` w zapytaniu GET, przykład plugin **SEO by YEST** wersja 1.7.3.3 (disclosure: 11.03.2015). Pomimo zastosowania prób filtracji – złe użycie funkcji `esc_sql()` – atakujący jest w stanie wykonać atak Blind SQL injection:

Listing 3. Kod podatnej klasy `class-bulk-editor-list-table.php`.

```
$orderby = ! empty( $_GET['orderby'] ) ? esc_sql( sanitize_text_field( $_GET['orderby'] ) ) : 'post_title';

$orderby = $this->sanitize_orderby( $orderby );

(...)

$query = "
    SELECT ID, post_title, post_type, post_status, post_modified, post_date
    FROM {$subquery}
    WHERE post_status IN ({$all_states}) $post_type_clause
    ORDER BY {$orderby} {$order}
    LIMIT %d,%d
```

Źródło: <https://wpvulndb.com/vulnerabilities/7841>.

Atak

Po wartości parametru `orderby=post_date` (sortowanie po dacie postu), atakujący jest w stanie przez dodanie znaku przecinka „,” (%2C w notacji URL) dołączyć payload BlindSQLi, który zostanie przekazany do bazy danych.

```
&orderby=post_date%2C(select+*+from+(select(sleep(3)))a)
```

Obrona

Odpowiednia filtracja danych przekazywanych w parametrze `orderby` jest w stanie powstrzymać wstrzyknięcie kodu i kontrolę nad zapytaniem bazodanowym. W tym przykładzie zastosujemy transformacje: dekodowania znaków w notacji URL do filtrowania znaku przecinka oraz łańcuch reguł, aby jednoznacznie zlokalizować podatne miejsce w aplikacji.

Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



Wirtualna poprawka

```
SecRule REQUEST_FILENAME "/admin.php" "phase:2, t:none, t:normalizePath,
t:lowercase, t:urlDecodeUni, chain, deny, log, id:1001"
SecRule ARGS_GET:orderby "," "t:urlDecode"
```

Rysunek 2. Powstrzymanie przez ModSecurity ataku BlindSQLi.

Alternatywnie można zastosować podejście **positive security**, w którym tylko dopuszczone nazwy kolumn mogłyby być przekazane do zmiennej orderby i druga część reguły wyglądałaby następująco:

```
SecRule ARGS_GET:orderby "!^((post_date)|(post_type)|(post_title)) "t:none"
```

Scenariusz #2

Podatność

Tylna furtka umożliwiająca **upload PHP shell** w pluginie Premium SEO Pack 1.8.0 do Wordpressa (disclosure: 24.04.2015).

Lokalizacja podatności

Klasa **abmRemoteSupport** umożliwia udzielenie zdalnej pomocy klientom, jednakże wymagany do uwierzytelnienia „klucz serwisowy” jest na stałe zapisany w kodzie źródłowym dodatku.

```
private function load_config(){
    require_once( 'remote_init.php' );
    $this->config = $aa_tunnel_config;

    /* in remote_init.php
     * $aa_tunnel_config = array(
     *   "key" -> "69efc4922575861f31125878597e97cf",
     * );
    */

}

private function validate_connection(){
    $coming_key = isset($_REQUEST['connection_key']) ? $_REQUEST['connection_key'] : '';
    if( trim($coming_key) == "" || $coming_key != $this->config['key']){
        $this->print_error( array('code' => 101, 'msg' => "Invalid key!"), 'fatal' );
    }
    return true;
}
```

Rysunek 3. Fragment kodu klasy abmRemoteSupport z zaszytym „backdoorem” Atak RemoteKey. Źródło: <https://wpvulndb.com/vulnerabilities/7934>.

Atak

Atakujący po analizie kodu źródłowego pluginu Premium SEO Pack 1.8.0, jest w stanie przygotować zapytanie POST odwołujące się do zasobu **plugins/premium-seo-pack/modules/remote_support/remote_tunnel.php**, podając w parametrze **connection_key** wartość **69efc4922575861f31125878597e97cf**, a w pozostałych parametrach – przykładowe wartości przedstawione na poniższym zrzucie.

Rysunek 4. Umieszczenie PHP shell na serwerze przez ‘tylną furtkę’ w oprogramowaniu.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

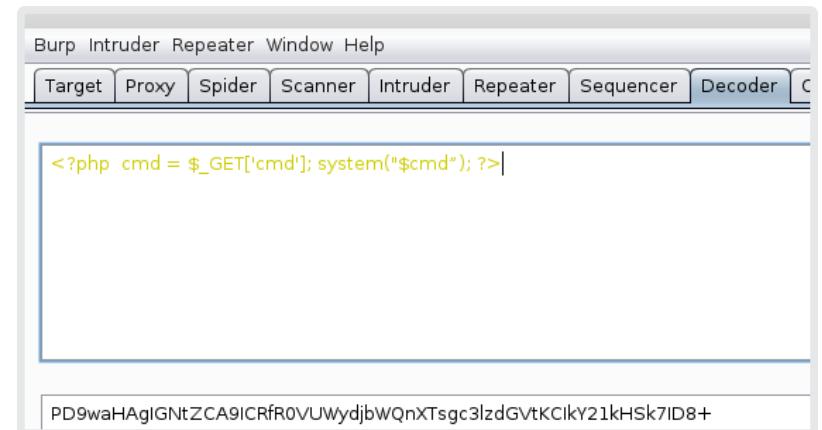
Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



Po przesłaniu takiego requestu POST atakujący umieścił własny skrypt typu PHP shell, zakodowany w base64, i jest w stanie wykonywać zdalnie polecenia systemu operacyjnego z prawami właściciela procesu serwera webowego (RCE – remote code execution).



Rysunek 5. PHP shell zakodowany w base64.

Obrona

Ponieważ SEO niestety często ma znacznie wyższy priorytet biznesowy niż bezpieczeństwo, często również nie ma możliwości wycofania pluginu lub zgody na modyfikację jego kodu. Rozwiążaniem pozostaje wirtualne załatwianie przez blokadę możliwości udzielenia „zdalnej pomocy” przez podmioty zewnętrzne.

Wirtualna poprawka

Zastosowane zostanie podejście **negative security** – blokowane będą zapytania POST do zasobu **remote_tunnel.php** zawierające ciąg znaków klucza serwisowego **connection_key**. Podejście to zapewni zgodność z zasadami, według których nie może dojść zarówno do fałszywego alarmu, jak i fałszywego odrzucenia.

```
SecRule REQUEST_FILENAME "/remote_tunnel.php" "phase:2, t:none, t:normalizePath, t:lowercase, t:urlDecodeUni, chain, deny, log, id:1002"
SecRule ARGS_POST:connection_key "69efc4922575861f31125878597e97cf" "t:none"
Fragment logu modsec_audit.log
--a1521748-H--
Message: Access denied with code 403 (phase 2). Pattern match
"69efc4922575861f31125878597e97cf" at ARGS_POST:connection_key. [file "/etc/
modsecurity/modsecurity_custom_rules.conf"] [line "4"] [id "1002"]
Action: Intercepted (phase 2)
```

Scenariusz #3

Podatność

Aplikacja webowa jest podatna na wstrzyknięcie polecenia systemu operacyjnego, na którym jest uruchomiona – OS injection, brak jest odpowiedniego filtrowania danych wejściowych do aplikacji.

Lokalizacja podatności

Podatność występuje w lokalizacji **/webapp/analyse**, a podatny jest parametr **input** w zapytaniu POST. Brak kodów źródłowych, kod napisany w języku Python został skompilowany. Wartości parametru input są kodowane **base64**. Z analizy aplikacji webowej wynika, że parametr ten do poprawnego działania aplikacji powinien przyjmować wyłącznie znaki alfanumeryczne do [a–z] lub [A–Z].

Atak

Atakujący za pomocą znaków kontrolnych wiersza poleceń w systemie LINUX <` & | > jest w stanie przekazać do aplikacji komendę, która zostanie wykonana z prawami właściciela procesu serwera WWW. Podatność ta może posłużyć do umieszczenia tylnej furtki lub eskalacji uprawnień do użytkownika uprzywilejowanego, i w konsekwencji – pełnego przejęcia serwera.

```
&input=c2VrdXJha1ppbmU=
base64decode(„c2VrdXJha1ppbmU=”) == sekurakZine

&input=c2VrdXJha1ppbmV8aWQ=
base64decode(„c2VrdXJha1ppbmV8aWQ=”) == sekurakZine|id
```

Obrona

Po przeprowadzonym teście penetracyjnym stwierdzono występowanie opisanej podatności. Aplikacja jest częścią systemu krytycznego i nie może być wyłączona ani zaktualizowana bez odpowiednich testów poprawek kodu, co jednak może narazić aplikację na poważne w skutkach ataki typu RCE.

Wirtualna poprawka

Zastosowane zostanie podejście **positive security**, zgodnie z którym ModSecurity będzie dopuszczał wyłącznie znaki alfanumeryczne z ograniczonego zbioru, co nie pozwoli przekazać znaków umożliwiających wstrzyknięcie komend wiersza poleceń LINUX. Jednak, z uwagi na kodowanie parametrów za pomocą **base64**, należy

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning

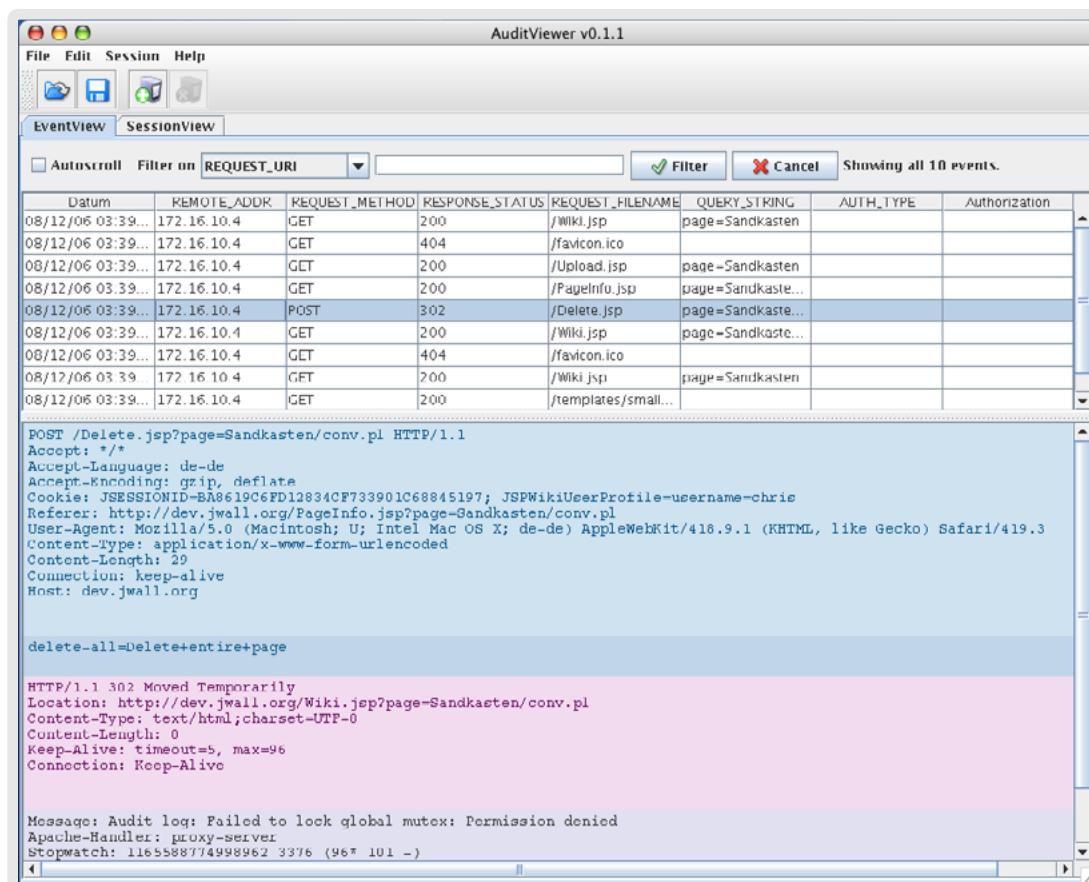


```
Request
Raw Params Headers Hex
POST /webapp/analyse HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0 Iceweasel/38.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1/poc.php
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 26

=c2VrdXJhalppbmV8aWQ=
```

Request	Response
<code></p> <hr> <address>Apache/2.4.10 (Debian) Server at 127.0.0.1 Port 80</address> </body></html></code>	<code>HTTP/1.1 403 Forbidden Date: Mon, 22 Feb 2016 21:33:36 GMT Server: Apache/2.4.10 (Debian) Content-Type: text/html; charset=iso-8859-1 Action: Intercepted (phase 2) Stopwatch: 1456176816180809 1171 [-1 -]//IETFF/DTD HTML 2.0/JEN* Stopwatch2: 1456176816180809 1171: combined=122, pl=61, p2=53, p3=0, p4=0, p5=8, s=0, r=0, sw=0, l=0, gc=0 t[le]=403 Forbidden/tile> Response-Body-Transformed: Dechunked Producer: ModSecurity for Apache/2.8.0 (http://www.modsecurity.org/) Server: Apache/2.4.10 (Debian) X-Powered-By: PHP/7.0.12 mod_fcgid by mod_fcgid/2.3.9 PHP/7.0.12 Engine-Mode: "ENABLED" is server.
 </p> <hr> <address>Apache/2.4.10 (Debian) Server at 127.0.0.1 Port 80</address></code>

Rysunek 6. Zablokowanie ataku OS command injection.



Rysunek 7. Dane z logu modsec_audit.log zobrazowane w Audit Viewer.
 Źródło: <https://jwall.org/web/audit/viewer.jsp>.

wcześniej znormalizować zapytanie, do czego służy transformacja `base64decode` na poziomie ModSecurity. Dopiero po odkodowaniu zapytania WAF analizuje, czy przekazane w parametrze `input` dane są zgodne z białą listą dopuszczonych znaków.

SecRule REQUEST_URI "/webapp/analyse" "phase:2, t:none, t:normalizePath, t:lowercase, t:urlDecodeUni, chain, deny, log, id:1003" (rysunek 6).

```
SecRule ARGS_POST:input "!@rx ^[a-zA-Z]*$" "t:base64decode"
```

JAK ZOBRAZOWAĆ DANE Z LOGÓW: SPLUNK I MODSECURITY

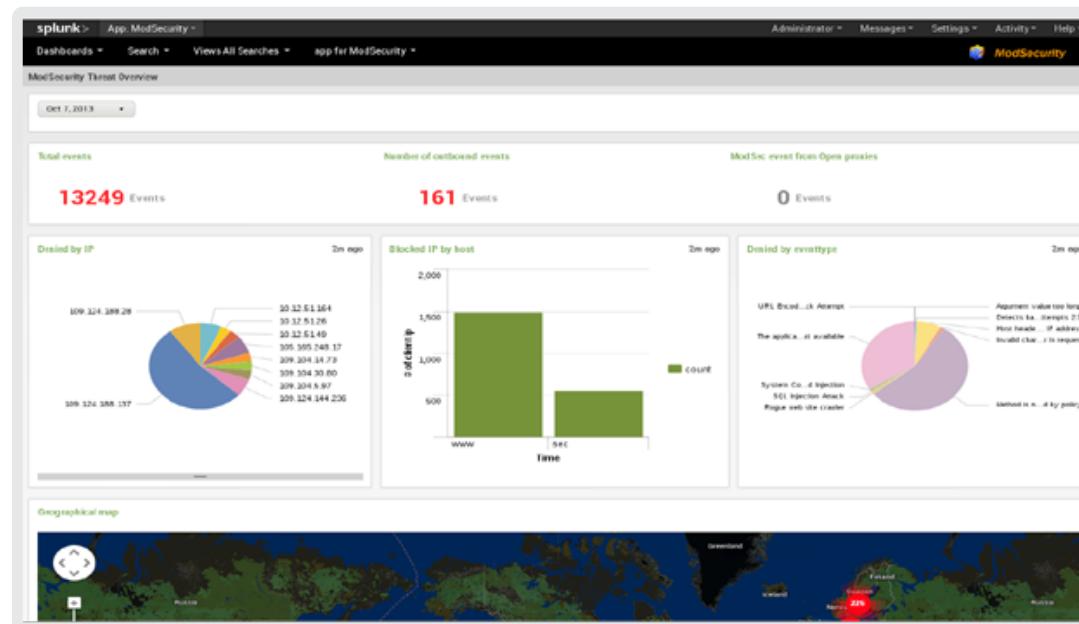
Log ModSecurity *modsec_audit.log* to kopalnia wiedzy, jednak łatwo też popaść w efekt przytłoczenia informacją.

Warto skorzystać z kilku rozwiązań, które pozwolą odfiltrować z tego logu niezbędne informacje o zdarzeniach bezpieczeństwa.

AuditViewer – aplikacja desktopowa (JAVA) do analizy logów, a także do ponownego wysyłania zmodyfikowanych zapytań w celu testowania zbioru reguł (rysunek 7).

ModSecurity for Splunk – dodatek do Splunk prezentujący dane w formie tabelarycznej i graficznej (rysunek 8).

Remo Rule editor – edytor reguł, projekt z 2007 (nierozwijany), rysunek 9.



Rysunek 8. Aplikacja ModSecurity jako dodatek do Splunk. Źródło: <https://splunkbase.splunk.com/app/880/>.

Mechanism Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

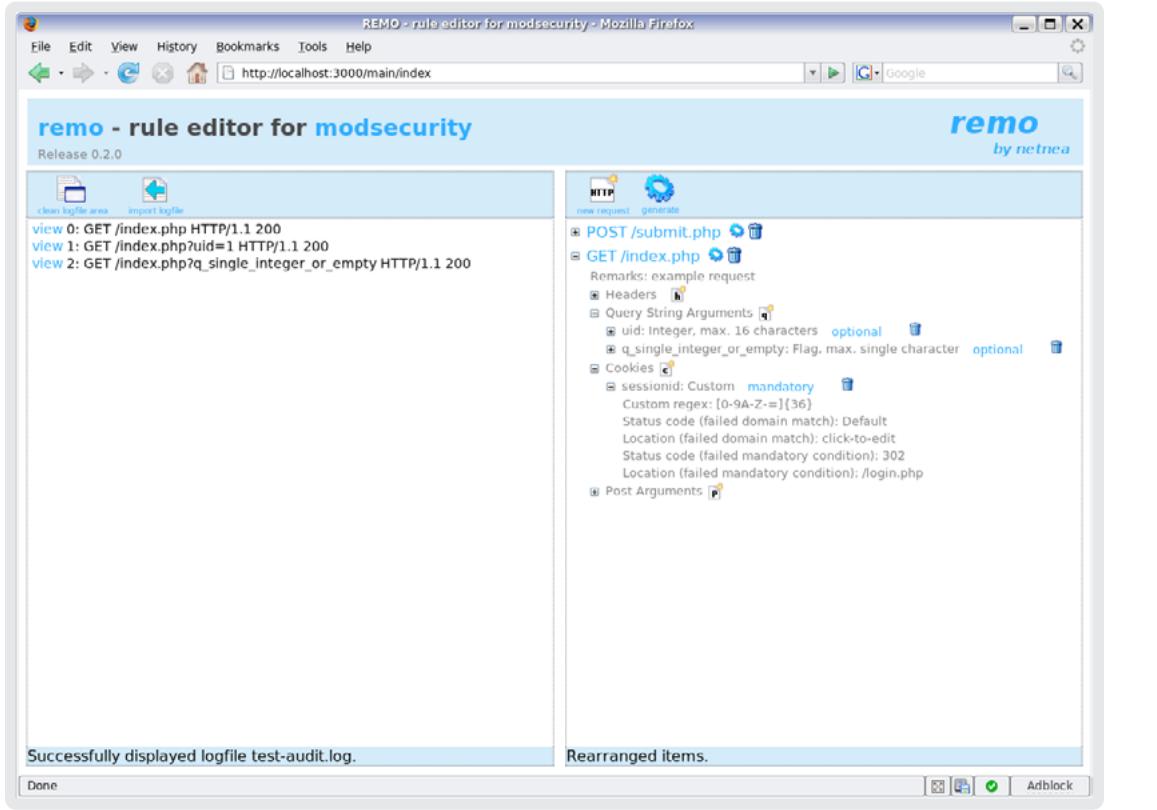
Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

*Czym jest i jak wykorzystać
podatność Relative Path
Overwrite/Path-Relative Style
Sheet Import (RPO/PRSSI)*

Mechanism HTTP Public Key Pinning





Rysunek 9. Edytor do budowy reguł ModSecurity.

Źródło: <https://www.netnea.com/cms/remo-a-rule-editor-for-modsecurity/>.

PODSUMOWANIE

Stosowanie procedury wirtualnych poprawek buduje kolejną warstwę bezpieczeństwa w podejściu „defense in depth”, która jest w stanie skutecznie powstrzymać atak na infrastrukturę. Wirtualna łata umożliwia redukcję ryzyk związanych z exploitami typu 0-day. Za pomocą takich opensourcowych narzędzi jak ModSecurity, obrońcy są w stanie realizować ochronę aplikacji o zamkniętym kodzie. Szczególnie warto rozważyć wdrożenie w swoich organizacjach opracowanej przez OWASP procedury stosowania wirtualnych poprawek i wkomponować ją w cykl zarządzania bezpieczeństwem.



Bartosz Jerzman. Admin, pentester z zacięciem na defensywne aspekty bezpieczeństwa IT. Założyciel secman.pl

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning





AKTUALNE WYDANIE PROGRAMISTY W EMPIKACH LUB PRENUMERACIE

POLECAMY

**WSZYSTKIE WYDANIA ARCHIWALNE
+ PRENUMERATA ELEKTRONICZNA NA ROK.
TYLKO 230 PLN!**

ZAMÓW:

<http://programistamag.pl/typy-prenumeraty/>

Czym jest atak Padding Oracle

Padding Oracle to atak pozwalający na wyłuskanie tekstu jawnego z za-szyfrowanych danych bez znajomości klucza szyfrującego, a także – bez konieczności wyszukiwania błędów w samym algorytmie szyfrującym. Swego czasu, atak przeprowadzany był między innymi w celu złamania mechanizmu sesji w ASP.NET, a to z kolei pozwalało na przejęcie uprawnień dowolnego użytkownika. Innym, świeższym przykładem, jest opisany w 2014 roku atak POODLE, czyli *Padding Oracle On Downgraded Legacy Encryption*, skutkujący zaleceniem niestosowania SSLv3. W tym artykule omówimy, czym jest Padding Oracle, oraz przedstawimy schemat ataku wraz z metodami obrony.

Zaczniemy jednak od podstaw. Najpopularniejsze obecnie algorytmy szyfrujące, takie jak AES czy 3DES, działają, bazując na blokach o stałych wielkościach. Na przykład: AES jest zdefiniowany do szyfrowania bloków o wielkości 128 bitów (16 bajtów). Oznacza to, że jeżeli chcemy zaszyfrować dane o dowolnej wielkości, musimy zadbać o to, by do funkcji szyfrującej przekazać je w postaci 16-bajtowych bloków.

Pojawiają się zatem zasadnicze pytania

- » Co zrobić, gdy mamy dane krótsze niż 16 bajtów? W jaki sposób wypełnić pozostałe bajty?
 - » Co zrobić, gdy mamy dane większe niż 16 bajtów? W jaki sposób podzielić je na bloki? W jaki sposób szyfrować poszczególne bloki?

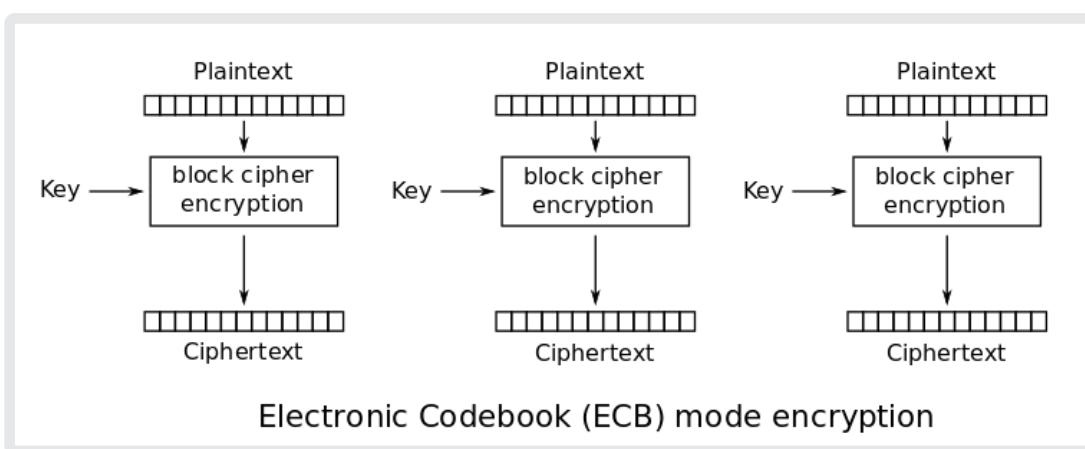
Odpowiedzi na te pytania zamieszczam w dalszej części artykułu. Zapraszam do lektury

PADDING – PKCS#7

Zacznijmy od ustalenia, co zrobić z danymi niebędącymi wielokrotnością wielkości bloku danego algorytmu. Na potrzeby przykładów założymy, że będziemy używać algorytmu AES, więc bloki będą miały 16 bajtów i chcemy zaszyfrować ciąg znaków "Programista". Ponieważ mamy tylko 11 znaków, musimy czymś wypełnić pozostałe 5 znaków, by móc użyć AES-a. Innymi słowy – musimy użyć **paddingu**. Najczęściej używanym schematem paddingu jest standard PKCS#7, zdefiniowany w RFC 5652. Algorytm jest prosty – polega bowiem na tym, aby do szyfrowanego ciągu znaków dodać bajty, których wartość jest równa liczbie brakujących bajtów do wielkości bloku.

SPOSÓBY ŁĄCZENIA BLOKÓW ZASZYFROWANYCH

Kolejnym problemem, z którym musimy się zmierzyć, chcąc używać algorytmów szyfrujących, jest sposób łączenia bloków. Najprostszym sposobem szyfrowania danych większych niż 16 bajtów, byłoby podzielenie ich na bloki o długości 16 bajtów każdy i szyfrowanie każdego z nich osobno. Taki tryb pracy znany jest jako ECB (ang. *Electronic CodeBook*, rysunek 1).



Rysunek 1. Sposób działania szyfrowania ECB (źródło: Wikipedia).

Zaletą tego trybu pracy jest jego wydajność – szyfrowanie każdego bloku jest niezależne, co pozwala na łatwe zrównoleglenie zarówno procesu szyfrowania, jak i deszyfrowania. Istotną wadą jest jednak fakt, że dla pewnych typów danych łatwo dostrzec wzorce, co sprawia, że szyfrowanie staje się nieefektywne. Najsłynniejszym na to przykładem jest szyfrowanie bitmapy przedstawiającej linuksowego pingwina

Mechanism Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

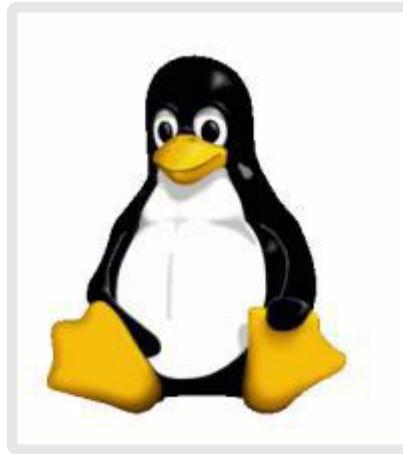
Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path–Relative Style Sheet Import (RPO/PRSSI)

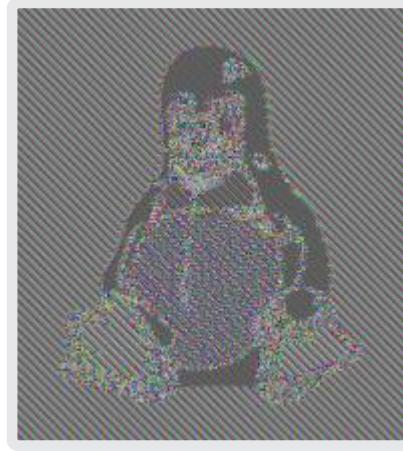
Mechanism HTTP Public Key Pinning



(rysunek 2). Pomimo użycia silnego algorytmu szyfrującego dla poszczególnych bloków, wynik końcowy ewidentnie wskazuje, jaki obraz został zaszyfrowany (rysunek 3).



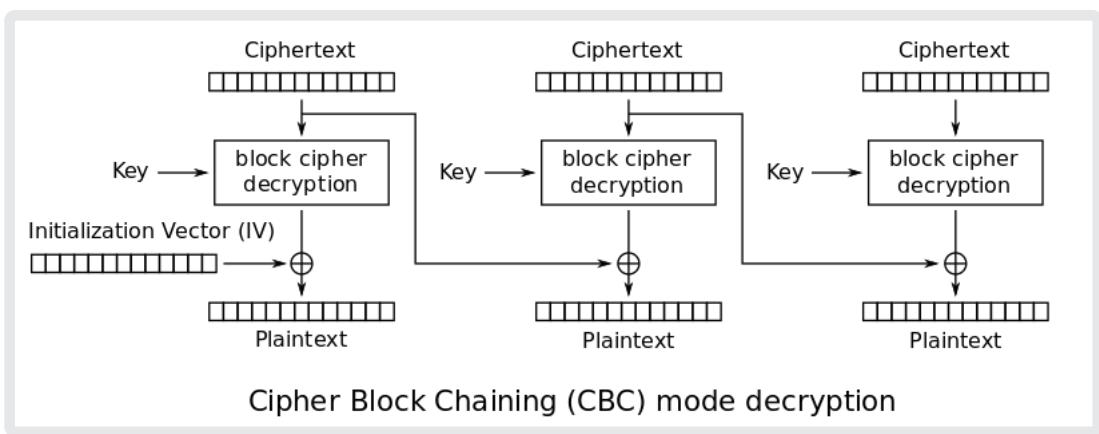
Rysunek 2. Tux przed szyfrowaniem (autor obrazka: lewing@isc.tamu.edu).



Rysunek 3. Tux zaszyfrowany w trybie ECB (autor obrazka: lewing@isc.tamu.edu).

Demaskuje to najistotniejszą wadę ECB – te same bloki danych tworzą ten sam szyfrrogram. W danych binarnych często może to być strumień null-bajtów.

Aby rozwiązać ten problem, powstało wiele innych sposobów łączenia szyfrowanych bloków danych. Najpopularniejszym z nich (a zarazem tym, w którym pojawiła się *Padding Oracle*) jest CBC (ang. *Cipher Block Chaining*). Na rysunku 4 przedstawiony jest schemat deszyfrowania danych w CBC.



Rysunek 4. Deszyfrowanie danych CBC (źródło: Wikipedia).

W CBC szyfrrogram każdego bloku jest zależny od poprzedniego. Aby zaszyfrować dane danego bloku, musimy najpierw wykonać operację XOR na tekście jawnym danego bloku oraz szyfrrogramu poprzedniego bloku. Z kolei w przypadku deszyfrowania – najpierw deszyfrujemy dane, używając algorytmu szyfrującego, a na wyniku tej operacji przeprowadzamy XOR z tekstem jawnym poprzedniego bloku. Specjalnym przypadkiem jest pierwszy blok, dla którego nie można przeprowadzić XOR-a z poprzednim blokiem. Zamiast tego, używa się tzw. wektora inicjalizacyjnego (IV – Initialization Vector). Dzięki takiemu sposobowi łączenia bloków, zmiana choćby jednego bajtu w tekście jawnym (zakładając, że używamy odpowiedniego algorytmu szyfrującego) wpływa na zmianę wszystkich kolejnych bloków w szyfrrogramie.

PADDING ORACLE

Okazuje się, że sposób działania CBC pozwala na specyficzny dla tego trybu atak, czyli *Padding Oracle*. Aby można było taki atak wykonać, muszą być spełnione dwa warunki:

1. aplikacja musi pozwalać użytkownikowi na podanie własnego szyfrrogramu,
2. aplikacja musi w jakiś sposób sygnalizować błędy paddingu.

Warunek pierwszy można spełnić dość łatwo – np. w aplikacjach webowych, często szyfrowane są wartości ciasteczek bądź niektórych parametrów GET/POST. Drugi warunek także często może być spełniony, jeżeli w aplikacji jest włączone wyświetlanie wyjątków. Praktycznie każda technologia, w której udostępniono metody do szyfrowania CBC, zasygnalizuje błąd (wyjątek), jeżeli po deszyfrowaniu danych zostanie

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



wykryty błąd paddingu. Błąd ten pojawia się, gdy padding nie jest zgodny z oczekiwany. Na przykład, we wcześniejszym przykładzie mieliśmy paddingowany ciąg znaków "Programista\x05\x05\x05\x05\x05". Gdyby w wyniku deszyfrowania danych, aplikacja otrzymała ciąg np. "Programista\x05\x05\x05\x05\x04", wyświetliłaby błąd o niepoprawnym paddingu.

Treść błędu zależy od konkretnej technologii, np. w .NET zobaczymy błąd „Padding is invalid and cannot be removed”, zaś w Javie będzie to po prostu „Bad padding”.

Podsumowując: jeżeli w aplikacji będziemy w stanie wpływać na dane, które będą deszyfrowane oraz aplikacja w wyraźny sposób zasygnalizuje błąd paddingu, będziemy w stanie przeprowadzić atak Padding Oracle.

Gdy spojrzymy ponownie na rysunek 4, możemy zauważyc, że tekst jawny każdego bloku zależy de facto od dwóch elementów: wyniku funkcji deszyfrującej dla tego bloku oraz szyfrogramu bloku poprzedniego.

Przymijmy następujące oznaczenia:

- » $C[i]$ – szyfrogram i-tego bloku (ciphertext),
- » $P[i]$ – tekst jawny i-tego bloku (plaintext),
- » $D(\cdot)$ – funkcja deszyfrująca pojedynczy blok,
- » \wedge – operacja XOR.

Z diagramu wprost wynika, że $P[i] = C[i-1] \wedge D(C[i])$. Zauważamy, że nad szyfrogramami mamy kontrolę, a więc możemy wpływać na $C[i-1]$, co w konsekwencji pozwala wpływać na $P[i]$. Oczywiście zwykle nie wiemy, jaka jest wartość $P[i]$ oraz co wychodzi z $D(C[i])$, jednak dzięki odpowiedniej analizie błędów paddingu będziemy w stanie to zrobić.

W pierwszej kolejności, będziemy chcieli tak ustawić wartość $C[i-1]$, aby w wyniku działania algorytmu deszyfrującego nie został zwrócony błąd paddingu. W tym celu, będziemy zmieniać wartość ostatniego bajtu tego bloku na kolejne bajty od "\x00" do "\xff" z pominięciem oryginalnego bajtu $C[i-1][15]$.

Na potrzeby przykładu założymy, że:

- » $C[i-1] = "abcdabcdabcd"$
- » $C[i] = '0b\xec\x1b\x18Ug5\x08\xa9\xc4\x9d\xec\x13R\x9d'$

Zaczynamy więc od prób wpisywania kolejnych bajtów na ostatniej pozycji w $C[i-1]$.

- » 'abcdabcdabcdabc\x00'
- » 'abcdabcdabcdabc\x01'
- » 'abcdabcdabcdabc\x02'
- » itp. ...

Okazuje się, że dla $C'[i-1] = 'abcdabcdabcdabc'$ nie otrzymamy błędu paddingu.

Zastanówmy się teraz, jakie możemy wyciągnąć z tego wnioski?

Otoż błąd paddingu na pewno nie pojawi się, gdy $P'[i][15] = "\x01"$ ponieważ jest to jednobajtowy padding, a więc pojedyncza wartość "\x01" jest dla niego poprawna (co ciekawe, pewne implementacje akceptują także padding "\x00", np. hurlant – biblioteka kryptograficzna dla ActionScript). Idąc dalej – jeśli zmodyfikujemy nasz wzór w taki sposób, by odnieść go tylko do ostatniego bajtu bloku, mamy: $P'[i][15] = C'[i-1][15] \wedge D(C[i])[15]$.

Zauważmy, że w równaniu składającym się z trzech elementów – wartości dwóch z nich znamy! Możemy więc obliczyć ostatni składnik wg wzoru:

$$\begin{aligned} D(C[i])[15] &= P'[i][15] \wedge C'[i-1][15] \\ D(C[i])[15] &= "\x01" \wedge "f" \\ D(C[i])[15] &= "g" (0x67) \end{aligned}$$

Voilà! Wiemy, jaki jest ostatni bajt wychodzący z funkcji deszyfrującej. Uzbrojeni w tę wiedzę, wracamy do oryginalnej wartości ostatniego bajtu bloku $C[i-1]$ – a więc "d". Wracamy do pierwotnego wzoru:

$$\begin{aligned} P[i][15] &= C[i-1][15] \wedge D(C[i])[15] \\ P[i][15] &= "d" \wedge "g" \\ P[i][15] &= "\x03" \end{aligned}$$

W ten sposób udało nam się zdeszyfrować oryginalną wartość ostatniego bajtu bloku!

Jest to "\x03", co wskazuje na to, że w bloku jest trzybajtowy padding. Spróbujmy to potwierdzić, deszyfrując przedostatni bajt bloku.

W pierwszym kroku staraliśmy się znaleźć taką wartość ostatniego bajtu $C[i-1]$, dla której $P'[i-1][15] = "\x01"$. W drugim kroku natomiast, będziemy się starali wymusić dwubajtowy padding, a więc spełnione będą następujące warunki:

- » $P'[i][15] = "\x02"$
- » $P'[i][14] = "\x02"$

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



Wykorzystując wiedzę z poprzedniego kroku, jesteśmy już w stanie ustalić takie $C'_{[i-1]}[15]$, by otrzymać oczekiwany $P'_{[i]}[15]$. Mianowicie:

$$\begin{aligned}C'_{[i-1]}[15] &= P'_{[i]}[15] \wedge D(C[i])[15] \\C'_{[i-1]}[15] &= "\x02" \wedge "g" \\C'_{[i-1]}[15] &= "e"\end{aligned}$$

Z kolei, aby ustalić wartość $C'_{[i-1]}[14]$, będziemy musieli robić to samo, co po-przednio, a mianowicie – próbować wszystkich możliwych wartości oprócz oryginalnej, czyli kolejno:

- » $C'_{[i-1]} = 'abcdabcdabcdab\x00e'$
- » $C'_{[i-1]} = 'abcdabcdabcdab\x01e'$
- » itd. ...

Tym razem okaże się, że błędu paddingu nie otrzymamy dla "abcdabcdabcdab-be". Powtarzamy więc kroki z wyciągania ostatniego bajtu:

$$\begin{aligned}D(C[i])[14] &= P'_{[i]}[14] \wedge C'_{[i-1]}[14] \\D(C[i])[14] &= "\x02" \wedge "b" \\D(C[i])[14] &= "\x03"\end{aligned}$$

Wiemy już, jaki wynik dała funkcja deszyfrująca na przedostatnim bajcie, spróbujmy więc ustalić wartość tekstu jawnego dla tego bajtu:

$$\begin{aligned}P'_{[i]}[14] &= C'_{[i-1]}[14] \wedge D(C[i])[14] \\P'_{[i]}[14] &= "c" \wedge "\x03" \\P'_{[i]}[14] &= "\x03"\end{aligned}$$

Potwierdziliśmy więc, że przedostatnim bajtem jest w istocie "\x03". Wyciąganie kolejnych bajtów odbywa się w analogiczny sposób:

1. chcąc wyciągnąć bajt trzeci od końca, musimy tak ustalić trzy ostatnie bajty, by padding wynosił "\x03\x03\x03",
2. chcąc wyciągnąć bajt czwarty od końca, musimy tak ustalić cztery ostatnie bajty, by padding wynosił "\x04\x04\x04\x04" itd.

Chętnych do przećwiczenia Padding Oracle zapraszam do wykonania trzeciego zadania dostępnego pod tym adresem: <https://rozwal.to/zadanie/20>.

METODY OCHRONY

Zobaczyliśmy, w jaki sposób używanie trybu CBC do szyfrowania i wyświetlania błędów paddingu może pozwolić na deszyfrację danych – bez znajomości klucza szyfrującego.

Jak się zatem obronić przed tego typu atakami?

Przede wszystkim należy zadbać o to, aby aplikacja nie wyświetlała wyjątków wskazujących wyraźnie na błędy w paddingu. Ujawnianie tego typu informacji zna-cząc o ułatwia przeprowadzanie ataków. Należy jednak pamiętać, że nawet jeśli błąd nie jest wprost wyświetlany, w pewnych sytuacjach można wywnioskować, że taki błąd wystąpił (np. serwer odpowiada kodem 500).

Z tego powodu zalecaną metodą ochrony jest dodawanie kodu MAC razem z zaszyfrowanymi wiadomościami. Można w tym celu użyć funkcji HMAC-SHA256. Przed przystąpieniem do deszyfrowania danych sprawdzamy najpierw, czy klucz HMAC-SHA256 pasuje do danych. Jeżeli nie – natychmiast odrzucamy żądanie. Dzięki temu, przeprowadzenie ataku nie będzie możliwe, bowiem niemożliwym bę-dzie wygenerowanie poprawnego klucza.

PODSUMOWANIE

Padding Oracle to atak, który dotyczy danych szyfrowanych w trybie CBC. Ze względu na specyficzne właściwości tego trybu, jak i najpopularniejszego schema-tu paddingu, możliwe jest deszyfrowanie danych niezależnie od tego, jak silny jest algorytm szyfrujący.

Zalecaną metodą zabezpieczenia się przed takim atakiem, jest dodawanie klu-cza HMAC do szyfrowanych danych – i sprawdzanie go przed przystąpieniem do deszyfracji.

Linki:

[1]: RFC 5652, definicja PKCS#7, <http://tools.ietf.org/html/rfc5652#section-6.3>

Michał Bentkowski. Realizuje testy penetracyjne oraz audyty bezpieczeństwa w firmie **Securitum**. Autor w serwisie sekurak.pl. Aktywnie (i w sukcesem) uczestniczy w znanych programach bug bounty.



[Mechanizm Service Workers](#)

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)





...praca dla pentestera...

...hackuj prawdziwe systemy i zarabiaj pieniądze...

securitum.pl/praca

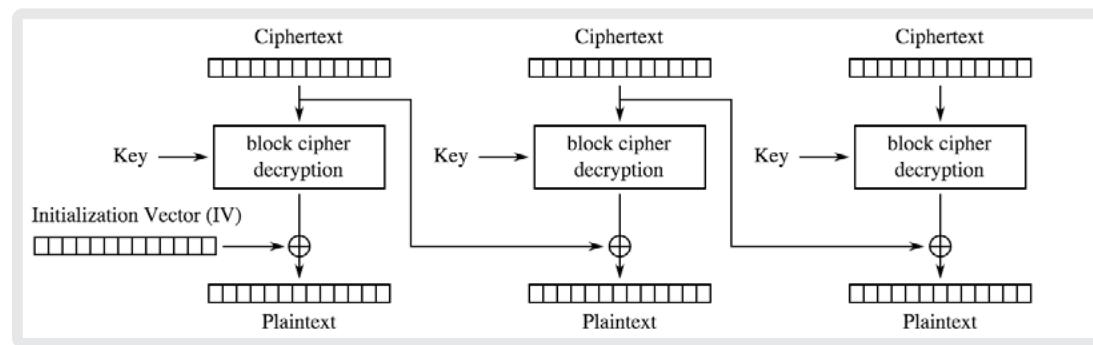
Czym jest Bit-Flipping

Bit-Flipping jest popularnym atakiem na blokowe algorytmy szyfrowania w trybie CBC (ang. Cipher Block Chaining), wiążąca bloków zaszyfrowanych. W wielu przypadkach prowadzi do przejęcia kont innych użytkowników (często administracyjnych), a w skrajnych przypadkach – nawet do zdalnego wykonywania kodu.

Ponieważ Bit Flipping to atak skierowany na szyfrowanie blokowe w trybie CBC, możemy dzięki niemu – bez znajomości klucza szyfrującego – zmodyfikować tekst jawny będący wynikiem procesu deszyfrowania.

TRYB CBC

Wyjaśniając mechanizm przeprowadzenia ataku Bit-Flipping, w pierwszej kolejności musimy wytłumaczyć, w jaki sposób działa deszyfrowanie w trybie CBC.



Rysunek 1. Schemat procesu deszyfrowania w trybie CBC ([Wikipedia](#)).

Szyfrogram składa się z bloków o stałej długości (8, 16 lub 32 bajtów), w naszym przykładzie posłużymy się najbardziej popularnym – 16-bajtowym. Do szyfrogramu dołączany jest IV czyli Initialization Vector, generowany podczas szyfrowania. To właśnie dzięki niemu szyfrowanie tych samych danych generuje różne szyfrogramy – tak długo, jak Initialization Vector (IV) jest różny.

Nasz szyfrogram wygląda więc w taki sposób:

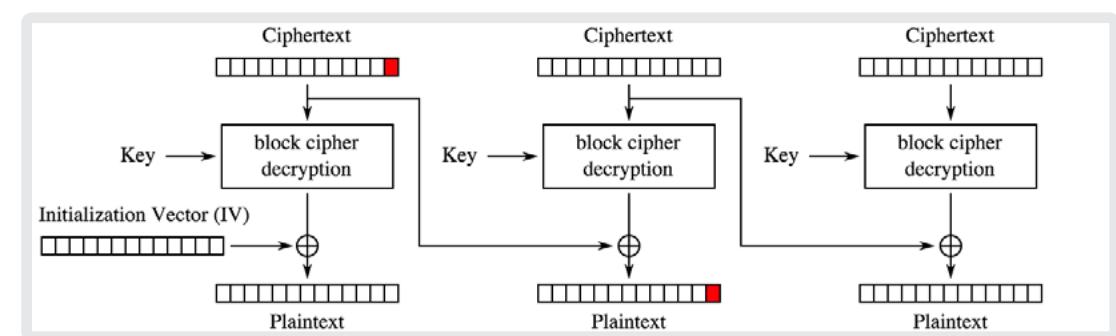
[Initialization Vector] [Blok 1] [Blok 2] [Blok 3]

Pierwszym etapem deszyfrowania jest rozbicie szyfrogramu na bloki i wydzielanie bloku IV.

Blok jest deszyfrowany za pomocą klucza, a następnie poddawany procesowi xorowania z poprzednim blokiem szyfrogramu lub w przypadku bloku pierwszego z wartością IV (Initialization Vector).

INTEGRALNOŚĆ CBC

Szyfrowanie CBC nie dostarcza żadnego mechanizmu, który dbałby o integralność danych, co sprawia, że modyfikując szyfrogram, wpływamy na wynikowy tekst jawny.



Rysunek 2. Schemat procesu deszyfrowania: zaznaczono ostatni bajt bloku.

Przejdźmy więc do prostego przykładu **szyfrowania** tekstu „Sekurak Party” za pomocą algorytmu AES w trybie CBC.

- Do tekstu dodawany jest padding „Sekurak Party\x03\x03\x03” (w naszym wypadku PKCS #7 – można o tym więcej przeczytać w artykule Michała Bentkowskiego na temat padding Oracle, str. 31–34),
- Generowany jest 16-bajtowy Initialization Vector (IV),
- Tekst jawny jest xorowany z IV i szyfrowany przy użyciu klucza szyfrującego,
- Wynikowy szyfrogram wygląda następująco:

[Initialization Vector] [Blok 1]

W tym konkretnym przypadku, **deszyfrowanie** będzie wyglądać tak:

- Szyfrogram zostanie rozbity na dwa bloki 16 bajtowe – IV oraz Blok 1,
- Blok 1 przejdzie przez algorytm deszyfrowania przy użyciu klucza szyfrującego,
- Wynikowy blok zostanie poddany xorowaniu z wartością IV i otrzymamy tekst jawny.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

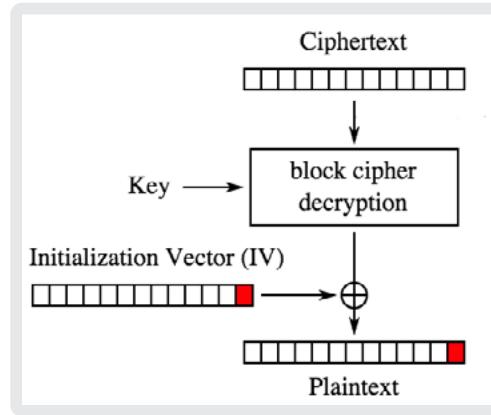
Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



BIT-FLIPPING – PRZYKŁADOWY ATAK

Do przeprowadzenia ataku Bit-Flipping wymagana jest wiedza dotycząca tekstu jawnego konkretnego szyfrogramu. Mamy więc (opisany poniżej) prosty skrypt, który potraktujemy jak czarną skrzynkę. Należy pamiętać o **niedekodowaniu klucza**, chcemy przecież przeprowadzić atak bez jego znajomości.



Rysunek 3. Deszyfrowanie pojedynczego bloku szyfrogramu.

Przyjrzyjmy się zatem przykładowemu skryptowi podatnemu na atak Bit-Flipping.

Listing 1. Skrypt podatny na atak Bit-Flipping.

```

#!/usr/bin/env python

import sys
from base64 import b64decode
from Crypto.Cipher import AES

def decrypt(cipher, key):
    IV = cipher[:16]
    cipher = cipher[16:]
    aes = AES.new(key, AES.MODE_CBC, IV=IV)
    plain = aes.decrypt(cipher)
    return plain

key = b64decode("WWVsbG93IFN1Ym1hcmluZQ==")
plain = decrypt(sys.argv[1].decode('hex'), key)
print repr(plain)
  
```

Skrypt przyjmuje jako argument szyfrogram zakodowany w postaci HEX-a, następnie przeprowadza proces deszyfrowania i wyświetla tekst jawnny.

Załóżmy więc, że tekstowi jawnemu „Sekurak Party\x03\x03\x03” odpowiada szyfrogram (szyfrogram został zakodowany w formie heksadecymalnej, na czerwono zaznaczono natomiast Initialization Vector):

6ca79ad332213699f3f0bf8b50bb3a19f8c17dcd69a95b8dc3bf92bc88d024d3

```

root@b00m:~/crypto# ./blackbox.py 6ca79ad332213699f3f0bf8b50bb3a19f8c17dcd69a95b8dc3bf92bc88d024d3
'Sekurak Party\x03\x03\x03'
root@b00m:~/crypto#
  
```

Rysunek 4. Działanie skryptu.

Wiedząc, że ostatni bajt to „\x03” i że jest on wynikiem operacji XOR „\x19” (ostatni bajt IV) z jakimś X (oczywiście można w prosty sposób wyliczyć jego wartość, choć nie ma takiej potrzeby):

$$0x19 \wedge X = 0x03$$

Aby jako wynik otrzymać 0x1, musimy poddać procesowi xorowania w pierwszej kolejności obie strony równania przez wartość 0x03:

$$0x19 \wedge X \wedge 0x03 = 0x00$$

A następnie przez wartość 0x1:

$$0x19 \wedge X \wedge 0x03 \wedge 0x01 = 0x01$$

Jak widać, cały algorytm sprowadza się do wykonania operacji XOR ostatniego bajtu IV – w pierwszej kolejności przez ostatni bajt tekstu jawnego następnego bloku, a następnie przez wartość, na jaką chcemy docelowy bajt zmienić.

Ostatni bajt IV musi więc zostać zamieniony na wartość:

$$0x19 \wedge 0x03 \wedge 0x01 = 0x1b$$

„6ca79ad332213699f3f0bf8b50bb3a1b1bf8c17dcd69a95b8dc3bf92bc88d024d3”

Tak utworzony szyfrogram przekazujemy do naszego skryptu czarnej skrzynki:

```

root@b00m:~/crypto# ./blackbox.py 6ca79ad332213699f3f0bf8b50bb3a1b1bf8c17dcd69a95b8dc3bf92bc88d024d3
'Sekurak Party\x03\x03\x01'
root@b00m:~/crypto#
  
```

Rysunek 5. Działanie skryptu.

Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



Tekst jawnny rzeczywiście został zmodyfikowany bez znajomości klucza szyfrującego. Proces jest na tyle prosty, że możemy stworzyć skrypt, który wygeneruje nowy Initialization Vector pozwalający zmienić wartość całego tekstu jawnego.

Aby zamienić ciąg „Sekurak Party\x03\x03\x03” na ciąg „Sekurak Hacking\x01” należy dokonać operacji XOR na odpowiednich bajtach bloku poprzedzającego – w tym wypadku IV.

Listing 2. Skrypt przeprowadzający atak Bit-Flipping.

```
def xor(a, b):
    res = ""
    for i in range(0, len(a)):
        res += chr(ord(a[i]) ^ ord(b[i]))
    return res

cipher = "6ca79ad332213699f3f0bf8b50bb3a19f8c17dc69a95b8dc3bf92bc88d024d3".
decode('hex')
IV = cipher[:16]
old = "Sekurak Party\x03\x03\x03"
new = "Sekurak Hacking\x01"

# nowe IV
newiv = xor(IV, xor(new, old))

print "Nowe IV: %s" % newiv.encode('hex')
print "Nowy szyfrogram: %s" % (IV + cipher[16:]).encode('hex')
```

Wynik działania skryptu:

```
root@b00m:~/crypto# ./bitflip.py
Nowe IV: 6ca79ad332213699ebf0ae9440d65e1b
Nowy szyfrogram: 6ca79ad332213699ebf0ae9440d65e1bf8c17dc69a95b8dc3bf92bc88d024d3
root@b00m:~/crypto#
```

Rysunek 6. Atak Bit-Flipping.

Teraz powstaje szyfrogram, który możemy załadować do algorytmu deszyfrującego, co zwróci nam wartość „Sekurak Hacking\x01”.

```
root@b00m:~/crypto# ./blackbox.py 6ca79ad332213699ebf0ae9440d65e1bf8c17dc69a95b8dc3bf92bc88d024d3
'Sekurak Hacking\x01'
root@b00m:~/crypto#
```

Rysunek 7. Zmodyfikowany tekst jawny.

ZDOBYWANIE INFORMACJI DOTYCZĄCYCH TEKSTU JAWNEGO

Jak wspominałem wcześniej, próbujemy zdobyć wiedzę dotyczącą tekstu jawnego dla konkretnego szyfrogramu. Te informacje możemy zdobyć na trzy sposoby:

1. przeprowadzając atak padding oracle,
2. korzystając z pewnej znajomości formatu danych,
3. modyfikując bajty w szyfrogramie i obserwując wyjścia aplikacji.

Pierwszy sposób został świetnie opisany przez Michała Bentkowskiego, dlatego odśalam do jego artykułu. Druga metoda to wiedza o tym, jaki format mogą przyjąć dane. Takim przykładem może być format JSON, który składa się z klamer, cudzysłówów oraz znaków dwukropka.

Skupmy się więc na sposobie trzecim, czyli modyfikacji bajtów i obserwacji wyjścia aplikacji.

PRAKTYCZNY ATAK

Przeprowadzimy bardziej praktyczny atak, którego celem będzie pozyskanie poufnych danych.

Nieco zmodyfikowany skrypt czarnej skrzynki:

Listing 3. Skrypt wczytujący plik na podstawie zaszyfrowanego ciągu.

```
#!/usr/bin/env python

import sys
import json

from base64 import b64decode
from Crypto.Cipher import AES

def decrypt(cipher, key):
    IV = cipher[:16]
    cipher = cipher[16:]

    aes = AES.new(key, AES.MODE_CBC, IV=IV)
    plain = aes.decrypt(cipher)

    return plain

key = b64decode("WWVsbg93IFN1Ym1hcmluZQ==")
plain = decrypt(sys.argv[1].decode('hex'), key)

try:
    res = json.loads(plain)
    print 'Reading %s...' % res['f']
```

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



```
f = open(res['f'], 'r')
print '-----\n%s\n-----' % f.read()
f.close()

except:
    sys.exit(1)
```

Uruchamiamy skrypt, przekazując jako argument następujący ciąg:

"0ec9a5b6a8304467899fda07d9eb15b741b186a4d266bb671927ae1e609389bd"

```
root@b00m:~/crypto# ./blackbox2.py 0ec9a5b6a8304467899fda07d9eb15b741b186a4d266bb671927ae1e609389bd
Reading test.txt...
Testowy Plik
-----
```

Rysunek 8. Odczytanie pliku „test.txt”.

Wyjście aplikacji wskazuje, że czytana jest treść pliku "test.txt".

Co się jednak stanie, jeżeli zaczniemy "flippować" bity w bloku IV dla konkretnych bajtów, zaczynając od końca?

Szyfrogram:

0ec9a5b6a8304467899fda07d9eb15b841b186a4d266bb671927ae1e609389bd

```
root@b00m:~/crypto# ./blackbox2.py 0ec9a5b6a8304467899fda07d9eb15b841b186a4d266bb671927ae1e609389bd
root@b00m:~/crypto#
```

Rysunek 9. Modyfikacja kolejnego bajtu szyfrrogramu.

Szyfrogram:

0ec9a5b6a8304467899fda07d9eb16b741b186a4d266bb671927ae1e609389bd

```
root@b00m:~/crypto# ./blackbox2.py 0ec9a5b6a8304467899fda07d9eb16b741b186a4d266bb671927ae1e609389bd
root@b00m:~/crypto#
```

Rysunek 10. Modyfikacja kolejnego bajtu szyfrrogramu.

Szyfrogram:

0ec9a5b6a8304467899fda07d9~~ec~~15b741b186a4d266bb671927ae1e609389bd

```
root@b00m:~/crypto# ./blackbox2.py 0ec9a5b6a8304467899fda07d9ec15b741b186a4d266bb671927ae1e609389bd
Reading test.txt...
root@b00m:~/crypto#
```

Rysunek 11. Modyfikacja kolejnego bajtu szyfrrogramu.

Możemy więc wnioskować, że modyfikując zaznaczone w szyfrrogramie bajty, jesteśmy w stanie zmienić całkowicie wartość test.txt. Na czerwono zostało zaznaczony fragment ciągu, pod którym kryje się zaszyfrowany tekst „test.txt”:

0ec9a5b6a830~~4467899~~**fda07d9eb**15b741b186a4d266bb671927ae1e609389bd

Aby zmienić wartość test.txt na flag.txt musimy wykonać operację XOR:

$4467899fda07d9eb \wedge \text{hex}(\text{"test.txt"}) \wedge \text{hex}(\text{"flag.txt"}) = 566e9b8cda07d9eb$

W efekcie otrzymujemy szyfrogram:

0ec9a5b6a830~~566e9b8cda07d9eb~~**15b741b186a4d266bb671927ae1e609389bd**

Po załadowaniu szyfrrogramu do atakowanego skryptu zwracana jest treść pliku „flag.txt”:

```
root@b00m:~/crypto# ./blackbox2.py 0ec9a5b6a830566e9b8cda07d9eb15b741b186a4d266bb671927ae1e609389bd
Reading flag.txt...
-----
Sekretne Dane!
-----
root@b00m:~/crypto#
```

Rysunek 12. Poufne informacje pozyskane za pomocą ataku Bit-Flipping.

Bez znajomości klucza udało się zmienić tekst jawny z „test.txt” na „flag.txt” i przeczytać sekretne dane.

OGRANICZENIA ATAKU BIT FLIPPING

Atak Bit-Flipping ma spore ograniczenia. Poprzednie przykłady były stosunkowo proste i składały się tylko z dwóch elementów – IV oraz pierwszego bloku.

Co stanie się w przypadku, gdy zaszyfrowany ciąg jest znacznie dłuższy i tworzy szyfrogram składający się z kilku bloków?

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

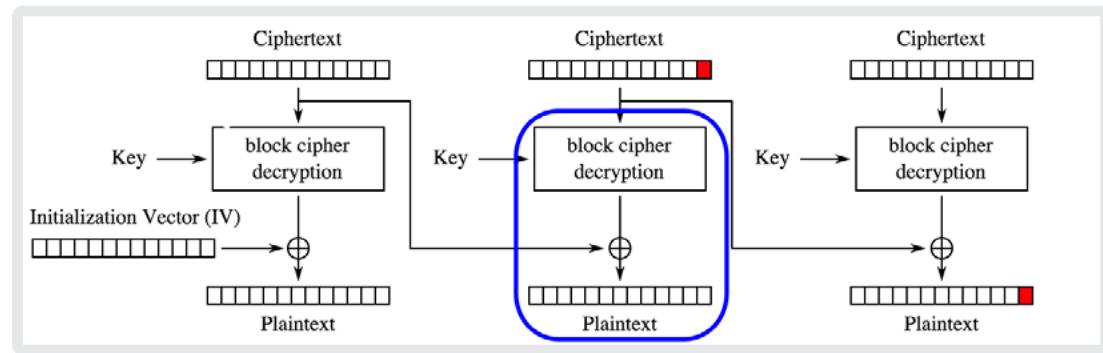
Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning





Rysunek 13. Deszyfrowanie kolejnych bloków szyfrogramu.

Jeżeli zaczniemy flippować bity w bloku 2, tak aby zmienić tekst jawny w bloku 3 (zaznaczone na czerwono), to szyfrogram w bloku 2 po deszyfrowaniu utworzy całkowicie inny tekst jawny w bloku 2. Takie zachowanie może prowadzić do błędu w przypadku, gdy w bloku 2 znajdowały się istotne dane – jak chociażby nazwy zmiennych czy zmiana formatu danych, np. "zepsucie JSON-a".

SPOSÓB OBEJŚCIA OGRANICZENIA

Jeżeli mamy możliwość modyfikacji danych, które są szyfrowane (np. username użytkownika), możemy spróbować ustawić ich długość oraz lokalizacje w taki sposób, aby "zepsuty blok" nadpisywał ich wartość.

Czyli, jeżeli szyfrowane dane wyglądają w taki sposób:

```
{"login":"test","admin":false}\x02\x02
```

utworzony szyfrogram da 2 bloki, które zostaną poprzedzone przez IV.

```
[ IV ] [{"login":"test"}, {"admin":false}\x02\x02]
```

Flippowanie bitów w bloku `["login":"test"]` spowoduje jego deformację i w efekcie zepsucie formatu JSON. Można jednak utworzyć konto z na tyle długim loginem, że jego część będzie stanowiła modyfikowany blok.

Rejestrujemy więc konto z loginem "AAAAAAABBBBBBBBBBBBBBBB", co powoduje że aplikacja zaszyfruje taki ciąg:

```
{"login":"AAAAAAABBBBBBBBBBBBBBBB","admin":false}
```

Ciąg zostanie rozbity na cztery 16-bajtowe bloki (IV + 3 bloki)

```
[ IV ] [{"login":"AAAAAA"} [BBBBBBBBBBBBBBBB] [", "admin":false}]
```

Dzięki temu, możemy modyfikować blok [BBBBBBBBBBBBBBBB], nie psując tym samym formatu JSON – oczywiście jeżeli nie pojawią się tam znaki, które w stringu znaleźć się nie powinny. Zainteresowanych odsyłam do [standardu opisującego format JSON](#).

PODSUMOWANIE – ZALECANE METODY OBRONY

Zalecaną metodą zabezpieczenia przed atakiem Bit-Flipping jest dbanie o integralność szyfrogramu przez dodanie klucza HMAC i jego weryfikację przed deszyfracją danych.

Należy przyjąć zasadę, że nigdy nie powinno się wykonywać operacji kryptograficznych na nieuwierzytelnionych danych.

Marcin Bury. Realizuje testy penetracyjne i audyty bezpieczeństwa w firmie **Securitum**. Posiadacz certyfikatów CEH, OSCE i OSCP.



Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Upload plików zalicza się do najczęściej występujących funkcjonalności w aplikacjach webowych. Zazwyczaj wiąże się z wgrywaniem na serwer obrazków bądź dokumentów. Jest zarazem miejscem, które bardzo chętnie badają pentesterzy, ze względu na liczne błędy bezpieczeństwa w implementacjach. W tym artykule przedstawimy najczęściej występujące błędy oraz pokażemy, w jaki sposób mogą one zostać wykorzystane. Omówimy także sposoby obrony.

DWA SPOJRZENIA

Na wszystkie omawiane w tym artykule implementacje będziemy spoglądać z dwóch stron:

- » **server-side:** jak błędy w danej implementacji wykorzystać od strony serwerowej, a mianowicie jak doprowadzić do wykonania dowolnego kodu (RCE – remote code execution) po stronie serwera,
- » **client-side:** w przypadku ataków po stronie klienta będzie chodziło albo o wykonanie XSS-a, albo o możliwość wykradnięcia danych (takich jak tokeny CSRF) z poziomu innej domeny.

DOPUSZCZANIE WSZYSTKICH TYPÓW PLIKÓW

Najbardziej podstawowa implementacja, która stale pojawia się w aplikacjach webowych. Zakładamy, że aplikacja zapisuje na serwerze plik o takim samym rozszerzeniu, jakie miał oryginalny plik, a ponadto nie jest sprawdzana zawartość pliku.

Spojrzenie server-side

Rozważmy kilka możliwych rozwiązań zapisywania pliku po stronie serwera.

1. Plik dostępny bezpośrednio przez URL (plik znajdujący się w webroocie)

Najprostszy możliwy wariant: użytkownik wgrywa plik o nazwie `hack.php`, który potem jest dostępny pod adresem np. `http://www.example.com/uploads/hack.php`. Bezpośrednie odwołanie się do tego URL-a prowadzi prosto do RCE w aplikacji.

2. Plik w webroocie, ale bez możliwości wykonywania

Istnieje możliwość zabezpieczenia się przed ww. atakiem, uniemożliwiając wykonywanie plików `.php` w pojedynczym katalogu. Można to zrobić dyrektywą `php_flag engine off` w pliku `.htaccess` lub jednym z plików konfiguracyjnych Apache. Dzięki temu po odwołaniu się bezpośrednio do pliku `http://www.example.com/uploads/hack.php` zostanie wyświetlone jego źródło, nie zostanie jednak wykonany kod. Skoro jednak atakujący może wysyłać pliki o dowolnych nazwach, może wysłać plik `.htaccess` o następującej zawartości:

```
1 AddHandler application/x-httpd-php .jpeg
```

Spowoduje to, że pliki o rozszerzeniu `.jpeg` będą interpretowane przez parser PHP, tym samym obchodząc zabezpieczenie. Dopuszczanie plików o dowolnych rozszerzeniach może również być przydatne w przypadku wystąpienia błędu LFI (*Local File Inclusion*), jeśli w innym miejscu aplikacji pojawi się kod podobny do poniższego:

```
1 <?php
2 include 'includes/'. $_GET['module']. '.php';
3 ?>
```

3. Pliki poza webrootem

Rozwiązanie, w którym uploadowane pliki znajdują się poza głównym drzewem katalogów dostępnym bezpośrednio przez URL-e. Na przykład webroot aplikacji może znajdować się w `/var/www`, zaś uploady w `/var/uploads`. W tym przypadku nie zadziała już sztuczka z `.htaccess`, jednak nadal plik może być przydatny w przypadku błędu LFI.

4. Pliki o zmienionych nazwach

Czasem programiści decydują się zapisywać plik o takim samym rozszerzeniu jak zostało wysłane, jednak zmieniana jest nazwa pliku. Czyli np. zamiast `hack.php` na serwerze mógłby zostać zapisany plik `4093b2546a7437d1a2ca3f9b7ac662e3.php`. Jeżeli nazwa nowego pliku jest później gdzieś bezpośrednio widoczna, to taki wariant nie różni się od omówionych wcześniej. Jednak nawet jeśli nazwa nie jest bezpośrednio znana, w pewnych przypadkach ataki mogą być możliwe:

- » Nazwa pliku tworzona jest w sposób przewidywalny, np. jako `md5(time())`. Na podstawie nagłówka `Date` z odpowiedzi HTTP można odgadnąć poprawną nazwę pliku.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



- » Inne miejsce w aplikacji pozwala na listing zawartości katalogów (na przykład w aplikacjach javowych taką możliwość daje **atak XXE**).
- » Nowa część nazwy pliku jest doczepiana do oryginalnej, tj. zamiast hack.php powstaje plik hack_d02eae87c92ec51173c3659216d87518.php. W takim przypadku zawsze warto spróbować ataków z bajtem zerowym w nazwie pliku. Nawet w nowych wersjach PHP atak tego typu może zadziałać; jeden z błędów null-byte (w funkcji move_uploaded_file) **poprawiono dopiero w marcu tego roku**.

Spojrzenie client-side

Jeżeli uploadowany plik jest dostępny bezpośrednio przez URL, istnieje szereg rozszerzeń, których wysłanie spowoduje wykonanie XSS-a. Najprościej wysłać prostu plik .html o treści:

```
1 <script>alert(1)</script>
```

XSS-y są możliwe jeszcze przez wiele rozszerzeń plików, które przeglądarka interpretuje jako XML, np. .xml, .xsl, .rss, .svg, .xslt i inne. Payload, który powinien zadziałać dla dowolnego pliku opierającego się o XML, to:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <html xmlns="http://www.w3.org/1999/xhtml">
3 <script>alert(1)</script>
4 </html>
```

Odsyłam również do [artykułu poświęconego w całości XSS-om przez SVG](#).

Kolejnym sposobem na wykonanie XSS-ów są pliki Flasha (.swf). Flash, poprzez klasę ExternalInterface, może odwoływać się do silnika JS w przeglądarce. Na GitHubie można znaleźć [przykładową implementację wykonywania XSSów przez .swf](#).

CZARNA LISTA ROZSZEŃ

Standardową odpowiedzią na powyższe problemy jest wprowadzenie czarnej listy rozszerzeń, innymi słowy: jawne podanie rozszerzeń, które nie są dopuszczalne. W przypadku wysłania do serwera pliku o rozszerzeniu z tej listy, odpowiada on komunikatem informującym o niepoprawnym typie.

Jakie problemy mogą wiązać się z takim rozwiązaniem?

Spojrzenie server-side

1. Zbyt uboga czarna lista

Autorzy aplikacji nie zawsze są świadomi, które konkretnie rozszerzenia skutkują wykonaniem kodu. Jeśli na czarnej liście znajduje się tylko rozszerzenie .php, być może zabezpieczenie będzie dało się obejść z użyciem .php5. W przypadku serwera Microsoft IIS – czarną listę składającą się z rozszerzenia .aspx, omijamy przez .asp itp.

2. Plik o wielu rozszerzeniach

Problem opisywany w tym punkcie jest specyficzny dla serwera Apache. Zasadniczo istnieją w nim dwa sposoby na zdefiniowanie obsługi plików .php przez parser tego języka: dyrektywy SetHandler oraz AddHandler. Przykładowy wpis w konfiguracji z użyciem tej drugiej:

```
1 AddHandler application/x-httpd-php .php
```

AddHandler ma jednak swoje specyficzne zachowanie, o którym [wspomina dokumentacja](#):

Filenames may have multiple extensions and the extension argument will be compared against each of them.

Oznacza to, że w przypadku wysłania do serwera pliku test.php.jpg, gdy na serwerze nie zdefiniowano żadnego handlera dla rozszerzeń .jpg (co jest bardzo prawdopodobne), taki plik zostanie wykonany przez parser PHP.

Co ciekawe, dyrektywa AddHandler była używana w domyślnej konfiguracji w dystrybucjach Red Hat Enterprise Linux 6/CentOS 6 oraz we wcześniejszych wersjach. Dopiero w wersji siódmej, wydanej w połowie zeszłego roku, zmieniono ją na SetHandler. Na wszelki wypadek sprawdźcie swoje konfiguracje :).

3. Niewrażliwość na wielkość liter

Problem dotyczy w szczególności aplikacji działających na Windowsie, gdzie wielkość liter w nazwach pliku nie jest brana pod uwagę. Możemy zatem natrafić na implementację, w której na czarnej liście znajdzie się tylko rozszerzenie .aspx, zaś plik o rozszerzeniu .ASPX zostanie już przepuszczony, co w efekcie, doprowadzi do RCE.

[Mechanizm Service Workers](#)

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



Spojrzenie client-side

Zazwyczaj w przypadku stosowania czarnej listy rozszerzeń twórcy aplikacji nie rozważają problemu ataków client-side. Z dużym prawdopodobieństwem będzie można wgrać plik o jednym z rozszerzeń wspomnianych wcześniej, tj. .html, .htm, .xml, .swf itp.

BIAŁA LISTA ROZSZERZEŃ

Podejście, w którym w aplikacji jawnie podano białą listę rozszerzeń, a więc listę rozszerzeń, których upload jest dopuszczalny. Dla obrazków, biała lista mogłaby przykładowo składać się z rozszerzeń: ["jpg", "png", "jpeg", "gif", "tiff"].

Spojrzenie server-side

1. LFI (Local File Inclusion)

W innym miejscu aplikacji może wystąpić błąd LFI.

```
1 <?php  
2 include 'includes/'. $_GET['template'];
```

Ponieważ aplikacja nie sprawdza zawartości plików, a tylko ich rozszerzenie, zuploadedany plik „obrazka” będzie mógł zostać wykorzystany do wykonania kodu PHP.

2. Weryfikacja zawartości pliku

Skoro aplikacja oczekuje obrazka, można dodać w niej kod weryfikujący, czy wysłany plik rzeczywiście nim jest, wykorzystując do tego jedną z bibliotek. Zasadniczo możemy tutaj wyróżnić dwa sposoby weryfikacji:

- » Sprawdzenie pliku bez modyfikacji jego zawartości. Można wówczas umieścić kod PHP w tagach EXIF lub (w przypadku niektórych formatów) dopisać go po prostu na końcu pliku.
- » Przetworzenie obrazka i zapisanie pliku o nowej zawartości. Istnieją jednak znane ataki, umożliwiające umieszczenie kodu PHP po przetworzeniu pliku PNG. Należy też mieć na uwadze, że mogą występować błędy w poszczególnych implementacjach (np. znany jest błąd w funkcjach `imagefromgif` oraz `imagefromjpeg` w PHP, pozwalający zachować część złośliwego kodu PHP).

Spojrzenie client-side

1. Niewłaściwa biała lista

Biała lista dopuszczalnych rozszerzeń obrazków może zawierać .svg, który *de facto* jest zwykłym plikiem XML, parsowanym przez przeglądarkę. Wówczas można użyć kodu wcześniej wspomianego w tym artykule i wykonać dzięki temu XSS-a.

2. Zuploadowanie pliku Flasha

Pliki .swf umożliwiają pewien ciekawy, specyficzny atak w przypadku, gdy aplikacja weryfikuje rozszerzenia, jednak nie sprawdza zawartości pliku. We Flashu obowiązują trochę inne zasady *Same Origin Policy* niż w przypadku samego JavaScriptu. Jeśli mamy przykładowo plik .swf zahostowany na domenie www.google.com, wówczas niezależnie od tego, z jakiej domeny jest on wywoływany, może pobierać dowolne zasoby (metodami POST i GET) z domeny www.google.com. Wcześniej wspomnianym skryptem `xss.swf` można potwierdzić, że taki atak rzeczywiście się powiedzie.

Co więcej, nawet jeśli plik ma rozszerzenie .jpg, można przekonać przeglądarkę, aby interpretowała go jako plik Flasha. Założmy, że na domenie victim.com udało się zahostować plik `xss.png`, którym w rzeczywistości jest `xss.swf`. Wówczas na domenie kontrolowanej przez atakującego należy dodać kod:

```
1 <embedallowscriptaccess="always"  
src="http://www.victim.com/uploads/xss.png?a=get&c=  
http://www.victim.com/iny_zasob" type="application/x-shockwave-flash">
```

Jeśli wszystko się powiedzie, powinniśmy otrzymać odpowiedź „cross domain request ok.” (literówka zamierzona, taka sama jest w kodzie skryptu).

METODY OCHRONY

Dotychczas skupiliśmy się wyłącznie na różnych metodach realizacji uploadu w aplikacjach webowych. W jaki sposób jednak powinniśmy się bronić?

Wszystkie ataki (czy to server-side, czy client-side) związane były z tym, że pobrane na serwer pliki znajdują się na tym samym hoście oraz w tej samej domenie, co aplikacja webowa. Jeśli tylko jest to możliwe, należy wydzielić osobną domenę (np. jeśli aplikacja działa w domenie www.example.com, to pliki mogą być hostowane w www.example-files.com), jak również osobnąinstancję serwera wyłącznie do uploadów. Dzięki temu zniwelowane zostanie ryzyko związane z wykonywaniem kodu, jak również wszelkie ataki typu XSS będą odbywały się na osobnej domenie.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



Nie zawsze jednak mamy środki i możliwości, by wprowadzić takie rozwiązanie.

Wówczas należy pamiętać o kilku podstawowych regułach:

- » Pliki uploadowane przez użytkowników powinny być zapisywane poza webrootem.
- » Pliki nie powinny być zapisywane pod taką samą nazwą pliku, jaką wysłał użytkownik. Najlepiej użyć losowego hasza. Oryginalna nazwa pliku może być przechowywana w bazie danych. Alternatywnie cały plik może być przechowywany w bazie.
- » Do odpowiedzi HTTP z pobieraniem plików wgranych wcześniej należy dodać nagłówki:
 - » **Content-Disposition: attachment** – dzięki któremu przeglądarka wyświetli monit z pobieraniem pliku,
 - » **X-Content-Type-Options: nosniff** – dzięki któremu zostaną uniemożliwione pewne ataki związane ze zgadywaniem typu dokumentu przez przeglądarki (dotyczy to w szczególności Internet Explorera).



Michał Bentkowski. Realizuje testy penetracyjne oraz audyty bezpieczeństwa w firmie [Securitum](#). Autor w serwisie sekurak.pl. Aktywnie (i z sukcesem) uczestniczy w znanych programach bug bounty.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



BEZPIECZEŃSTWO SYSTEMÓW IT



Zapraszamy na autorskie szkolenia z zakresu **bezpieczeństwa IT**

{ Bezpieczeństwo aplikacji WWW }

{ Offensive HTML, SVG, CSS and other Browser-Evil }

{ Bezpieczeństwo sieci / testy penetracyjne }

{ Wprowadzenie do bezpieczeństwa IT }

{ Zaawansowany monitoring i obsługa
incydentów bezpieczeństwa informacji }

{ Szkolenie przygotowujące do egzaminu CEH
(Certified Ethical Hacker) }

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Ukryte katalogi oraz pliki pozostawione przez nieuwagę na serwerze WWW, mogą stać się nieocenionym źródłem informacji podczas testu penetracyjnego. W skrajnych sytuacjach, takich jak pozostawiony katalog .git lub .svn, pozwalają na dostęp do kodu źródłowego aplikacji, co w rezultacie skutkuje całkowitą kompromitacją i uzyskaniem nieautoryzowanego dostępu oraz możliwością przeprowadzenia statycznej analizy kodu źródłowego i odkrycia wszystkich luk.

Wbrew pozorom, bardzo często można natknąć się na tego rodzaju zasoby. Poniższy artykuł na przykładzie kilku najpopularniejszych katalogów prezentuje praktyczne przykłady wykorzystania informacji w nich zawartych.

SYSTEMY KONTROLI WERSJI

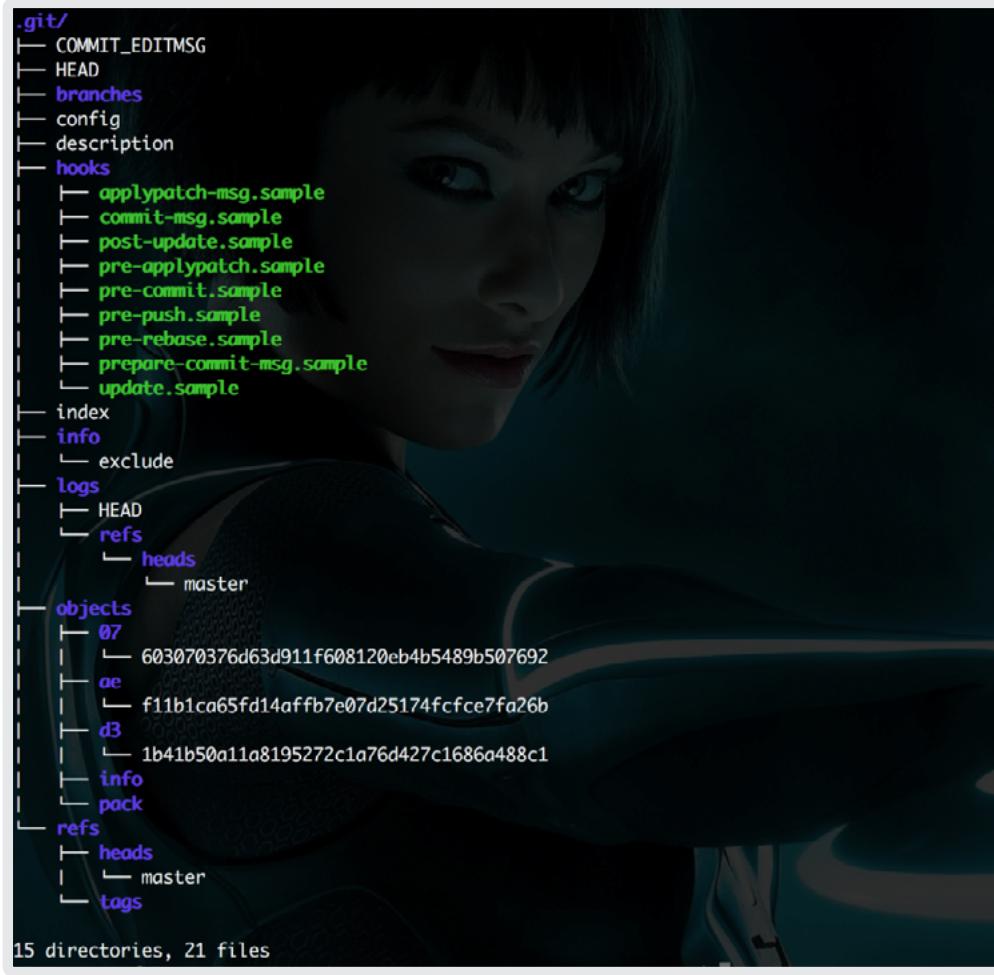
Git (<https://git-scm.com/>) to jeden z powszechnie wykorzystywanych rozproszonych systemów kontroli wersji. Jego popularność dodatkowo zwiększa serwisy takie jak GitHub czy Bitbucket.

Podstawowe informacje o obiektach Gita

Wszystkie informacje na temat projektu pod kontrolą Gita znajdują się w folderze .git, w głównym katalogu. Jego przykładową strukturę przedstawia rysunek 1.

Przyjrzyjmy się zawartym w nim informacjom z punktu widzenia crackera. W folderze .git/objects znajdują się pliki zawierające wszystkie operacje dokonywane na repozytorium, a także zawartość wszystkich plików w takiej postaci, w jakiej znajdowały się w momencie dokonywania jakiekolwiek zmiany, np. operacji *commit*. Nazwami obiektów są 40-znakowe skróty (hasze) SHA-1. Każdy obiekt może być jednym z trzech poniższych typów:

- » *commit* – zawiera informacje na jego temat, takie jak: autor, komentarz oraz hasze obiektów aktualnego drzewa katalogów i plików projektu;
- » *tree* – zawiera skrót (hasz) obiektu przechowującego strukturę plików i katalogów;
- » *blob* – zawiera zawartość pliku zapisaną w postaci binarnej, możliwą do odczytania polecienniem *git cat-file -p [hasz SHA1]*.



Rysunek 1. Zawartość przykładowego katalogu .git.

Jeżeli programista pozostawił na serwerze folder .git, nic nie stoi na przeszkodzie, aby odczytać zawartość dowolnego pliku, nawet jeśli nie mamy uprawnień do pobrania całego repozytorium polecienniem *git clone git checkout*.

Jak sprawdzić, czy folder .git znajduje się na serwerze?

Wystarczy sprawdzić, czy url w postaci np. <http://adresserwisu/.git> zwróci nam odpowiedź HTTP z kodem innym, niż 404 Not Found. Z reguły, konfiguracja serwera nie zezwala na listing zawartości katalogów, wobec czego otrzymujemy odpowiedź 403 Forbidden, co jednoznacznie wskazuje na to, że trafiliśmy na naszą „informacyjną żyłę złota”:

Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

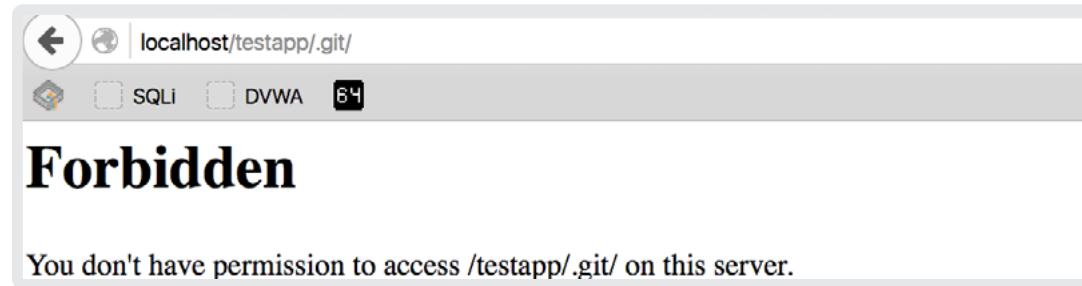
[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)

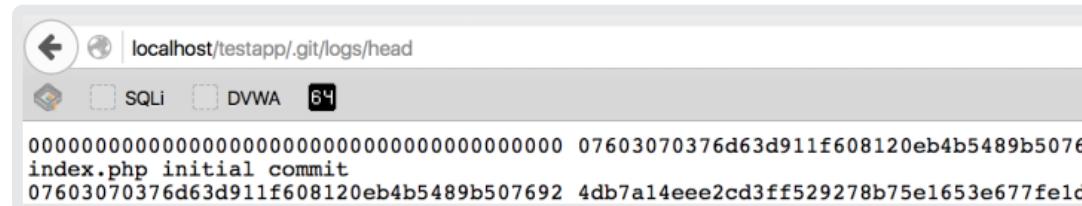




Rysunek 2. Odpowiedź HTTP 403 serwera wskazująca na obecność katalogu .git.

Wiemy, że .git znajduje się na serwerze, ale nie wiemy, jakie skróty SHA-1 identyfikują poszczególne obiekty. Jak uzyskać taką informację?

Cała historia operacji (dostępna po wykonaniu polecenia `git log`), zapisywana jest w pliku `.git/logs/head`. Jest to zwykły plik tekstowy, zawierający informacje o wszystkich commitach:



Rysunek 3. Zawartość pliku .git/logs/head.

Przyjrzyjmy się bliżej pierwszemu wpisowi w tym pliku:

```
00000000000000000000000000000000 07603070376d63d911f608120eb4b5489b507692  
bloorq@gmail.com <bloorq@gmail.com> 1452195279 +0000 commit (initial): index.php  
initial commit
```

Pierwsze dwa hasze to kolejno: poprzedni i aktualny commit. Ponieważ jest to wpis dotyczący pierwszego commita w tym repozytorium, pierwszy hasz to po prostu same zera.

Zanim zaczniemy pobierać obiekty Gita identyfikowane przez odnalezione hasze, dla ułatwienia stworzymy szkielet repozytorium Git. W tym celu, w konsoli należy wydać polecenie:

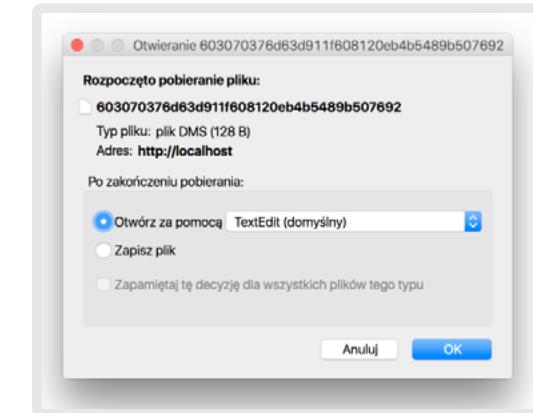
```
$ git init
```

Wynikiem będzie katalog `.git` zawierający szkielet repozytorium, w tym folder `objects`, gdzie zapisywać będziemy pobrane obiekty. Najpierw musimy jednak odwzorować hasz obiektu na fizyczną ścieżkę w zdalnym repozytorium. Ścieżka składa się kolejno:

1. ze stałego fragmentu – `git/objects/`,
2. z dwuliterowej nazwy folderu, będącej dwoma pierwszymi znakami hasza obiektu,
3. z nazwy pliku utworzonej z pozostałych 38 znaków hasza:

```
http://localhost/testapp/.git/objects/07/603070376d63d911f608120eb4b5489b507692
```

Po otwarciu w przeglądarce powyższego adresu, powinniśmy ujrzeć okno dialogowe pobierania pliku:



Rysunek 4. Okno dialogowe pobierania pliku w przeglądarce Firefox.

Plik zapisujemy w naszym tymczasowym repozytorium Git, zachowując prawidłową ścieżkę – w folderze `.git/objects/07` – jako plik o nazwie `603070376d63d911f608120eb4b5489b507692`. Aby móc odwoływać się do tego obiektu w poleceniach, musimy pamiętać o posługiwaniu się pełnym 40-znakowym haszem.

Typ obiektu możemy sprawdzić poleceniem `git cat-file -t`:

```
$ git cat-file -t 07603070376d63d911f608120eb4b5489b507692
```

Polecenie `git cat-file -p` pozwala na sprawdzenie zawartości obiektu:

```
$ git cat-file -p 07603070376d63d911f608120eb4b5489b507692
```

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



```
~/hacking/research -- bash /Library/WebServer/Documents/testapp -- bash ~/hacking/playground/diggit/test1 -- bash
bl4de on Rafals-MacBook in /Library/WebServer/Documents/testapp $ git cat-file -t 07603070376d63d911f608120eb4b5489b507692
commit
bl4de on Rafals-MacBook in /Library/WebServer/Documents/testapp $ git cat-file -p 07603070376d63d911f608120eb4b5489b507692
tree d31b41b50a11a8195272c1a76d427c1686a488c1
author bloorq@gmail.com <bloorq@gmail.com> 1452195279 +0000
committer bloorq@gmail.com <bloorq@gmail.com> 1452195279 +0000

index.php initial commit
bl4de on Rafals-MacBook in /Library/WebServer/Documents/testapp $
```

Rysunek 5. Wynik wykonania polecień git cat-file w oknie konsoli.

Dokładna analiza treści commita pozwoli nam na uzyskanie kolejnej, istotnej informacji: jaki jest hasz reprezentujący drzewo katalogów i plików w repozytorium. Będzie to informacja dotycząca stanu, w jakim znajdowało się ono w momencie wykonania tego commita, a nie aktualnego stanu.

Przykład poniżej:

```
~/hacking/research -- bash /Library/WebServer/Documents/testapp -- bash ~/hacking/playground/diggit/test1 -- bash
bl4de on Rafals-MacBook in /Library/WebServer/Documents/testapp $ git cat-file -p d31b41b50a11a8195272c1a76d427c1686a488c1
100644 blob aef11b1ca65fd14affb7e07d25174fcfce7fa26b index.php
bl4de on Rafals-MacBook in /Library/WebServer/Documents/testapp $
```

Rysunek 6. Hasz odpowiadający plikowi index.php w bieżącym drzewie katalogów i plików.

Jak widzimy, po pierwszym commicie w repozytorium znajdował się tylko jeden plik. Mamy również informację o haszu, jego typie (jest to *blob*, więc obiekt zapisany pod taką nazwą zawiera już dane – w tym wypadku będzie to kod źródłowy pliku index.php):

```
~/hacking/research -- bash /Library/WebServer/Documents/testapp -- bash ~/hacking/playground/diggit/test1 -- bash
<?php
echo "Hello testapp!";
bl4de on Rafals-MacBook in /Library/WebServer/Documents/testapp $
```

Rysunek 7. Kod źródłowy pliku index.php.

Bingo!

Jak wyżej wspomniałem, zawartość pliku index.php, jaką udało nam się odtworzyć, jest aktualna w momencie zatwierdzania pierwszego commita z odnalezionej repozytorium. Gdy przyjrzymy się plikowi logu, widzimy, że to nie był ostatni commit, i w kolejnym mogły nastąpić jakieś zmiany (w praktyce rzadko będzie nas interesowała zawartość starszych commitów – ostatni zawiera hasz aktualnego drzewa katalogów, które z kolei pozwoli nam uzyskać hasze aktualnych wersji plików).

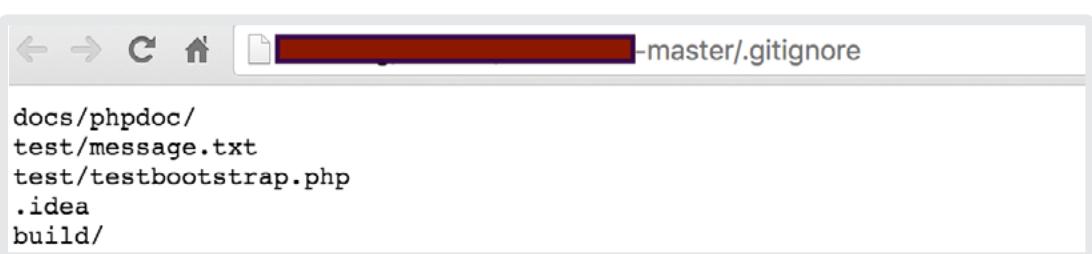
Sprawdźmy więc, czy commit *4db7a14eee2cd3ff529278b75e1653e677fe1d02* zwróci nam inną zawartość pliku index.php. Postępując w dokładnie ten sam sposób, co wyżej (rysunki 1–6), ostatecznie uzyskujemy kod źródłowy wskazujący, że faktycznie aktualna zawartość pliku jest inna:

```
$ git cat-file -p a4215057b6545240452087ad4d015bf9b5b817c5
<?php
echo "Hello testapp!";
$i = 100;
echo "Value of i is $i";
```

Powyższy przykład prezentuje jedną z kilku dostępnych metod uzyskania dostępu do kodu źródłowego aplikacji webowej. Jednak niezależnie od metody, pozostawanie folderu *.git* na serwerze, na którym znaleźć się nie powinien, na przykład serwerze WWW, oznacza katastrofę z punktu widzenia bezpieczeństwa.

Plik .gitignore

Jeśli uda nam się odnaleźć folder *.git*, bardzo prawdopodobnym jest, że znajdziemy również plik *.gitignore* – jego przeznaczeniem jest wskazanie, które foldery i pliki Git ma zignorować (czyli nie będą one uwzględniane w ramach katalogu roboczego Gita, commitowane itp.). Z punktu widzenia pentestera, jest to po prostu lista folderów i plików, które najprawdopodobniej znajdują się na serwerze, lecz nie będą osiągalne opisaną wyżej metodą:



Rysunek 8. Przykładowy plik .gitignore.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



SUBVERSION (SVN)

Subversion (lub SVN) to kolejny bardzo popularny system kontroli wersji, rozwijany w ramach Apache Software Foundation (<https://subversion.apache.org/>). Podobnie jak opisany powyżej Git, SVN zapisuje informacje o aktualnym katalogu roboczym w ukrytym folderze o nazwie `.svn`.

Przykładowy folder z informacjami o repozytorium przedstawia poniższy zrzut ekranu:

```
bl4de on Rafals-MacBook in /Library/WebServer/Documents/project_wombat/.svn $ tree .
.
├── entries
├── format
└── pristine
    ├── 6f
    │   └── 6f3fb98418f14f293f7ad55e2cc468ba692b23ce.svn-base
    ├── 94
    │   └── 945a60e68acc693fc74ababd588aac1a9135f62.svn-base
    ├── 9f
    │   └── 9fbe0a5122c313baeba8d447c777325a39906a50.svn-base
└── tmp
└── wc.db

5 directories, 6 files
bl4de on Rafals-MacBook in /Library/WebServer/Documents/project_wombat/.svn $
```

Rysunek 9. Struktura katalogu `.svn`.

Z naszego punktu widzenia, najważniejszy jest plik systemu bazodanowego SQLite `wc.db` oraz zawartość folderu `pristine`. To tam znajdziemy wszystkie interesujące nas informacje.

Zaczniemy od pliku `wc.db`. Jeśli po otwarciu w przeglądarce adresu:

http://server/path_to_vulnerable_site/.svn/wc.db

ukaże nam się okno pobierania pliku, oznacza to, że mamy dostęp do informacji zapisanych przez SVN w katalogu roboczym. Aby odczytać informacje zawarte w pobranej bazie danych, najlepiej posłużyć się klientem SQLite:

Listing 1. Odczytanie zawartości bazy danych przy pomocy konsolowego klienta SQLite.

```
$ sqlite3 wc.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite> .databases
sqlite> seq name file
-----
0 main /Users/bl4de/hacking/playground/wc.db
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE REPOSITORY ( id INTEGER PRIMARY KEY AUTOINCREMENT, root TEXT UNIQUE
NOT NULL, uuid TEXT NOT NULL );
INSERT INTO "REPOSITORY" VALUES(1,'svn+ssh://192.168.1.4/var/svn-repos/
project_wombat','88dcec91-39c3-4b86-8627-702dd82cfa09');

(...)

INSERT INTO "NODES" VALUES(1,'trunk',0,'',1,'trunk',1,'normal',NULL,NULL,'dir',X'
2829','infinity',NULL,NULL,1,1456055578790922,'bl4de',NULL,NULL,NULL);
INSERT INTO "NODES" VALUES(1,'',0,NULL,1,'',1,'normal',NULL,NULL,'dir',X'2829','i
nfinity',NULL,NULL,1,1456055578790922,'bl4de',NULL,NULL,NULL);
INSERT INTO "NODES" VALUES(1,'trunk/test.txt',0,'trunk',1,'trunk/test.txt',2,'nor
mal',NULL,NULL,'file',X'2829',NULL,
'$sha1$945a60e68acc693fc74ababd588aac1a9135f62',
NULL,2,1456056344886288,'bl4de',38,1456056261000000,NULL,NULL);
INSERT INTO "NODES" VALUES(1,'trunk/test2.txt',0,'trunk',1,'trunk/test2.txt',3,'n
ormal',NULL,NULL,'file',NULL,NULL,'$sha1$6f3fb98418f14f293f7ad55e2cc468ba692b23ce
',NULL,3,1456056740296578,'bl4de',27,1456056696000000,NULL,NULL);

(...)
```

Operacje INSERT do tabeli NODES zawierają hasze SHA-1 (podobnie, jak w przypadku Gita) oraz informację, którego pliku dotyczą. Pliki zapisane pod postacią haszy znajdują się w folderze `pristine` – jeśli mamy informacje wydobyte z bazy `wc`, nic nie stoi już na przeszkodzie, by pobrać je na dysk naszego komputera.

Aby zmapować hasz z: `$sha1$945a60e68acc693fc74ababd588aac1a9135f62` do fizycznej ścieżki na serwerze zdalnym, musimy wykonać kilka prostych operacji:

1. należy najpierw usunąć przedrostek `$sha1$`;
2. następnie pozostałą część uzupełnić o przyrostek `.svn-base`;
3. dwa pierwsze znaki hasza to folder w katalogu `pristine` (podobnie, jak w przypadku Gita i ścieżki do folderu `.git/objects/XX`);
4. ostatnim krokiem jest utworzenie kompletnego adresu url do pliku na serwerze:

http://server/path_to_vulnerable_site/.svn/pristine/94/945a60e68acc693fc74ababd588aac1a9135f62.svn-base

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

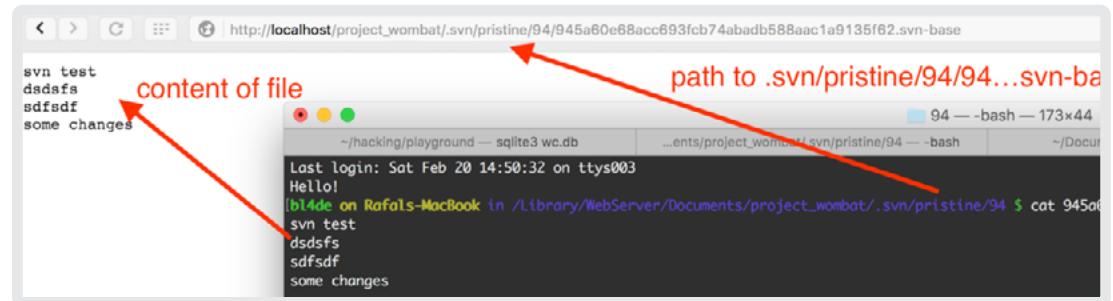
Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



Gdy użyjemy powyższego adresu w przeglądarce, powinniśmy uzyskać możliwość pobrania i zapisania pliku, bądź odczytania go bezpośrednio w oknie przeglądarki:



Rysunek 10. Ustalenie ścieżki do pliku w folderze .svn pozwala na wyświetlenie jego treści w przeglądarce.

Dodatkową informacją uzyskaną z bazy danych `wc.db` może być też adres do repozytorium centralnego, zapisany w tabeli `REPOSITORIES`:

```
svn+ssh://192.168.1.4/var/svn-repos/project_wombat
```

Podobnie jak w przypadku Gita, pozostawienie katalogu `.svn` na serwerze oznacza dla właściciela serwisu katastrofę w przypadku jego odkrycia przez cyberprzestępów chcących skompromitować aplikację bądź serwis internetowy.

W obu opisanych sytuacjach (SVN i Git), ani technologia, w jakiej serwis została napisany, ani użyte w kodzie zabezpieczenia, nie mają znaczenia.

Należy mieć świadomość, że dostęp do kodu źródłowego oznacza z reguły dostęp do danych dostępnych do serwerów bazodanowych, z którymi aplikacja się komunikuje, czy jakichkolwiek innych zasobów.

Analiza kodu źródłowego może też umożliwić wykorzystanie luk, których odkrycie w tradycyjny sposób nie byłoby możliwe (np. błędy w zabezpieczeniu uploadu plików czy odkrycie „tylnych furtek” pozostawionych przez programistów w celach diagnostycznych).

KATALOGI I PLIKI KONFIGURACYJNE ŚRODOWISK PROGRAMISTYCZNYCH

IDE (*Integrated Development Environment*) – zintegrowane środowiska programistyczne wykorzystywane przez wielu developerów aplikacji internetowych, mają

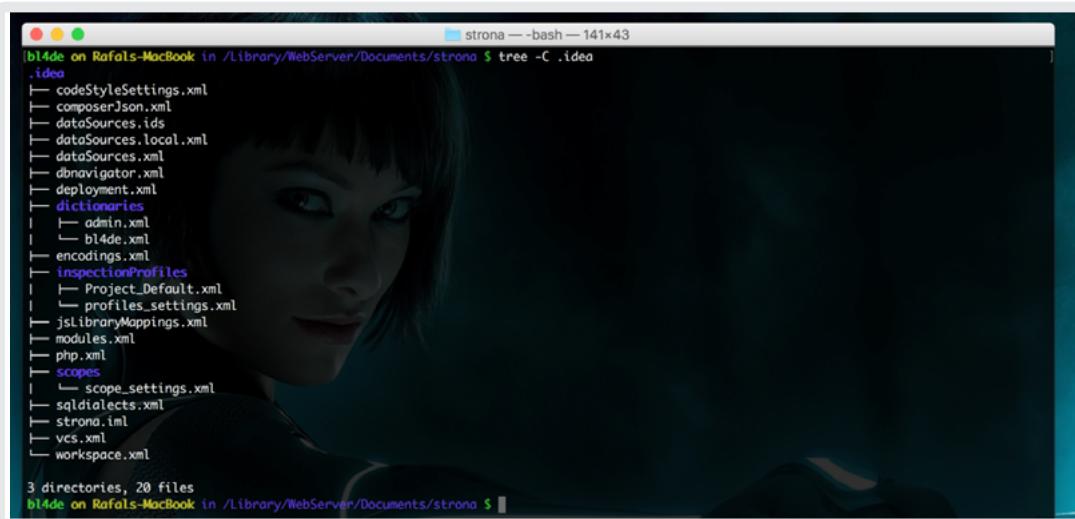
jedną wspólną cechę – podobnie jak systemy kontroli wersji, zapisują wiele informacji na temat projektu oraz konfiguracji samego środowiska w pewnych, charakterystycznych dla siebie lokalizacjach. Z reguły te informacje nie są dostępne na serwerach produkcyjnych, ale zdarza się, że – podobnie jak w przypadku Gita czy SVN – mniej doświadczeni lub nieuwazni deweloperzy, pozostawiają takie foldery i pliki na ogólnie dostępnych serwerach.

JetBrains IDE – PHPStorm, WebStorm, PyCharm, IntelliJ IDEA

Środowiska programistyczne z czeskiej „stajni” JetBrains (<http://www.jetbrains.com>), to bardzo popularne i cenione na całym świecie produkty. Poza dość zunifikowanym interfejsem, ustawieniami oraz ogólną filozofią działania niezależnie od platformy programistycznej, ich wspólną cechą jest folder `.idea`, w którym zapisują wszystkie informacje związane z projektem oraz ustawieniami samego IDE.

Szczególnie jeden z plików znajdujących się w tym folderze jest bardzo wartościowy z punktu widzenia cyberprzestępcy lub pentestera: `workspace.xml`. Zawiera on wszystkie informacje, które pozwalają na łatwe odtworzenie struktury plików i katalogów aplikacji bez potrzeby uciekania się do narzędzi typu `DirBuster`.

Na poniższym zrzucie ekranu przedstawione jest przykładowe drzewo plików i katalogów w folderze `.idea`. Poza `workspace.xml` jest tam jeszcze kilka plików, których analiza może dostarczyć wielu wartościowych informacji na temat projektu.



Rysunek 11. Zawartość folderu `.idea` środowiska PHPSt.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



Przyjrzyjmy się plikowi *workspace.xml* dokładniej:

Listing 2. Plik workspace.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
(...)

<component name="FileEditorManager">
<leaf>
<file leaf-file-name="README.md" pinned="false" current-in-tab="false">
<entry file="file://$/PROJECT_DIR$/README.md">
(...)
</component>
(...)
```

Wszystkie węzły znajdujące się w elemencie *FileEditorManager* zawierają relatywne ścieżki do wszystkich plików wchodzących w skład projektu. Upraszczając – jest to XML-owa wersja wyniku wykonania polecenia *ls -l* w głównym katalogu projektu.

Bliższa analiza każdego z tych węzłów pozwala znaleźć informacje np. o użytym systemie kontroli wersji (co może nas nakierować na użycie opisanych wcześniej metod uzyskiwania informacji z takich systemów):

```
<component name="Git.Settings">
<option name="UPDATE_TYPE" value="MERGE" />
<option name="RECENT_GIT_ROOT_PATH" value="$PROJECT_DIR$" />
</component>
```

Znajdziemy również dane na temat commitów do tych systemów:

```
(...)
<task id="LOCAL-00211" summary="change WebSocket port to 1099">
<created>1436206418000</created>
<option name="number" value="00211" />
<option name="project" value="LOCAL" />
<updated>1436206418000</updated>
</task>
(...)
```

a także informacje o historii lokalnej zmian w projekcie (historii zapisanej na komputerze programisty, a nie w systemie kontroli wersji):

```
<component name="ChangeListManager">
(...)
<change type="DELETED" beforePath="$PROJECT_DIR$/chat/node_modules/socket.io/
node_modules/socket.io-adapter/node_modules/debug/Makefile" afterPath="" />
(...)
</component>
```

Jeśli programista do zarządzania bazą danych używa wbudowanego w IDE JetBrains menedżera połączeń bazodanowych oraz klienta SQL, również informacje na temat połączeń z serwerami bazodanowymi są dla nas dostępne z poziomu plików konfiguracyjnych IDE (*dataSources.ids*, *dataSource.xml*, *dataSources.xml*, *dataSources.local.xml*, *dbnavigator.xml*), przykładowo *dbnavigator.xml*:

Listing 3. Plik dbnavigator.xml.

```
<database>
<name value="database_name" />
<description value="" />
<database-type value="MYSQL" />
<config-type value="BASIC" />
<database-version value="5.7" />
<driver-source value="BUILTIN" />
<driver-library value="" />
<driver value="" />
<host value="localhost" />
<port value="3306" />
<database value="mywebapp" />
<url-type value="DATABASE" />
<os-authentication value="false" />
<empty-password value="false" />
<user value="root" />
<password value="cm9vdA==" /> <!-- tak, to jest hasło ,root' i Base64 :-->
</database>
```

czy *dataSources.local.xml*:

Listing 4. Fragment pliku dataSources.local.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
<component name="dataSourceStorageLocal">
<data-source name="MySQL - mywebapp@localhost"
uuid="8681098b-fc96-4258-8b4f-bfb00012e2b">
<secret-storage>master_key</secret-storage>
<user-name>root</user-name>
<schema-pattern>mywebapp.*</schema-pattern>
<default-schemas>mywebapp.*</default-schemas>
</data-source>
</component>
</project>
```

Ilość i zawartość tych plików zależy od użytych w IDE wtyczek, konfiguracji i wielu innych czynników. Najlepszą metodą zbadania zawartości w katalogu *.idea*, jest:

1. pobranie i zainstalowanie jednego z środowisk IDE na własnym komputerze;

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning

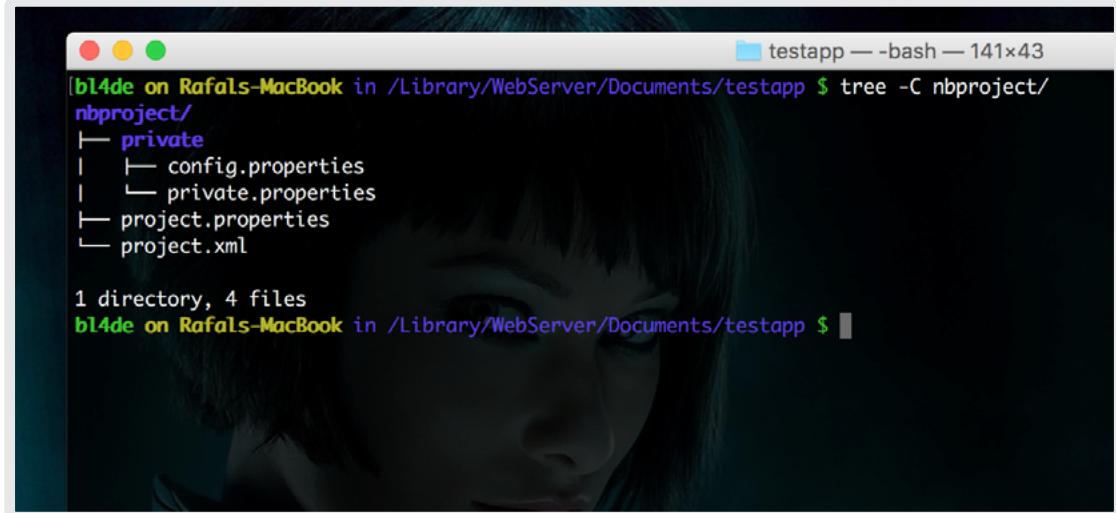


2. utworzenie przykładowego projektu;
3. użycie kilku pluginów lub wbudowanych klientów do systemów kontroli wersji lub zarządzania bazami danych;
4. obserwacja, jakie informacje/dane pojawiają się w plikach w tym folderze w trakcie pracy z edytorem i wykonywania standardowych operacji, jak utworzenie nowego pliku i zapisanie go na dysk.

NetBeans IDE

NetBeans (<https://netbeans.org/>) to kolejne, bardzo popularne IDE, będące darmowym oprogramowaniem wspieranym przez firmę Oracle. W porównaniu z IDE, JetBrains nie jest aż tak bardzo pomocne i nie udostępnia aż tak wielu informacji w folderze z konfiguracją projektu i samego środowiska zapisanym bezpośrednio w projekcie.

Katalog o nazwie *nbproject* jest dość skromny, ale i tak analiza pliku *project.xml* może dostarczyć kilku cennych wskazówek i udzielić odpowiedzi na kilka podstawowych pytań odnośnie użytych w projekcie technologii.



```
testapp — bash — 141x43
[b14de on Rafals-MacBook in /Library/WebServer/Documents/testapp $ tree -C nbproject/
nbproject/
└── private
    ├── config.properties
    └── project.properties
    └── project.xml
1 directory, 4 files
b14de on Rafals-MacBook in /Library/WebServer/Documents/testapp $
```

Rysunek 12. Przykładowa zawartość folderu *nbproject*.

ActiveState Komodo IDE

Stosunkowo mało znane i nie tak popularne, jak poprzednie opisywane IDE – Komodo z ActiveState – spośród wymienionych rozwiązań, udostępnia najmniej

nieuprawnionych informacji i w katalogu projektu nie przechowuje praktycznie żadnych danych istotnych z punktu widzenia cyberprzestępcy czy pentestera.

Postanowiłem jedynie wspomnieć o tym rozwiązaniu – dotychczas nie miałem okazji analizować zawartości większego projektu w tym IDE – sugeruję jednak, że w określonych warunkach, możliwe jest uzyskanie istotnych informacji, m.in. w trakcie analizy pliku konfiguracyjnego.

Plik konfiguracyjny składa się z nazwy projektu oraz przyrostka *.komodoproject*:

```
$ ls -l | grep komodo
-rw-r--r-- 1 b14de staff 300 Feb 4 23:19 bwapp.komodoproject
```

PLIKI KONFIGURACYJNE NARZĘDZI DEWELOPERSKICH

W ostatnim czasie wraz z rosnącą popularnością wszechobecnego JavaScript w katalogach projektów jak grzyby po deszczu zaczęły pojawiać się pliki kończące się z reguły na „-rc” i zaczynające od znaku kropki. Pojawiło się również sporo plików JSON, które osobie nie zorientowanej w temacie niewiele powiedzą, ale dla wprawnego cyberprzestępcy lub pentestera mogą stać się cennym źródłem informacji o projekcie i zastosowanych technologiach, frameworkach czy bibliotekach. Użycie narzędzi takich, jak *DirBuster* (<http://sourceforge.net/projects/dirbuster/>) nie zawsze pozwala na odnalezienie takich plików, dlatego warto wiedzieć, gdzie i czego szukać.

bower.json, package.json

Bower to biblioteka umożliwiająca bezproblemową instalację i aktualizację – wraz z zależnościami, bibliotek i frameworków używanych przez web developerów w aplikacjach napisanych w JavaScript.

Przykładowy plik *bower.json* przedstawiony jest poniżej:

Listing 5. Plik *bower.json*.

```
{
  "name": "testapp",
  "version": "2.1.0",
  "authors": [
    "Rafał 'bl4de' Janicki <bloorq@gmail.com>"
  ],
  "description": "test application",
  "main": "index.html",
  "moduleType": [
    "globals"
  ]}
```

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



```
],  
"license": "MIT",  
"dependencies": {  
  "angular": "1.4",  
  "pure": "~0.5.0",  
  "angular-route": "~1.2.26",  
  "angular-ui-router": "~0.2.11",  
  "angular-bootstrap-datetimepicker": "latest",  
  "angular-translate": "~2.6.1"  
},  
"devDependencies": {}  
}
```

Bardziej interesujące z punktu widzenia atakującego lub pentestera będą niewątpliwie informacje o technologiach użytych po stronie serwera (*node.js* lub *io.js* – <https://nodejs.org/en/>, <https://iojs.org/en/>) – plik *package.json*:

```
{  
  "name": "Test application server dependencies",  
  "version": "1.0.0",  
  "author": "bl4de",  
  "dependencies": {  
    "socket.io": "^1.3.5",  
    "mysql": "^2.9.0"  
  }  
}
```

W przypadku odkrycia luki typu LFI (Local File Inclusion, czyli atak umożliwiający odczytanie dowolnego pliku znajdującego się na serwerze poprzez dołączenie jego zawartości do źródła strony bądź aplikacji WWW) i odczytania tego pliku, atakujący zdobędzie informacje np. o użytych pakietach *npm* (<https://www.npmjs.com/>) do obsługi połączeń poprzez mechanizm *WebSocket* czy też zastosowanego systemu bazodanowego (w tym wypadku *MySQL*).

Warto również poszukać plików takich jak *.bowerrc*, *.eslintrc* czy *jshintrc* – każdy z nich może zawierać wskazówkę, jakie technologie mogły zostać użyte w aplikacji, odkryć skonfigurowane na potrzeby deweloperskie ustawienia, bądź połączenia do innych, niemożliwych do zidentyfikowania z poziomu przeglądarki lub skanera sieciowego – zasobów.

PODSUMOWANIE

Jako programista aplikacji internetowych, z doświadczeniem wiem, jak wiele informacji na temat projektu jest przechowywanych w zasobach zupełnie nie związanych

z samą aplikacją czy serwisem internetowym. Mogą one w banalny sposób udostępnić informacje, które byłyby nie do uzyskania innymi tradycyjnymi metodami.

Przykładowo – kod w pliku PHP jest niedostępny na prawidłowo skonfigurowanym serwerze *Apache* czy *nginx*, nawet jeśli znamy bezwzględną ścieżkę do tego pliku. Pozostawienie folderu systemu kontroli wersji na serwerze sprawia, że możemy bez trudu odczytać zawartość pliku i żadne zabezpieczenia już tutaj nie pomogą.

Z punktu widzenia programisty – bardzo ważny jest nadzór nad zawartością umieszczaną na serwerze. Niedopuszczalne jest, by w środowisku produkcyjnym znalazł się np. folder *.idea* czy zawartość typowych folderów „roboczych”, typu: *tmp*, *temp*, *dev*, *backup*, *debug*, *log*, *logs* i tym podobne, często wykorzystywane w trakcie pracy nad aplikacją, lokalizacje.

Z punktu widzenia atakującego – dobrze jest mieć świadomość tego, jak nowoczesne, wspomagające programistów w ich codziennej pracy narzędzia, mogą przez nieuwagę czy zwykłe niedbalstwo – stać się źródłem wycieku informacji i najslabszymogniwem w nawet najlepiej zabezpieczonej aplikacji webowej.

Rafał 'bl4de' Janicki. Od wielu lat związany z aplikacjami internetowymi, od kilku jako HTML5/JavaScript Developer z doświadczeniem w dużych korporacjach. Interesuje się także tematyką bezpieczeństwa aplikacji internetowych.



Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning





CYBERSEC PL

POLSKIE FORUM CYBERBEZPIECZEŃSTWA

8 KWIETNIA 2016, WARSZAWA

POTWIERDZENI PRELEGENCI



ANNA STREŻYNSKA

Minister Cyfryzacji



TOMASZ SZATKOWSKI

Podsekretarz Stanu
w Ministerstwie Obrony Narodowej



TOMASZ ZDZIKOT

Podsekretarz Stanu
w Ministerstwie Spraw
Wewnętrznych i Administracji



MACIEJ PYZNAR

Główny Specjalista
w Rządowym Centrum
Bezpieczeństwa



BOGDAN ŚWIĘCZKOWSKI

I Zastępca Prokuratora
Generalnego,
Prokurator Krajowy

CZYM JEST CYBERSEC PL?

CYBERSEC PL jest corocznym wydarzeniem o charakterze public policy conference, poświęconym strategicznym aspektom cyberbezpieczeństwa Polski. Konferencja skierowana jest do polskich instytucji publicznych, jak również do przedstawicieli sektora prywatnego i środowisk eksperckich.

MISJA

Misją CYBERSEC PL jest budowanie narodowych zdolności służących wzmacnianiu krajowego systemu cyberbezpieczeństwa, pozwalających na prowadzenie skutecznych i suwerennych działań w cyberprzestrzeni.



ŚCIEŻKA PAŃSTWO

Poszukiwanie skutecznych rozwiązań, mających na celu ochronę infrastruktury krytycznej – wzmacnianie roli państwa, współpraca publiczno-prywatna.

CZTERY ŚCIEŻKI TEMATYCZNE



ŚCIEŻKA WOJSKO

Zwiększenie roli polskiej armii w zapewnianiu cyberbezpieczeństwa, lipcowy szczyt NATO w Warszawie i konieczność podjęcia dalszych kroków w polityce cyberbezpieczeństwa Sojuszu.



ŚCIEŻKA PRZYSZŁOŚĆ

Podnoszenie świadomości w zakresie rosnącej luki zatrudnienia w dziedzinie IT i cyberbezpieczeństwa oraz wyznaczenie priorytetów systemu edukacji w tym obszarze.



ŚCIEŻKA BIZNES

Wyzwania dla Polski w zakresie cyfryzacji, w tym promocja sektora nowych technologii, a także kwestia wzmacniania innowacyjności polskiego przemysłu w obszarze cyberbezpieczeństwa.

ORGANIZATOR



WŚRÓD PATRONÓW HONOROWYCH



REJESTRACJA

Administracja Publiczna, Służby Mundurowe,
Wykładowcy uczelni wyższych
BEZ OPŁAT

ZAREJESTRUJ SIĘ TERAZ!

CYBERSEC PL. MYŚLIMY STRATEGICZNIE
O NARODOWYM CYBERBEZPIECZEŃSTWIE

WWW.CYBERSECFORUM.PL

TWITTER: @CYBERSECPL

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path–Relative Style Sheet Import (RPO/PRSSI)

Relative Path Overwrite (lub – jak w tytule – Path–Relative Style Sheet Import), jest nowym rodzajem ataku, w którym wykorzystuje się nietypowe zachowania serwerów aplikacyjnych oraz aplikacji webowych w zakresie mechanizmów przetwarzających względne adresy URL. Jak dużym zagrożeniem jest RPO/PRSSI i w jaki sposób podejść do pisania exploitów dla tej podatności? Odpowiedzi na te pytania znajdziecie w poniższej publikacji.

W 2014 roku znany badacz bezpieczeństwa przeglądarki – [Gareth Hayes](#) – opisał ciekawe zachowanie pewnej aplikacji webowej. Manipulując adresem URL, spowodował, że przeglądarkę do załadowania arkusza stylów o innej treści, niż spodziewałby się tego programista serwisu. Ostatecznie doprowadziło to do wstrzygnięcia złośliwego kodu CSS, a podatność zyskała nazwę „Path–Relative Style Sheet Import”. Uogólniając: Relative Path Overwrite jest zagrożeniem, w którym korzystając z manipulacji adresu względnego strony, staramy się zmienić jej zachowanie.

SŁOWEM WSTĘPU

Praktyczne wykorzystanie RPO/PRSSI wymaga połączenia kilku technik atakowania aplikacji webowych, przeglądarek oraz serwerów. W celach demonstracyjnych przeprowadzę analizę bezpieczeństwa fikcyjnej platformy sprzedaży znajdującej się pod adresem:

» <http://vulnerable.com/vuln.php>.

Serwis testowy działa w bardzo prostym modelu – zarejestrowany sprzedawca może dodać nowy przedmiot (grę) o wskazanej nazwie do serwisu. Na stronie głównej klienci widzą listę wszystkich dostępnych gier oraz mogą zdecydować się na ich kupno – tak jak na popularnych portalach aukcyjnych.

Wygląd serwisu demonstruje rysunek 1, zaś jego kod został przedstawiony na listingach 1 oraz 2.



Rysunek 1. Centrum sprzedaży gier, w którym sprzedawcy mogą wystawiać produkty na sprzedaż.

Listing 1. Kod HTML testowanego serwisu (przed atakiem).

```
<!doctype html>
<html>
<head>
  <link rel="stylesheet" href="style.css" />
</head>
<body>
  <h1>Twoje Zaufane Centrum Sprzedaży</h1>
  <h3>Lista produktów wystawiona przez sprzedających:</h3>
  <ul>
    <li>Fallout 4 (PS4 Blu-ray)</li>
    <li>Darkness Dungeon (Steam key)</li>
    <li>This War of Mine (Steam key)</li>
  </ul>
</body>
</html>
```

Listing 2. Zawartość pliku style.css.

```
h1 {
  color: #00CC00;
  font-style: italic;
}
```

GDZIE SZUKAĆ PODATNOŚCI

Podatności RPO szukają się na stronach, na których przeglądarka wysyła żądania po dodatkowe zasoby – w szczególności arkusze stylów CSS – które zostały zadeklarowane w postaci adresów względnych.

W omawianym przykładzie, niebezpieczne odwołanie znajduje się w linii 4.

```
<link rel="stylesheet" href="style.css" />
```

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path–Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



Gdy treść strony ładowana jest spod adresu <http://vulnerable.com/vuln.php>, arkusz stylów zostanie załadowany z adresu <http://vulnerable.com/style.css>.

Powyższe odwołanie do pliku CSS rozpoczyna zabawę w wykorzystanie podatności RPO. Aby jednak zrozumieć ideę ataku, najpierw musimy wiedzieć, w jaki sposób serwery WWW obsługują dynamiczne treści – takie jak skrypty PHP, JSP czy ASP – w kontekście adresów URL.

Poniższe opracowanie wykorzysta w przykładach sposób testowania wskazany wyżej, tj. dopisywanie na końcu adresu URL ciągów przypominających strukturę katalogów (*/testing/example*). W zależności od konfiguracji serwera, frameworka czy nawet samej aplikacji webowej istnieją także inne metody manipulacji adresem URL wywołującym opisywane zachowanie – odsyłam do lektury zasobów wskazanych w materiałach dodatkowych na końcu artykułu.

JAK SERWER OBSŁUGUJE ADRES URL

Jednym z etapów przetwarzania żądania przez serwer WWW (serwer aplikacyjny) jest podjęcie decyzji, który moduł (uchwyt, biblioteka, rozszerzenie) powinien finalnie obsługiwać żądanie. Każdy serwer robi to w odmienny sposób, ale w większości przypadków decyzja podejmowana jest na podstawie rozszerzenia zasobu, o który prosi klient. I tak: żądanie o zasób */index.php* zostanie przekazane do interpretera PHP, natomiast */index.aspx* trafi na pipeline ASP. Pliki typu *style.css*, *script.js* lub *image.png* są z reguły obsługiwane przez uchwyt plików statycznych, którego zadaniem jest odczytanie pliku z dysku i odesłanie jego treści do użytkownika.

Zgodnie z powyższym, odwiedzenie strony <http://vulnerable.com/vuln.php> zostanie odebrane przez uchwyt PHP, który przeczyta plik *vuln.php* na serwerze, prześle go do interpretera języka PHP, a odpowiedź odesle do przeglądarki internauty.

Co ciekawe, okazuje się, że odwiedzenie strony <http://vulnerable.com/vuln.php> **vizzdoom** oraz <http://vulnerable.com/vuln.php> zostanie obsłużone w dokładnie taki sam sposób – w obu przypadkach do interpretera PHP trafi zawartość pliku *vuln.php*. Jest to nieintuicyjne – można by się spodziewać, że pierwsze żądanie zostanie obsłużone przez uchwyt plików statycznych, który odszuka plik *vizzdoom* w katalogu *vuln.php* serwera. Tak się jednak nie stanie. I właśnie takie zachowanie w adresach URL jest wykorzystywane w podatnościach RPO/PRSSI.

Uchwyty do obsługi dynamicznej treści WWW mogą ignorować niektóre elementy adresu URL. Jest to łatwe do przetestowania na własnych serwerach – wystarczy odwiedzić adres kończący się rozszerzeniem *.php*, *.asp*, *.aspx*, *.jsp*, a następnie dopisać po nim ciąg */testing/example* i przeanalizować odpowiedź.

JAK WYGLĄDA PODATNOŚĆ RPO/PRSSI

Jeśli znamy sposób przetwarzania żądań HTTP przez serwery, nie powinno być dla nas niespodzianką, że odwołanie do adresów:

1. <http://vulnerable.com/vuln.php>,
2. <http://vulnerable.com/vuln.php/put/evil/here>

zwróci dokładnie ten sam kod HTML (listing 1) zawierający odwołanie do zewnętrznego arkusza stylów:

```
<link rel="stylesheet" href="style.css" />
```

W pierwszym przypadku przeglądarka wyśle do serwera vulnerable.com żądanie HTTP GET */style.css*, otrzymując w odpowiedzi selektory CSS z listingu 2, co można zobaczyć podczas analizy ruchu sieciowego (rysunek 2). W testowanej aplikacji efektem będzie wyświetlenie strony sklepu z zielonym nagłówkiem (rysunek 1).

Network								
SUMMARY		DETAILS						
URL	Protocol	Method	Result	Type	Received	Taken	Initiator	Timings
http://vulnerable.com/vuln.php	HTTP	GET	200	text/html	1.17 KB	31 ms	refresh	
/style.css	HTTP	GET	200	text/css	502 B	< 1 ms	<link rel="stylesheet">	

Rysunek 2. Ruch sieciowy podczas odwiedzenia strony <http://vulnerable.com/vuln.php>.

Ciekawsze jest jednak zachowanie przeglądarki dla drugiego adresu (<http://vulnerable.com/vuln.php/put/evil/here>), gdyż tutaj żądanie HTTP będzie skierowane do zasobu */vuln.php/put/evil/style.css*. Zamiast arkusza CSS serwer w odpowiedzi zwróci kod HTML z listingu 1, co spowoduje błędne wyświetlenie nagłówka (prawidłowy powinien być zielony, natomiast błędny jest wyświetlany inaczej) – porównaj rysunek 1 oraz 3).

Jak widać, manipulując adresem URL wpłynieliśmy na załadowanie innego zasobu, niż przewidział to programista strony (rysunek 4). W obecnej sytuacji przeglądarka wczytuje stronę *vuln.php* z arkuszem stylów, którego zawartością jest... kod strony *vuln.php*.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



Wstrzyknięcie źródła HTML do arkusza stylów to sztandarowy przykład ataku *Path-Relative Style Sheet Injection*, który jest spowodowany manipulacją odwołań relatywnych (*Relative Path Overwrite*).

Listing 3. Adres URL rozpoczynający atak RPO/PRSSI w domenie vulnerable.com.

<http://vulnerable.com/vuln.php/put/evil/here>

URL	Protocol	Method	Result	Type	Received	Token	Initiator
http://vulnerable.com/vuln.php/put/evil/here	HTTP	GET	200	text/html	0.88 KB	16 ms	navigate
http://vulnerable.com/vuln.php/put/evil/style.css	HTTP	GET	200	text/html	0.88 KB	31 ms	<link rel="style..."

Rysunek 3. Wygląd strony i ruch sieciowy podczas odwiedzenia strony <http://vulnerable.com/vuln.php/put/evil/here>.

```

<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="style.css" />
</head>
<body>
<h1>Twoje Zaufane Centrum Sprzedaży</h1>
<h3>Lista produktów wystawiona przez sprzedających:</h3>
<ul>
<li>Fallout 4 (PS4 Blu-ray)</li>
<li>Darkness Dungeon (Steam key)</li>
<li>This War of Mine (Steam key)</li>
</ul>
</body>
</html>

```

Rysunek 4. Zamiast definicji stylów (lub odpowiedzi 404 Not Found), serwer odpowiada jak na żądanie HTTP GET /vuln.php.

W atakach RPO nie musimy zawsze dodawać tej samej strony (tutaj: *vuln.php*) – zapraszam do lektury na temat innych wariantów udostępnionych w sekcji „*Źródła i materiały dodatkowe*”.

Warto również dodać, że błędy RPO/PRSSI od pewnego czasu są raportowane przez narzędzie **Burp Suite Pro** (w trybie active scan, ale czasem również w passive), co można zobaczyć na rysunku 5.

Rysunek 5. Moduł skanera Burp Suite PRO wykrywa zagrożenie manipulacji ścieżki w celu załadowania złośliwych reguł CSS i raportuje je jako "Path-relative style sheet import".

DOSTARCZENIE ŁADUNKU ZE ZŁOŚLIWYM KODEM

Załadowanie arkusza CSS o zawartości będącej kodem HTML nie powoduje jeszcze żadnych strat. Praktyczna eksplatacja tej „podatności” to zazwyczaj bardzo skomplikowany, wieloetapowy proces.

Najpierw zastanówmy się, w jaki sposób możemy dostarczyć sam ładunek ze złośliwym kodem. Ponieważ na wejściu interpretera CSS pojawi się źródło atakowanej strony, dostarczenie payloadu jest dość proste – wystarczy w jakikolwiek sposób wpłynąć na zawartość atakowanej strony.

Można to zrobić następująco:

- manipulując parametrami URL, które wypisywane są na stronie – do czego świetnie nadają się wszelkiego rodzaju wyszukiwarki działające z parametrem ?search;
- przez dodanie nowego elementu do bazy serwisu, który później będzie wyświetlony (tzw. „Second order injection”) – jeśli nie widzimy bezpośredniej refleksji parametru URL.

Na potrzeby dalszej analizy użyję drugiej metody.

W tym celu, jako sprzedawca serwisu dodam nowy produkt, który zostanie wyświetlony na stronie głównej. Strona dość restrykcyjnie koduje kluczowe znaki HTML, zabezpieczając się przed klasycznymi atakami XSS, jednak bez problemów udaje mi się dodać przedmiot o nazwie: * {color: red; font-size: x-large}.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



Warto zastanowić się, dlaczego RPO omawia się głównie w kontekście wstrzygnięć CSS – przecież relatywna ścieżka może dotyczyć skryptów JS (<script src="script.js">). Otóż po wstrzygnięciu na wejście interpretera Javascript trafi kod rozpoczynający się od <!doctype html> lub <html>. Spowoduje to zgłoszenie błędu składni oraz przerwanie dalszego przetwarzania „skryptu”.

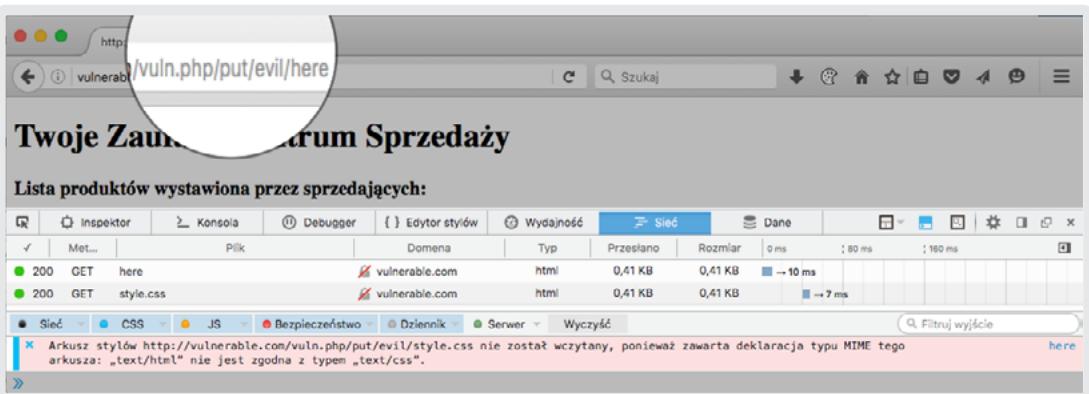
Interpreter CSS jest za to bardziej tolerancyjny — w momencie napotkania błędu, może podjąć próbę dalszej interpretacji pliku (w szczególności gdy strona wyświetlana jest w tzw. „Quirks Mode”).

OMINIĘCIE WALIDACJI TYPU MIME

Na liście przedmiotów sklepu wyświetlany jest teraz szkodliwy ładunek. Odwiedzając adres z listingu 3, spowodujemy sytuację, w której przeglądarka wczyta arkusz stylów ze znacznikami HTML oraz selektorem * {color: red; font-size: x-large}. Powinniśmy spodziewać się strony, której tekst zostanie wyświetlony na czerwono.

Niestety, tak się nie stanie – współczesne przeglądarki zauważają, że arkusz CSS posiada podejrzany typ MIME (*Content-Type* odpowiedzi *text/html*). Dlatego też w konsoli Javascript przeglądarki pojawi się komunikat błędu informujący o zaprzestaniu przetwarzania podejrzanej pliku (rysunek 6).

Powyższy mechanizm zatrzyma próbę ataku we wszystkich współczesnych przeglądarkach internetowych – Firefox, Chrome, Safari, Internet Explorer – w wersjach 9–11 oraz MsEdge. Czyżby ofiarą ataku mogli być tylko użytkownicy Internet Explorer 8 (który używany jest wyłącznie w przestarzałym systemie Windows XP)? Niekoniecznie.



Rysunek 6. Przeglądarki mogą blokować ładowanie zasobów z podejrzany typem MIME (powyżej: Firefox).

Przeglądarki z rodziny Internet Explorer potrafią korzystać z trybu wstecznej kompatybilności (**Document Mode**). W takiej sytuacji, np. IE 11, wyświetli stronę, korzystając z mechanizmów IE 8. Zachowanie takie można włączyć przez dodanie znacznika HTML <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE8"> do sekcji <head>. Starsze wersje Internet Explorera nie walidują typu MIME plików CSS, więc strona wyświetliona w Document Mode IE 8 spowoduje ominięcie restrykcji dla złośliwego ładunku CSS.

W jaki sposób spowodować, aby strona <http://vulnerable.com/vuln.php> była wyświetlana w trybie wstecznej kompatybilności Internet Explorera? Przecież nie możemy tak po prostu zmodyfikować jej kodu i dodać tag <meta>...

Aby kontynuować atak, trzeba znać ciekawą właściwość trybu wstecznej kompatybilności IE – otóż ten tryb może być dziedziczony z dokumentu rodzica, bez restrykcji nakładanych przez reguły *Same Origin Policy*. Jeśli testowana strona (*vuln.php*) nie posiada definicji X-UA-Compatible, wartość taka może zostać pobrana z nadzawanego kontekstu przeglądarki – czyli przykładowo – ze strony (atakującego), która zaramkuje (znacznikiem *iframe*) testowaną stronę.

Dalsza demonstracja ataku, omijająca restrykcje typu MIME, kierowana będzie na użytkowników Internet Explorer 11 – podczas pisania artykułu jest to przeglądarka oficjalnie wspierana i aktualizowana w systemach Windows 7/8/8.1.

Jako agresor tworzę dokument HTML na fikcyjnej domenie *attacker.com*, którego kod widnieje na listingu 3. Strona atakującego:

- » tworzy ramkę do <http://vulnerable.com/vuln.php/put/evil/here>,
- » w sekcji <head> włącza tryb Document Mode IE8 (X-UA-Compatible).

Po odwiedzeniu strony agresora w przeglądarce IE ramka będzie renderowana przy pomocy mechanizmów IE 8, gdyż ramka odziedziczy Document Mode z ramkującej strony.

Listing 4. Kod strony atakującego, który wymusza użycie trybu wstecznej kompatybilności IE dla atakowanej strony.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE8">
</head>
<body>
<iframe height="500" width="500" src="http://vulnerable.com/vuln.php/put/evil/here"></iframe>
</body>
</html>
```

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



W ten sposób omijamy restrykcje przeglądarek, które odrzucają próbę załadowania arkusza CSS z nagłówkiem Content-Type: text/html.

Chociaż w ten sposób uda nam się ominąć restrykcje IE 11 dla typu MIME, atak dalej się nie powiedzie. Zawartość ramki nadal nie zawiera strony z czerwonymi napisami, co oznacza, że ładunek `* {color: red; font-size: x-large}` mimo poprawnego załadowania – nie zadziałał.

OSZUKANIE PARSERA CSS

W sekcji „Dostarczenie ładunku ze złośliwym kodem” wspomniałem, że parser CSS nie jest tak restrykcyjny, jak interpreter Javascript. Niestety, nie jest on na tyle liberalny, aby poprawnie zinterpretować linię:

```
<li>* {color: red; font-size: x-large}</li>
```

Znaczniki `` pojawiają się w kodzie strony, gdyż każdy dodawany w serwisie przedmiot jest elementem nieuporządkowanej listy HTML. Refleksja tego przedmiotu jest ujęta w znaczniki ``, które nie są poprawnym selektorem CSS. Parser zauważa ten błąd i przejdzie do analizy reszty (wstrzyknietego) pliku.

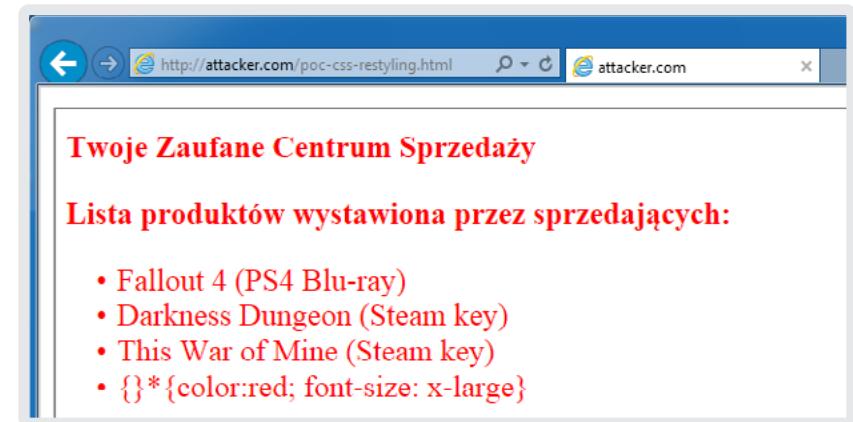
Aby wstrzyknięcie złośliwych właściwości CSS doszło do skutku, należy lekko zmodyfikować nasz ładunek. W tym celu dodajemy do serwisu przedmiot, który zaczyna się od znaków `{`, po których dodajemy złośliwy selektor (*, body, p lub dowolny inny) wraz z definicjami. Wówczas otrzymamy:

```
<li>{* {color: red; font-size: x-large}</li>
```

Pierwsza (czerwona) część zostanie zinterpretowana jako niepoprawny selektor o pustej liście właściwości. Parser odrzuci tę część i spróbuje znaleźć kolejny, poprawny fragment. Znak gwiazdki jest poprawnym selektorem CSS, dlatego zielona część zostanie wzięta pod uwagę przez przeglądarkę. Pozostała część pliku jest nieistotna, ponieważ udało się nam już wstrzyknąć to, co chcieliśmy.

W tym momencie, ofiara odwiedzająca domenę attacker.com, zobaczy ramkę z zawartością `vuln.php` z czerwonym, powiększonym tekstem – co potwierdza udane wstrzyknięcie CSS (rysunek 7).

W końcu sukces!

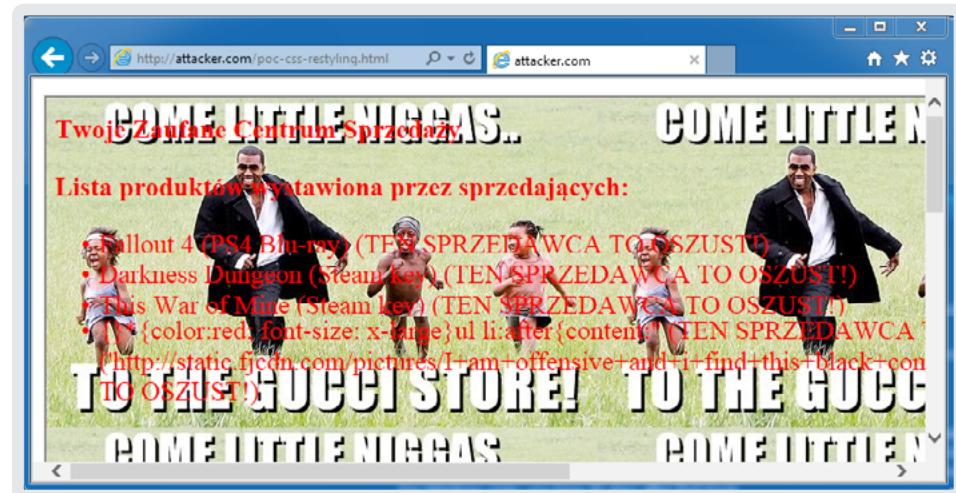


Rysunek 7. Udane wstrzyknięcie kodu CSS na testowanej stronie.

Aby urealnić atak, możemy pokusić się o załadowanie złośliwego obrazka oraz dopisanie własnego tekstu w określone miejsca strony. Ładunek wykonujący te operacje przedstawia listing 5, zaś jego efekt można zobaczyć na rysunku 8.

Listing 5. Wycinek pliku ze stylami, który zawiera złośliwy ładunek.

```
<li>{* {color:red; font-size: x-large}ul li:after{content:" (TEN SPRZEDAWCA TO OSZUST!)"}body{background-image:url('http://static.fjcdn.com/pictures/I+am+offensive+and+i+find+this+black+come_3d9dcc_3223497.jpg')}</li>
```



Rysunek 8. Demonstracja udanego wektora ataku. CSS Injection pokazany w IE11 dzięki ramkowaniu, Relative Path Override oraz trybie kompatybilności wstępnej przeglądarki.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



XSS PRZEZ CSS, CZYLI DŁUGA DROGA DO GROŹNEJ PODATNOŚCI

Atak na wizerunek korporacji może być smacznym kąskiem dla agresora, jednak w większości wypadków, tak skomplikowane działania powodujące zmianę wyglądu strony, nie będą się opłacały. Co innego, gdyby przez manipulację stylem można było wykonać dowolny kod Javascript. Czy XSS przez CSS Injection jest dzisiaj jeszcze możliwe?

W Internet Explorer 7 pojawiły się dynamiczne właściwości CSS, zwane również CSS Expressions (dynamic properties). W Internet Explorer 8–10 już z nich zrezygnowano, ale Expressions dalej działały w nich w trybie wstępnej kompatybilności. Od razu na myśl przychodzi zaramkowanie strony w starym Document Mode i wywołanie XSS przez wstrzyknięcie CSS Expression. Microsoft jednak całkowicie usunął dynamiczne właściwości CSS w IE 11 i taki atak nie jest już możliwy.

Internet Explorer (w odróżnieniu od MsEdge) nadal wspiera inne, „egzotyczne” mechanizmy, które mogą pomóc nam w ataku. Jednym z takich mechanizmów jest **DHTML Behaviors** – wprowadzony w IE 5.5, działający aż do IE 10 (oraz w Document Mode w najnowszej wersji Internet Explorer). W tym mechanizmie właściwość CSS może odwoływać się do pliku **skryptletu** (.sct scriptlet), który może wykonać kod Javascript.

Aby skryptlet zadziałał, musi znajdować się na tej samej domenie, co ładujący go arkusz stylów. W dodatku plik skryptletu musi zwracać w odpowiedzi HTTP typ MIME **text/x-scriptlet**. Są to spore utrudnienia dla atakującego, ale postaramy się je obejść.

Zacznijmy od tego, co już potrafimy – ramkujemy stronę <http://vulnerable.com/put/evil/here> i wstrzykujemy styl CSS przez dodanie produktu o następującej nazwie:

```
{}*{color:red;}body{behavior: url(../ODWOLANIE-DO-SCRIPTLETU")}
```

Działanie właściwości **behavior** będzie zaimplementowane w skryptlecie. Ramka w domenie atakującego jest tu potrzebna, aby strona obsłużyła wstrzyknięcie CSS oraz włączyła archaiczny mechanizm skryptletów.

Następnie tworzymy sam skryptlet. Dla demonstracji jego implementacja będzie pokazywać informację w kontekście jakiej domeny wykonano atak XSS (listing 6).

Listing 6. Skryptlet będący elementem ataku XSS.

```
<scriptlet>
<implements type="behavior"/>
<script>alert("scriptlet XSS on " + document.domain)</script>
</scriptlet>
```

Teraz musimy zadbać o załadowanie tego pliku na serwer. Mogą w tym pomóc funkcje serwisu związane z dodawaniem nowych obrazków lub awatarów. Dla uproszczenia założymy, że możemy wgrać dowolny plik o rozszerzeniu **.png** na domenę vulnerable.com. Zapisujemy więc scriptlet jako plik **xss.png** i ładujemy go na stronę.

Ostatnim krokiem będzie ominięcie wymagania odnośnie typu MIME skryptletu. Serwer, zwracając plik <http://vulnerable.com/xss.png>, nie odpowie nagłówkiem **Content-Type: text/x-scriptlet**, co uniemożliwi uruchomienie właściwości **behavior** w CSS. Aby obejść to zabezpieczenie, atakujący musi skorzystać z kolejnej słabości przeglądarki – tym razem będzie to Content Sniffing.

Content Sniffing to mechanizm, który pozwala przeglądarce określić, w jaki sposób ma przetworzyć plik, niezależnie od jego wartości Content-Type. Coś, co jest obrazkiem (**image/png**), może więc okazać się stroną HTML, skryptem lub innym zasobem.

Internet Explorer aktywuje Content Sniffing m.in. podczas przetwarzania odpowiedzi HTTP z nagłówkiem Content-Type o wartości:

- » **text/html**,
- » **text/plain**,
- » **image/***,
- » **video/mpeg**,
- » **video/avi**.

Zgodnie z powyższym, gdy załadujemy obrazek <http://vulnerable.com/xss.png>, przeglądarka otrzyma odpowiedzi z nagłówkiem Content-Type: **image/png**, więc włączy mechanizm Content Sniffing. Analizując odebrane dane, Internet Explorer wykryje, że faktyczną zawartością ładowanego zasobu jest plik o typie MIME **text/x-scriptlet**.

Mamy już więc wszystkie elementy układanki. Skutecznie wykonany atak XSS w kontekście vulnerable.com będziemy mogli zobaczyć po wykonaniu następujących kroków:

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning

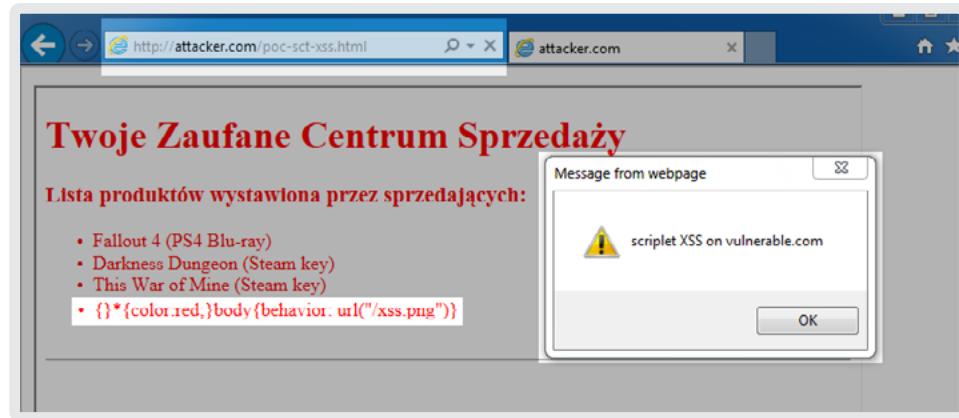


1. dodanie produktu o nazwie z listingu 7,
2. załadowanie skryptletu `xss.png`,
3. odwiedzenie strony w domenie `attacker.com`.

Końcowy efekt demonstruje rysunek 9.

Listing 7. Działający ładunek, uaktywniający skryptlet.

```
{}*{color:red;}body{behavior: url(../xss.png")}
```



Rysunek 9. XSS przez CSS Injection w Internet Explorer 11 – upragnione wykonanie złośliwego kodu Javascript.

PODSUMOWANIE I WNIOSKI Z ATAKU

Cały wektor ataku, skutkujący wykonaniem złośliwego kodu Javascript w domenie `vulnerable.com`, wygląda następująco:

1. Zauważamy, że `vulnerable.com` jest podatne na atak RPO.
2. Wykorzystujemy RPO (<http://vulnerable.com/put/evil/here>) w celu załadowania pliku `style.css` o zawartości strony `vuln.php`.
3. Wpływamy na treść strony, dodając produkt o nazwie z listingu 7. Ładunek posiada właściwość CSS `behavior`, której implementacja znajduje się w pliku `xss.png`.
4. Dodajemy obrazek na atakowanej stronie, którego zawartością jest skryptlet z listingu 6.
5. Na domenie `attacker.com` tworzymy stronę `poc-sct-xss.html`.
6. Dodajemy do niej ramkę ze stroną z punktu 2.
7. Do ramkującej strony dodajemy tag `X-UA-Compatible`, który spowoduje wyświetlenie strony `poc-sct-xss.html` w trybie IE 8 – pozwoli to m.in. na przetwarzanie

- nie skryptletów oraz ominięcie walidacji typów MIME.
8. Nakłaniamy ofiarę, aby odwiedziła domenę `attacker.com` przy użyciu dowolnej wersji Internet Explorer. Atak XSS wykona się w kontekście `vulnerable.com`.

Warto nadmienić, że w trakcie pisania artykułu (początek 2016 roku), udział wersji Internet Explorera na rynku przeglądarek, na których zadziała powyższy wektor ataku, wahala się w przedziale 5–40% (w zależności od źródeł).

JAK SIĘ CHRONIĆ

Sugerowaną przeze mnie metodą ochrony przeciwko atakom RPO jest dodanie znacznika HTML `<base href="http://example.com/" />`. Jego celem jest definiowanie ścieżki bazowej dla wszystkich elementów „relatywnych” na stronie, co zatrzymuje pierwszy etap ataku. Aby nie pozwolić na nadpisanie wartości elementu `base` podczas wstrzyknięć, strona powinna dodatkowo nakładać obostrzenia Content Security Policy dyrektywą `base-uri` (więcej o CSP w artykule na str. 8-19).

Uwaga: Znacznik `<base>` musi być jednym z pierwszych elementów sekcji `head`, jeszcze przed odwołaniami do stylów oraz skryptów.

Remedium na skomplikowane wektory ataków jest dobra jakość kodu oraz wdrażanie metod dogłębnej ochrony (ang. defense in depth). Dlatego, aby uchronić się przed tego rodzaju zagrożeniami, należy w aplikacji webowej:

- » deklarować `<!DOCTYPE html>` w pierwszej linii odpowiedzi strony HTML,
- » nie pozwalać na „ramkowanie” strony (używaj nagłówków X-Frame-Options oraz Content-Security-Policy z dyrektywą `frame-ancestors`),
- » wyłączyć mechanizm Content Sniffing (nagłówek X-Content-Type-Options: nosniff),
- » uniemożliwić dziedziczenie trybu wstępnej kompatybilności w IE – dodaj znacznik `<meta http-equiv="X-UA-Compatible" content="IE=Edge" />` (Edge oznacza tu najnowszą wersję Internet Explorer, a nie tryb „Microsoft Edge”),
- » zwracać poprawny typ MIME oraz informacje o kodowaniu (np. UTF8) w nagłówkach Content-Type.

W poprawieniu jakości kodu mogą pomóc [automaty do statycznej analizy](#).

Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



PODSUMOWANIE

Relative Path Override to łatwe w wykryciu zagrożenie, jednak wyrządzenie realnych szkód przy pomocy tej podatności wymaga sporo wysiłku. Obecne przeglądarki internetowe są coraz bezpieczniejsze i stopniowo utrudniają wykorzystanie takiej luki. Nie zapominajmy, że doświadczony atakujący przy pomocy RPO/PRSSI może próbować:

- » wpływać na wygląd strony przez CSS (co może skutkować utratą wizerunku, czasem też niedostępnością serwisu),
- » wykradać tokeny anty-CSRF (przez co atakujący będzie mógł nakłonić zalogowanego użytkownika do wykonania określonych akcji na stronie),
- » wykradać zawartość URL, np. identyfikatory sesji (co grozi przejęciem konta zalogowanego użytkownika),
- » czy nawet wykonać XSS (kradzież identyfikatora sesji, kradzież haseł przez keyloggerów webowych i wiele innych...).

Jak widać, błędy RPO mogą stymulować poważne podatności – taka sytuacja dotyczyła chociażby forum [phpBB 3](#) lub frameworku [Cake PHP](#). Dlatego też nie powinniśmy lekceważyć tego zagrożenia i uwzględniać RPO/PRSSI jako element testów bezpieczeństwa aplikacji webowej.

Źródła i materiały dodatkowe

- <http://blog.portswigger.net/2015/02/prssi.html>
- <http://blog.innerht.ml/cascading-style-scripting/>
- <http://www.mbsd.jp/Whitepaper/rpo.pdf>
- <http://garethhayes.net/>

Adrian 'Vizzdoom' Michalczyk. Interdyscyplinarny bezpiecznik i geek.
Zakochany w dobrej fabule, roleplay'u i fotografii...
Strona domowa autora: <http://adrian.michalczyk.website/>



Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



Mechanizm HTTP Public Key Pinning

Bezpieczne połączenia przy użyciu protokołu HTTPS stanowią jeden z podstawowych budulców dzisiejszego Internetu. HTTPS zapewnia poufność, integralność oraz autentyczność komunikacji. Ten ostatni cel jest realizowany dzięki standardowi X.509, który definiuje infrastrukturę klucza publicznego. Model zaufania oparty na X.509 ma jednak pewną fundamentalną wadę, której rozwiążaniem ma być mechanizm HTTP Public Key Pinning (HPKP).

JAK DZIAŁA X.509

Jak już wspomniano wyżej, X.509 definiuje infrastrukturę klucza publicznego, umożliwiając tym samym potwierdzenie, czy host, z którym próbujemy się połączyć, jest tym, za który się podaje. Potwierdzenie jego autentyczności odbywa się przy użyciu szeregu certyfikatów. Na samej górze mamy certyfikat główny (*root certificate*). Ten certyfikat należy do głównego urzędu certyfikującego (*root CA*) i jest samopodpisany (*self-signed*). Dzisiejsze systemy operacyjne oraz przeglądarki internetowe zawierają kilkudziesiąt zaufanych certyfikatów głównych.

Główne urzędy certyfikujące mogą uprawniać inne podmioty do działania w ich imieniu przez wygenerowanie im certyfikatów pośrednich (*intermediate certificate*), podpisując je kluczem prywatnym swojego certyfikatu głównego.

Zazwyczaj ścieżka certyfikacji dla stron internetowych składa się z trzech certyfikatów: certyfikatu głównego, jednego certyfikatu pośredniego oraz certyfikatu domeny. Sprawdzenie poprawności łańcucha certyfikatów odbywa się poprzez validację podpisów cyfrowych każdego z nich. Jeśli walidacja się powiedzie, przeglądarka bądź system operacyjny uzna ją, że połączenie jest zaufane.

Gdzie tkwi problem w tym modelu?

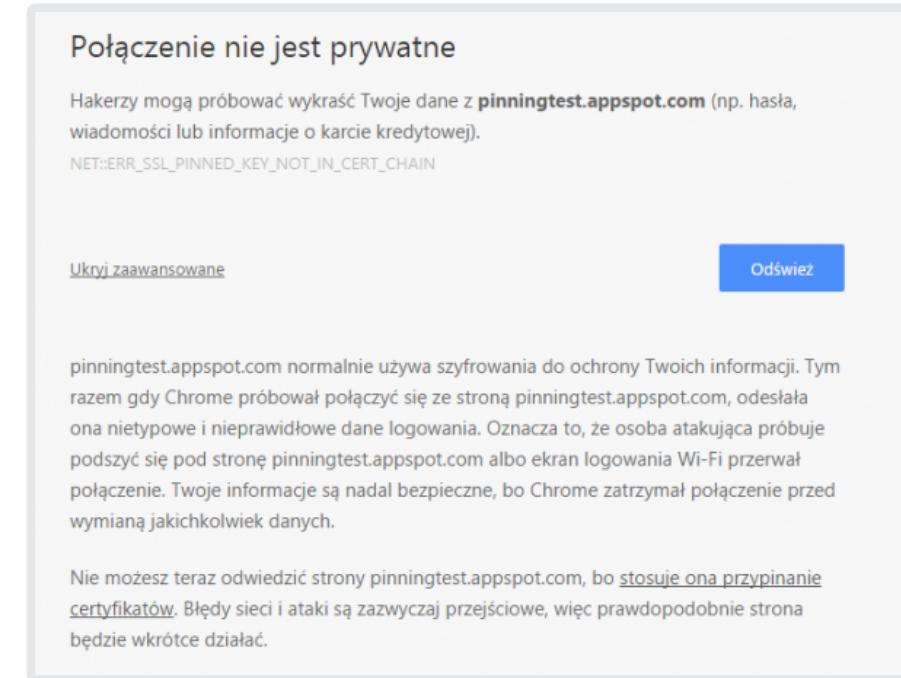
Główne oraz pośrednie centra certyfikacji nie mają wyszczególnionych konkretnych domen, do których mogą wystawiać certyfikaty. Oznacza to, że każdy z tych podmiotów może wystawić zaufany certyfikat dla dowolnej domeny. Jak już wspomniano wyżej, głównych urzędów certyfikacji jest kilkudziesiąt, zaś, jak podała organizacja EFF (*Electronic Frontier Foundation*) w [prezentacji z 2010 roku](#), w sumie wtedy wszystkich zaufanych podmiotów było 1482.

Głównym urzędem certyfikacji trudno utrzymać kontrolę nad wszystkimi z nich i czasem zdarzają się nadużycia. Ostatni głośny przypadek tego typu pochodzi

z marca 2015, gdy egipska firma MCS Holdings wystawiła nieautoryzowane certyfikaty dla kilku domen Google. Dzięki temu pracownicy firmy mogli przeprowadzić atak typu MiTM (*man in the middle*) na część użytkowników usług Google, którzy nie byliby nawet tego świadomi, bowiem przeglądarka internetowa nie wyświetliłaby żadnego ostrzeżenia o błędnych certyfikacjach.

PRZYPINANIE CERTYFIKATÓW

Aby rozwiązać opisany powyżej problem, w przeglądarkach Chrome oraz Firefox zaczęto stosować przypinanie certyfikatów (*certificate pinning*). Ten mechanizm pozwala określić, które urzędy certyfikacji są uprawnione do wystawienia certyfikatu dla danej domeny. Na przykład Google może zdefiniować, że wszystkie certyfikaty dla ich domen muszą zostać podpisane przez urząd certyfikacji Google Internet Authority G2. Gdy ten warunek nie zostanie spełniony, przeglądarka wyświetli ostrzeżenie (rysunek 1.). Dzięki temu, opisany wcześniej przypadek z wystawieniem fałszywego certyfikatu dla domen Google nie powiodłby się.



Rysunek 1. Ostrzeżenie przeglądarki Chrome w razie użycia nieprzypiętego certyfikatu – informacja o nieprzypiętym certyfikacie znajduje się na dole komunikatu.

Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning



Mechanizm przypinania certyfikatów nie jest nowym „wynalazkiem”; w Chrome jest stosowany już od 2011 roku. Dotychczas jednak polegał na tym, że w przeglądarce zdefiniowano listę konkretnych domen oraz przypiętych certyfikatów.

NAGŁÓWEK HTTP PUBLIC-KEY-PINS (HPKP)

W kwietniu 2015 roku została opublikowana finalna wersja [RFC 7469](#), w którym zdefiniowano nagłówek odpowiedzi Public-Key-Pins, dzięki któremu właściciel do wolnej witryny internetowej może skorzystać z dobrodziejstw przypinania certyfikatów. Oto przykładowa wartość tego nagłówka:

```
Public-Key-Pins: pin-sha256="d6qzRu9z0ECb90Uez27xWltNsj0e1Md7GkYYkVoZWmM="; pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g="; pin-sha256="LPJNul+wow4m6DsxbninhSwH1wfp0JecwQzYp0LmcQ="; max-age=10000; includeSubDomains; report-uri="http://example.com/hpkp-report"
```

Poniżej opis poszczególnych dyrektyw:

- » **pin-sha256** – pole wymagane, odcisk palca jednego z certyfikatów z łańcucha (więcej o generowaniu wartości dla tego pola w dalszej części artykułu); RFC przewiduje, że w przyszłości mogą być używane inne algorytmy niż sha256, jednak na razie jest jedynym obsługiwany; wartość hasha musi z kolei być enkodowana w base64,
- » **max-age** – pole wymagane, czas ważności danego przypięcia w sekundach; innymi słowy: po upływie danego czasu przeglądarka przestanie traktować daną domenę jako używającą piningu certyfikatów,
- » **includeSubDomains** – pole opcjonalne. Domyślnie przypinanie certyfikatów działa tylko dla domeny, na której był obecny nagłówek Public-Key-Pins; dzięki użyciu tej dyrektywy przypinanie działa także dla wszystkich poddomen,
- » **report-uri** – pole opcjonalne, pozwala zdefiniować adres, pod który przeglądarka zgłosi naruszenie certyfikatu.

Nagłówek Public-Key-Pins wygląda dość podobnie do HSTS (HTTP Strict Transport Security). Podobnie jak w przypadku HSTS, możliwe jest zdefiniowane trybu Report-Only (trzeba wówczas użyć nazwy nagłówka: Public-Key-Pins-Report-Only), w którym przeglądarka będzie tylko informować o wykryciu naruszeń certyfikatów, nie będzie zaś blokować dostępu do witryny.

Najistotniejszym polem z wyżej wymienionych, jest pin-sha256, gdzie zdefiniowany jest odcisk palca jednego z certyfikatów z łańcucha. Jeżeli przeglądarka nie

znajdzie w łańcuchu certyfikatów żadnego certyfikatu pasującego do jakiegokolwiek odcisku, nie pozwoli otworzyć danej witryny. Nie ma znaczenia, z którego certyfikatu weźmiemy odcisk; wybór jest w gestii administratora. Może zatem to być zarówno certyfikat głównego CA, jak i certyfikat domeny. Trudno jednoznacznie powiedzieć, który wybór jest najlepszy; jeśli jednak spodziewamy się, że często będziemy zmieniać certyfikaty, najsensowniej będzie wskazać albo certyfikat główny, albo pośredni.

Co istotne, RFC 7469 definiuje, że w nagłówku muszą znaleźć się co najmniej dwie dyrektywy pin-sha256, z czego co najmniej jedna nie może odwoływać się do żadnego certyfikatu z łańcucha certyfikatów. Innymi słowy: administrator witryny musi zdefiniować zapasowy pin (*backup pin*), który musi odnosić się do innego, jeszcze nieużywanego certyfikatu. Wprowadzenie pinów zapasowych jest uzasadnione potrzebą zapewnienia ciągłości działania strony internetowej. Wyobraźmy sobie sytuację, w której używany jest tylko jeden pin, a następnie – na przykład w wyniku jakiegoś incydentu bezpieczeństwa – certyfikat zostaje unieważniony. Administrator witryny generuje nowy certyfikat u innego dostawcy, jednak użytkownicy, tak czy inaczej, nie będą mogli jej odwiedzić ze względu na niezgodność pinów. Dzięki zastosowaniu zapasowego pinu, problem zostaje rozwiązyany, bowiem w razie incydentu bezpieczeństwa administrator natychmiast wystawia stronę z zapasowym certyfikatem, zaś wcześniej w nagłówku Public-Key-Pins umieszczono już odcisk jego palca.

Aby wygenerować odcisk palca dla certyfikatu, można posłużyć się poniższym poleciением (zakładamy, że certyfikat znajduje się w pliku cert.pem):

```
openssl x509 -noout -in cert.pem -pubkey | openssl asn1parse -noout -inform pem -out /dev/stdout|openssl dgst -sha256 -binary /dev/stdin |openssl enc -base64
```

HPKP A PROXY

Testerzy aplikacji internetowych często korzystają z oprogramowania typu proxy (jak np. Burp czy Fiddler). Aby testować aplikacje działające w protokole HTTPS, najczęściej do lokalnego magazynu certyfikatów dodaje się certyfikat wygenerowany przez proxy. Teoretycznie mogłoby tutaj istnieć ryzyko, że testowanie witryn przestanie być możliwe, bowiem certyfikaty wystawione przez proxy z pewnością nie znajdą się na liście certyfikatów przypiętych do domeny. Okazuje się, że zarówno Firefox, jak i Chrome domyślnie pomijają sprawdzanie pinów, jeżeli certyfikat głównego CA znajduje się w lokalnym, prywatnym magazynie certyfikatów.

Mechanizm Service Workers

[Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej](#)

[Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity](#)

[Czym jest atak Padding Oracle](#)

[Czym jest Bit-Flipping](#)

[Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu](#)

[Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych](#)

[Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import \(RPO/PRSSI\)](#)

[Mechanizm HTTP Public Key Pinning](#)



W przeglądarce Firefox istnieje możliwość zmiany tego zachowania i wymuszenia bezwarunkowego sprawdzania pinów. Należy w tym celu ustawić w about: flags wartość flagi security.cert_pinning.enforcement_level na 2 (gdzie domyślną wartością jest 1). W Chrome takiej możliwości nie ma.

PODSUMOWANIE

Reasumując, mechanizm HTTP Public-Key-Pins (HPKP) pozwala minimalizować ryzyko bezpośrednio związane z modelem X.509, które polega na możliwości wygenerowania certyfikatu dla każdej domeny przez dowolne centrum certyfikacji. Dzięki nagłówkowi Public-Key-Pins administratorzy stron internetowych mogą sami określić, jakie certyfikaty są dopuszczalne.

Należy pamiętać, że HPKP wymaga posiadania pinu zapasowego. Najlepiej i najbezpieczniej jest, gdy pin odnosi się do zapasowego certyfikatu wygenerowanego przez inne centrum certyfikacji niż ten używany na „żywej” witrynie. Ze względu na konieczność posiadania dwóch różnych certyfikatów, można się spodziewać, że mechanizm HPKP będzie używany raczej tylko w dużych wdrożeniach, takich jak np. systemy bankowe.

Michał Bentkowski. Realizuje testy penetracyjne oraz audyty bezpieczeństwa w firmie [Securitum](#). Autor w serwisie sekurak.pl. Aktywnie (i z sukcesem) uczestniczy w znanych programach bug bounty.



Mechanizm Service Workers

Wszystko o CSP 2.0 – Content Security Policy jako uniwersalny strażnik bezpieczeństwa aplikacji webowej

Ochrona podatnych aplikacji webowych za pomocą wirtualnych poprawek w ModSecurity

Czym jest atak Padding Oracle

Czym jest Bit-Flipping

Bezpieczeństwo aplikacji webowych: podatności w mechanizmach uploadu

Ukryte katalogi i pliki jako źródło informacji o aplikacjach internetowych

Czym jest i jak wykorzystać podatność Relative Path Overwrite/Path-Relative Style Sheet Import (RPO/PRSSI)

Mechanizm HTTP Public Key Pinning

