ODOITWP, laboratorium nr 5

Julia Sadecka, Jakub Augustyn, Beniamin Jankowski

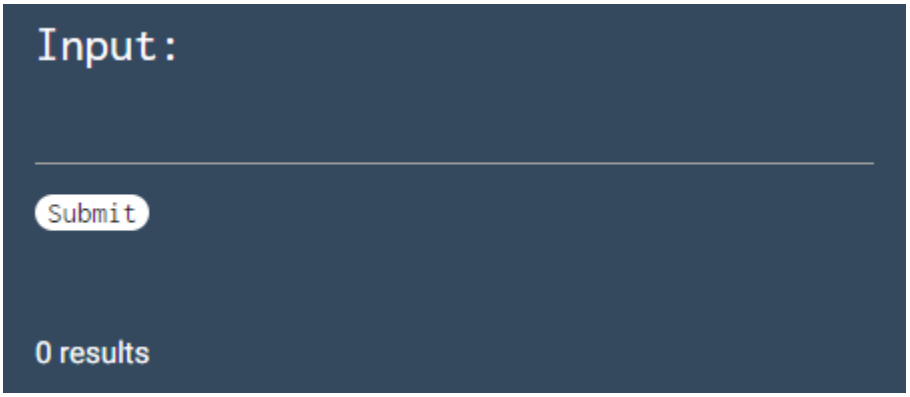
Nr 3. Injection

,,Wstrzykiwanie” nieprzerwanie figuruje na liście OWASP od początku jej powstania, a od 2007 roku jest na jednym z trzech topowych jej miejsc. Polega na wprowadzeniu złośliwego fragmentu kodu - najczęściej na stronie internetowej - w celu zmiany komendy wykonywanej na serwerze. Konsekwencją może być wyciek danych (ujawnienie całej tabeli zamiast jednego rekordu), niepożądana autoryzacja, czy nawet modyfikacja bądź usunięcie danych.
Do najpopularniejszych ataków typu Injection należą m. in. SQL injection, OS Command Injection, Cross-Site Scripting. Oczywiście to nie koniec, ponieważ tę samą podatność można wykorzystać też w plikach formatu PHP, XLS oraz niemal każdym który pobiera dane od użytkownika

SQL Injection

Język SQL jest stosowany do komunikacji z bazą danych, zatem manipulacja zapytaniami generowanymi przez niego może doprowadzić do niepożądanej autoryzacji oraz manipulacji danymi.

Przykładem może być poniższa strona.



Zapytanie wysyłane do serwera wygląda następująco:

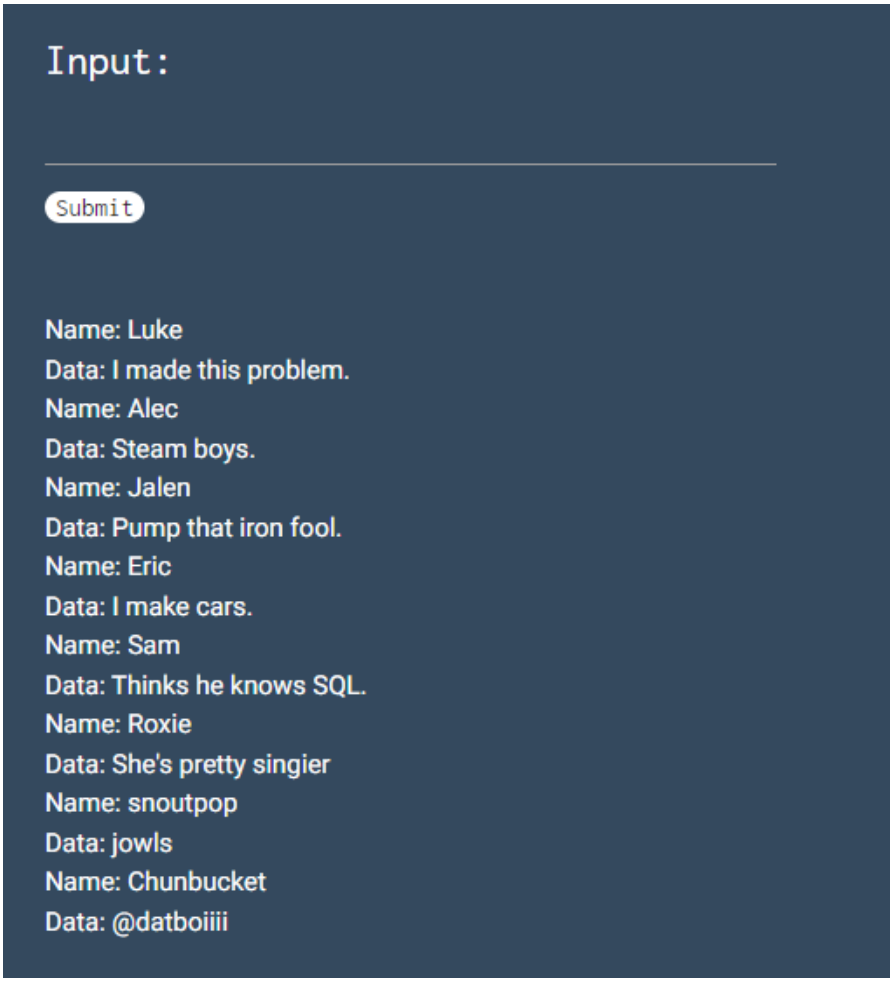
```
SELECT * FROM table where name = '$input'
```

Z założenia w polu *input* znajduje się imię. Przykładem SQL injection może być wstrzyknięcie kodu ” ‘ OR 1=1; – “
Dlaczego?
Spójrzmy jak teraz będzie wyglądać treść komendy:

```
SELECT * FROM table where name = '' OR 1=1; – '
```

Jak widać, komenda każe wypisać wszystkie dane dla użytkownika kiedy spełniony jest co najmniej jeden z dwóch warunków. Oczywistym jest, że drugi warunek jest zawsze prawdziwy, zatem Zapytanie wypisze wszystkie wiersze z powyższej tabeli. Ponadto podwójna pauza - znak komentarza - daje pewność, że do naszej komendy nie zostanie nic doklejone - komentarze nie są wykonywane.

Jest to niedopuszczalny wyciek danych - jedna komenda daje dostęp do wielu rekordów.



Przykład 2:
Polecenie DROP TABLE usuwa tabelę. Jeśli użyjemy go w powyższym formularzu, wszystkie rekordy tabeli zostaną skasowane i nie będzie można ich odczytać w żaden sposób.
Treść zapytania będzie następująca:

```
SELECT * FROM table where name = '' OR 1=1; DROP TABLE table; -
```

Polecenie zatem podaje wyniki komendy, po czym kasuje tabelę.

Przykład 3:
Bardzo niebezpieczna może okazać się w działaniu komenda UNION.
Przypomnijmy: UNION pokazuje kwerendę łączącą dwa zapytania O ILE odnoszą się one do jednakowej ilości kolumn. Zatem metodą prób i błędów jesteśmy w stanie dowiedzieć się o ilości kolumn w poleceniu.
Ponadto jesteśmy w stanie wyciągnąć całkowicie inne informacje. Przykładowo:
Niech tabela `people` zawiera dane osób:
- imię
-nazwisko
- adres zamieszkania
- numer telefonu
Przyjmijmy, że szukając osoby wyświetla się jedynie jej imię oraz nazwisko:

```
SELECT imie, nazwisko FROM people where name = '$input1' AND
nazwisko = '$input2';
```

Chcąc wyciągnąć numer telefonu oraz adres możemy w drugie pole formularza wstrzyknąć polecenie UNION. Niech naszym celem będzie użytkownik Jan Kowalski. Wtedy wstrzykujemy w pierwsze pole `Jan`, a w drugie `Kowalski' UNION SELECT adres, nr_tel FROM people; -` (Nie zapominajmy o znaku komentarza na końcu!)

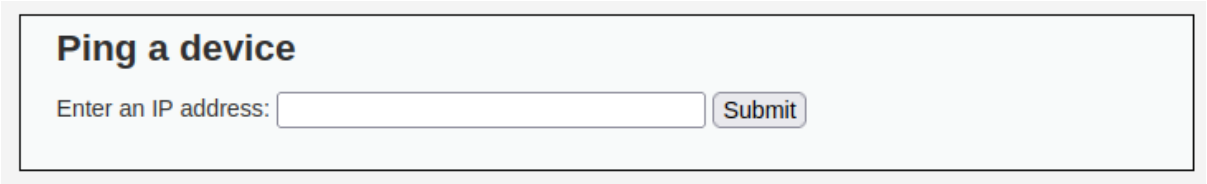
Co nam daje powyższy złośliwy kod? Wszystkie numery telefonów!
Wprawdzie nazwy kolumn zazwyczaj nie są znane, tak jak ilość w zapytaniu, jednak metodą prób i błędów możemy w niedługim czasie dopiąć swego.
Wobec powyższych zapytanie do bazy danych będzie brzmieć:

```
SELECT imie, nazwisko FROM people where name = 'JAN' AND nazwisko
= 'Kowalski' UNION SELECT adres, nr_tel FROM people; – ';
```

OS Command Injection

Tutaj metoda polega na wstrzyknięciu kodu do konsoli systemu. Najczęściej jest to system UNIX'owy, co można rozpoznać wstrzykując podstawowe komendy jak np. *pwd, ls* (odpowiednikiem w windowsie jest *dir*).
Co może dać nam nieautoryzowany dostęp do linii komend?
Odczyt, podmiana, usuwanie plików - to tylko kilka z poważnych zagrożeń, które niesie z sobą ten typ ataku.

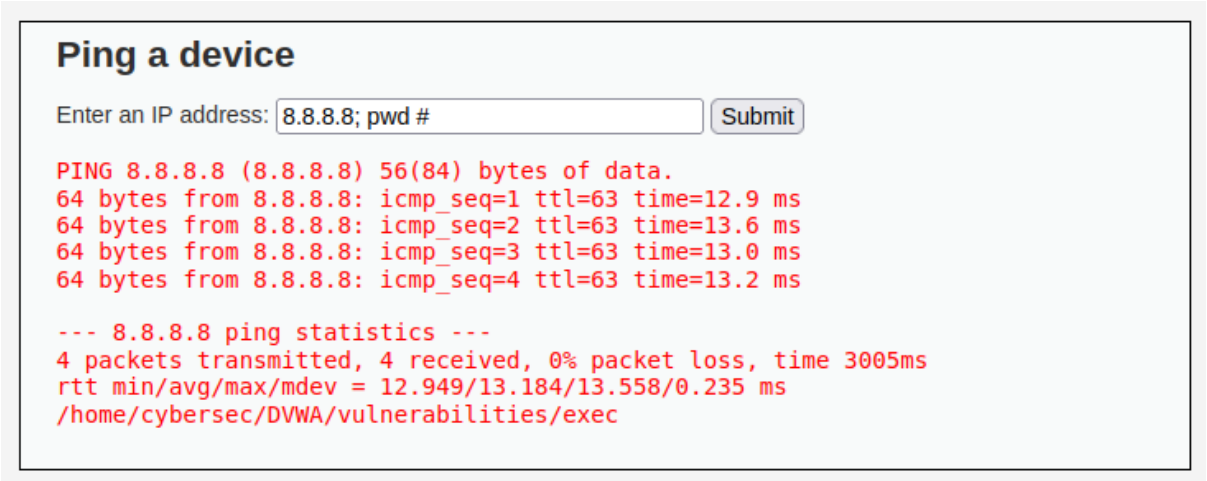
Przykład:
Narzędzie wysyłające ping na wskazany adres IP



W rzeczywistości komenda ma postać

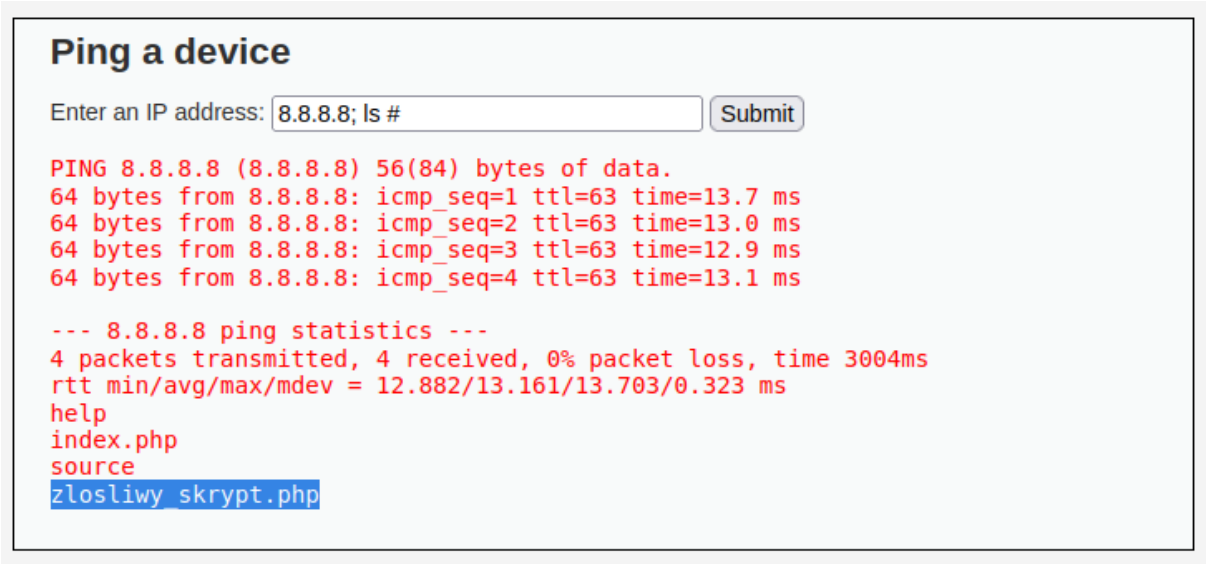
```
ping -c 4 `$input',
```

oraz wykonywana jest na serwerze systemu UNIX, co widać po doklejeniu do adresu IP komendy *pwd* po średniku - separatorze poleceń w języku skryptowym bash.
Podobnie jak wcześniej, nie zapominamy o znaku komentarza - w tym przypadku jest to #.



Jak łatwo sprawdzić - tworząc przykładowy plik - mamy dostęp do edycji plików w obecnym folderze.

Co to znaczy dla atakującego?
W folderze znajdują się skrypty w języku php. Podmiana na złośliwy skrypt w tej chwili jest banalnie proste.
Z punktu widzenia systemu istnieje prawdopodobieństwo dostępu do wielu innych folderów zawierających wrażliwe dane (nawet bez dostępu do *root'a*), jak np. chociażby folder *.ssh* z kluczami, które komenda *cat* może wyświetlić; wiele plików z folderów powyżej domowego jak np. *etc* są do wglądu dla zwykłego użytkownika.



Server-side template injection (SSTI)

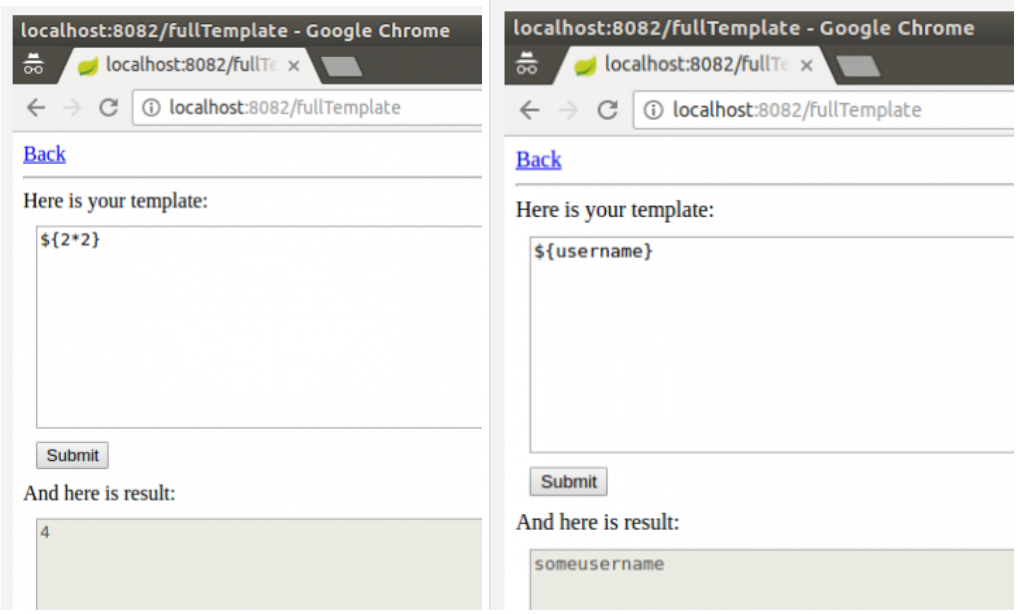
1. Co to są ataki typu SSTI?

Jak sama nazwa mówi, ta konkretna podatność typu *injection* opiera się na stronie serwera. Atakujący potrafi tak użyć natywnych silników szablonów, aby wstrzyknąć złośliwy payload, który potem zostanie wykonany właśnie po stronie serwera. Szablony to nic innego jak pewne ułożenia kodu, które mają ujednolicić pewne strony czy kody, innymi słowy - są to pewnego rodzaju wzorce. Skutkami tego typu ataków może być wiele różnych rzeczy - w zależności, który szablon atakujemy. Przykłady:

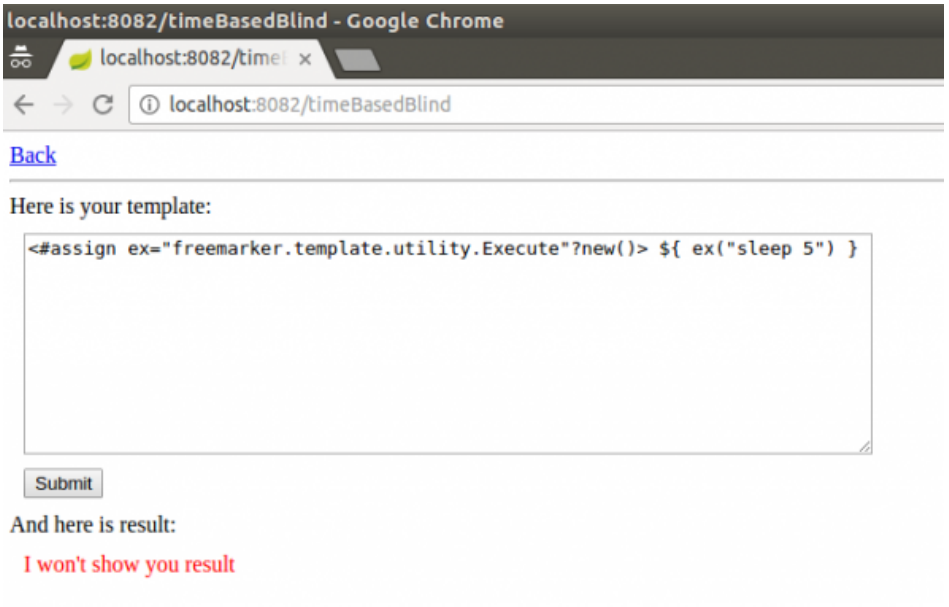
- możliwość wykonania własnego kodu po stronie serwera
- pełne przejęcie back-end serwera
- nawet jeśli atakujący nie będzie w stanie w pełni zdalnie wykonać kodu, wciąż może być w stanie przeanalizować pliki i dane znajdujące się po stronie serwera, które są niedostępne dla zwykłego użytkownika

2. Jak wykryć, że strona jest podatna na tego typu ataki oraz przykłady ataków

Oczywistym faktem jest to, że strona musi przyjmować jakieś dane od użytkownika - jeszcze łatwiej będzie, gdy strona będzie te dane w jakiś sposób zwracała. Najczęściej staramy się w prosty sposób sprawdzić, jak pola działają z wyrażeniami arytmetycznymi czy sprawdzenie użytkownika, np:



Są zatem opcje, że albo nieistniejąca zmienna została zupełnie zignorowana, albo możemy spowodować błąd, o którym sama strona nas może powiadomić. W drugim wypadku możliwe jest zdobycie wiedzy, jakiego szablonu silnika używa strona. Poniżej przykładowy atak, ze znalezionym szablonem *Freemaker*:



Powyższy payload opóźnia pracę serwera.

Cross site scripting (XSS)

1. Co to jest XSS?

Kolejnym atakiem typu injection jest XSS, czyli *cross site scripting*. Atakujący używa aplikacji webowej do przekazania złośliwego kodu do innego końcowego użytkownika. Atak wymaga pola na podanie inputu przez użytkownika, dzięki któremu potencjalnie możemy wykonać własny kod. Według przeglądarki, script pochodzi od zaufanego źródła, dzięki czemu przeglądarka bez namysłu wykonana podane linijki kodu. W polu input przeglądarka może przyjmować takie rzeczy jak cookies, tokeny sesji czy inne wrażliwe informacje, którymi operuje przeglądarka.

2. Typy ataków:

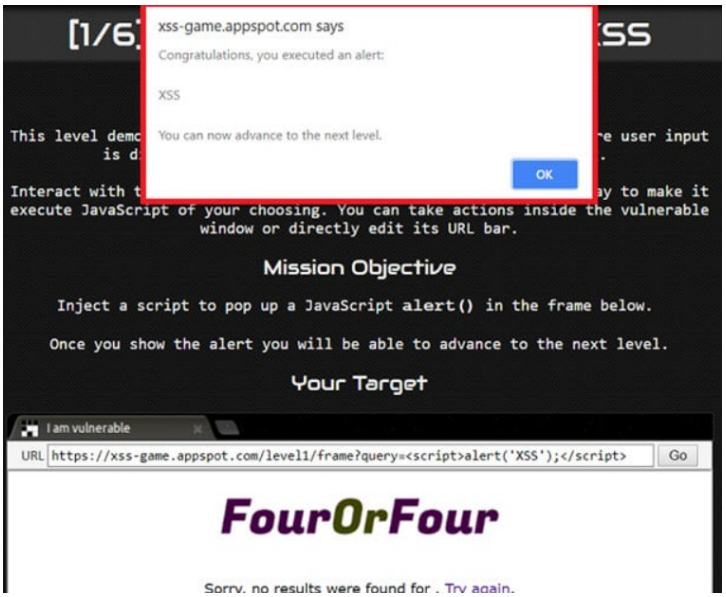
- Reflected XSS - przeglądarka bezpośrednio zwraca błąd, wynik wyszukiwania czy jakąś inną odpowiedź, która była spowodowana inputem użytkownika. Sam input nie został zbadany przez przeglądarkę pod kątem bezpieczeństwa.

- Stored XSS - input użytkownika jest przechowywany przez serwer, np. w bazie danych. Przewiduje się, że payloady będą przechowywane w np. HTML5 database, które to nigdy nie zostaną bezpośrednio przesłane do serwera.

- DOM based XSS - tutaj bazujemy na strukturze DOM - danych, które nie wypłyną dalej do serwera. Są to np. *document.write* czy *document.location.href*.

3. Przykłady ataków

Poniżej podano ataki, które najlepiej obrazują istotę cross site scritpingu:
<script>alert('XSS')</script>



Powyższa wiadomośc oznacza, że strona - dokładniej pole wyszukiwania przez użytkownika - jest podatne na ataki XSS, gdzie w tym wypadku jest to reflected cross site scripting.
Poniżej znajduje się ciekawy link, w którym zawarto najciekawsze i najczęściej używane payloady (jest ich aż ponad 6,5 tys.):
https://github.com/payloadbox/xss-payload-list/blob/master/Intruder/xss-payload-list.txt

A08. Software and Data Integrity Failures

W roku 2021 wprowadzono nową kategorię błędów związanych z integralnością oprogramowania i danych, która koncentruje się na popełnianiu założeń dotyczących aktualizacji oprogramowania, krytycznych danych oraz potoków CI/CD bez weryfikacji integralności. Jednym z najważniejszych czynników wpływu na tę kategorię, bazującym na danych z systemu Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS), jest mapa 10 najczęstszych słabych punktów (CWE) w tej kategorii. W ramach tej kategorii znajduje się również problem niebezpiecznej deserializacji, który wcześniej był osobną kategorią (A8:2017).

Ponieważ w opisie tej kategorii znajduje się sporo technicznych wyrażeń, poniżej znajduje się słowniczek:

- Integralność oprogramowania i danych oznacza, że dane są przechowywane, przesyłane i przetwarzane w sposób bezpieczny i dokładny, a oprogramowanie działa zgodnie z oczekiwaniami i z zabezpieczeniami, które zostały na nim wprowadzone.

- Protokoły CI/CD (ang. Continuous Integration/Continuous Delivery) to procesy, które umożliwiają programistom dostarczanie oprogramowania do użytkowników w sposób ciągły i zautomatyzowany. CI/CD pomaga w szybszym dostarczaniu nowych funkcjonalności czy poprawianiu błędów.

- Common Vulnerability Scoring System (CVE/CVSS) jest narzędziem do oceny zagrożeń i podatności w oprogramowaniu. CVE/CVSS przypisuje wartości i punkty dla różnych typów zagrożeń, aby pomóc w zrozumieniu ich wpływu i priorytetów w zakresie działań naprawczych.

- Mapa 10 najczęstszych słabych punktów (CWE) to narzędzie używane przez organizacje do oceny i priorytetyzowania podatności w oprogramowaniu. CWE to skrót od "Common Weakness Enumeration", czyli powszechna numeracja słabości. CWE to lista znanych podatności w oprogramowaniu, która opisuje, jakie błędy mogą prowadzić do zagrożenia bezpieczeństwa. Mapa CWE 10 zawiera dziesięć najczęstszych słabych punktów, które stanowią największe ryzyko dla bezpieczeństwa oprogramowania. Każdy z tych słabych punktów ma swój unikalny numer identyfikacyjny i opisuje różne sposoby, w jakie programista może nieumyślnie wprowadzić podatność do oprogramowania.

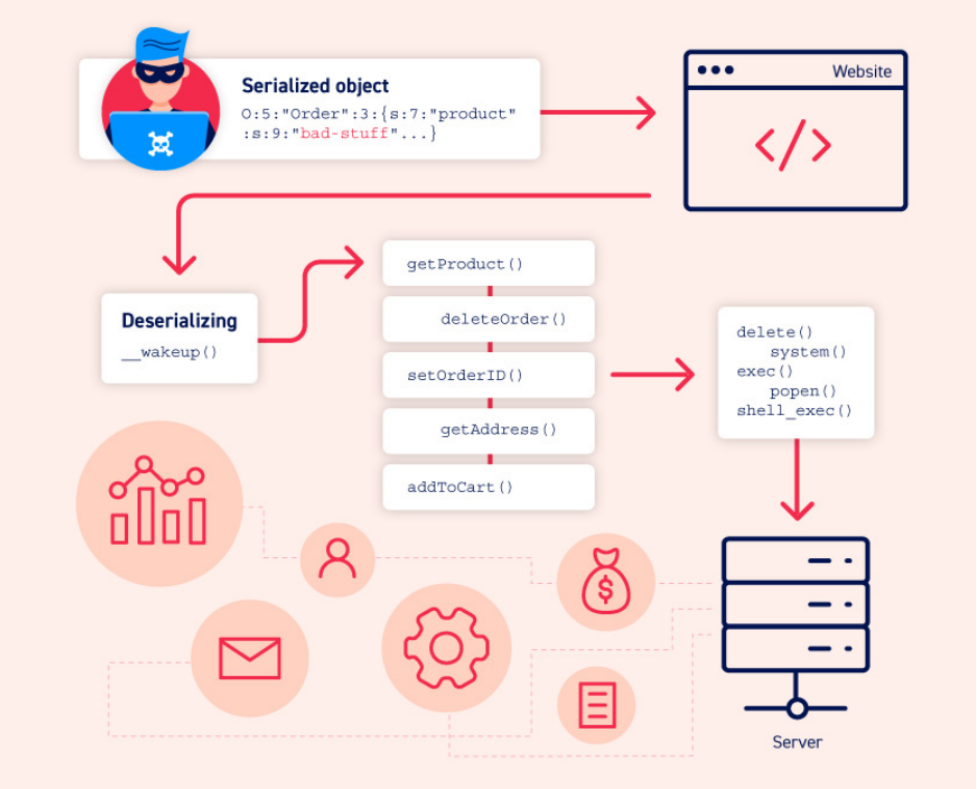
- Niebezpieczna deserializacja to sytuacja, gdy obiekt serializowany (przekształcony na postać tekstową) jest deserializowany (przekształcony z

postaci tekstowej na obiekt) w sposób, który może być wykorzystany przez atakującego w celu przejęcia kontroli nad aplikacją.

Jak zapobiegać?

- Wykorzystaj podpisy cyfrowe lub podobne mechanizmy, aby zweryfikować, że oprogramowanie lub dane pochodzą z oczekiwanego źródła i nie zostały zmienione.
- Upewnij się, że biblioteki i zależności, takie jak np. npm lub Maven, korzystają z zaufanych repozytoriów. Jeśli masz wyższy poziom ryzyka, rozważ utworzenie wewnętrznego repozytorium, które będzie sprawdzone i uznane za dobre.
- Upewnij się, że istnieje proces przeglądu zmian kodu i konfiguracji, aby zminimalizować szansę na wprowadzenie złośliwego kodu lub konfiguracji do Twojego procesu wytwarzania oprogramowania.
- Upewnij się, że twoje CI/CD (Continuous Integration/Continuous Delivery) ma odpowiednie odseparowanie, konfigurację i kontrolę dostępu, aby zapewnić integralność kodu przepływającego przez procesy budowania i wdrażania.

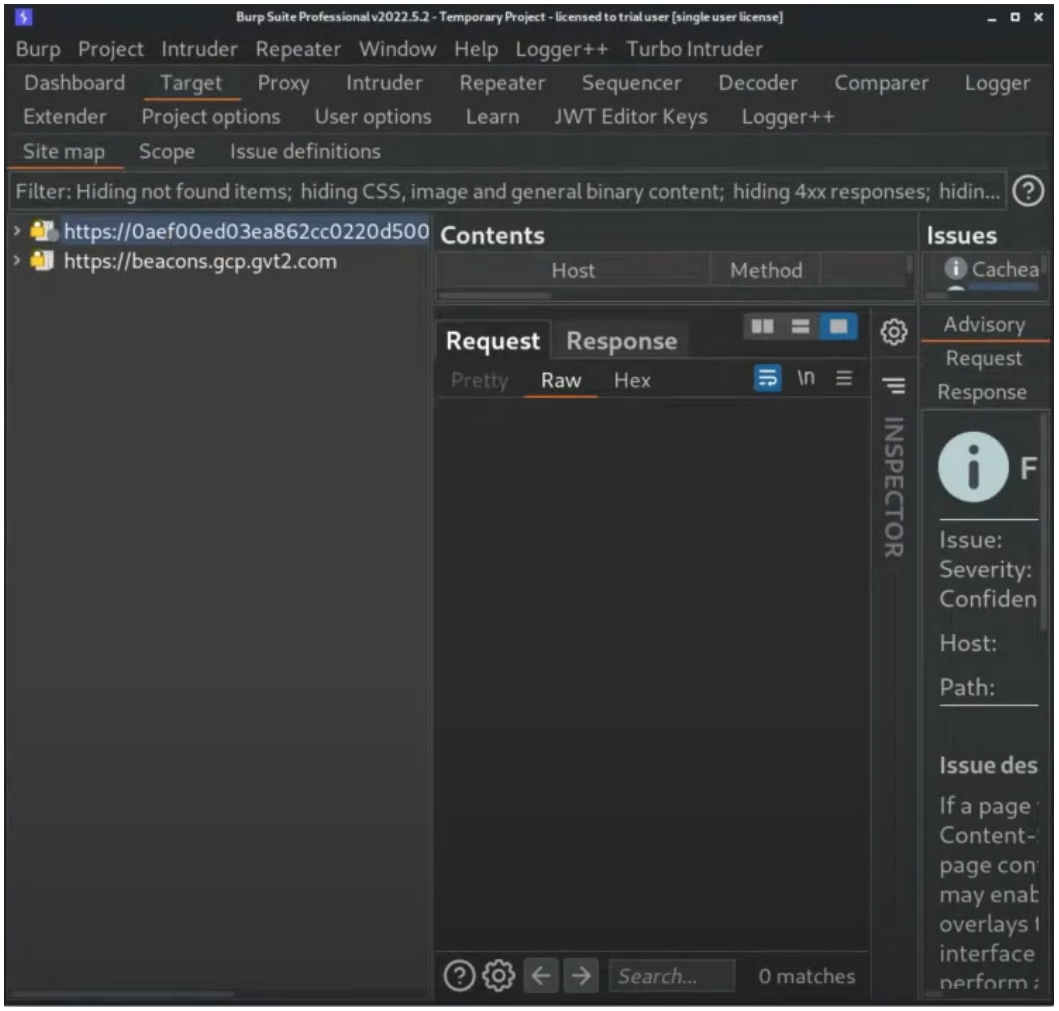
Niebezpieczne deserializacja



obrazek ze strony PortSwigger

Tak jak wyżej jest opisane: niebezpieczna deserializacja to sytuacja, gdy dane użytkownika są deserializowane przez stronę internetową, umożliwiając atakującemu manipulowanie serializowanymi obiektami. Atakujący może nawet zastąpić serializowany obiekt obiektem z innej klasy. Ta podatność jest nazywana czasami luką w zabezpieczeniach "wstrzykiwania obiektów". Ataki oparte na deserializacji mogą zostać zakończone przed zakończeniem samej deserializacji, co oznacza, że nawet silnie typowane strony internetowe mogą być podatne na tę technikę.

Jak sprawdzić czy strona jest podatna na niebezpieczną deserializacje?
Wiele aplikacji takich jak Burp Suite automatycznie oznaczy wszelkie wiadomości HTTP, które wydają się zawierać serializowane obiekty.



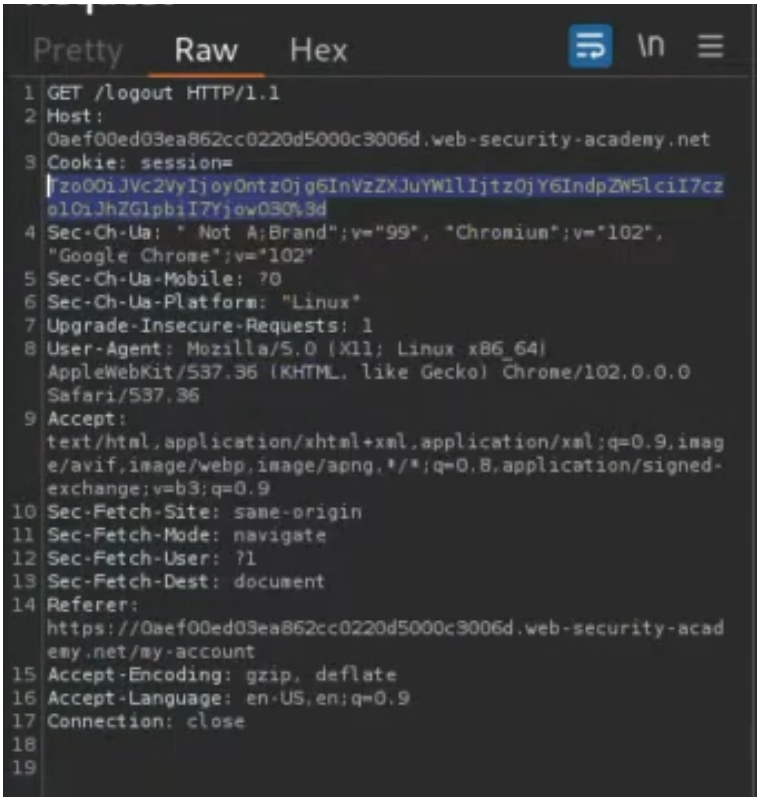
Format serializacji w PHP:
Obiekt User, który ma poniższe atrybuty:

```
$user->username = "carlos";
$user->isAdmin = false;
```

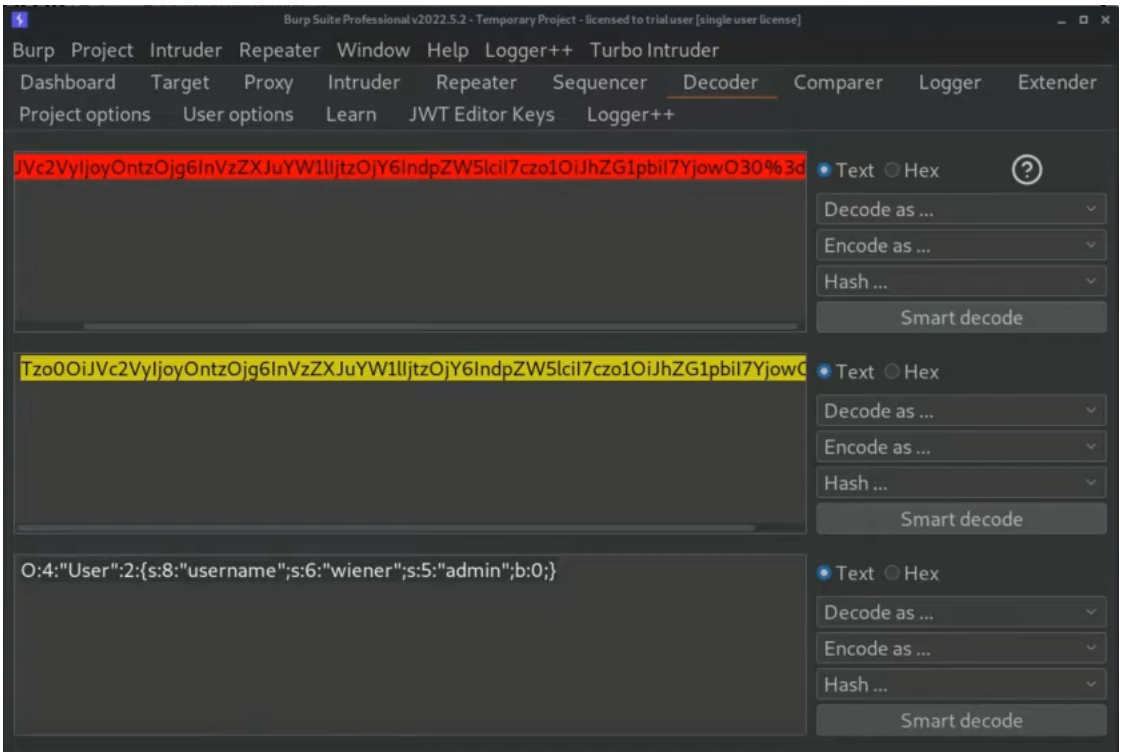
Po serializacji będzie wyglądał tak:

```
O:4:"User":2:{s:8:"username";s:6:"carlos";s:7:"isAdmin";b:0;}
```

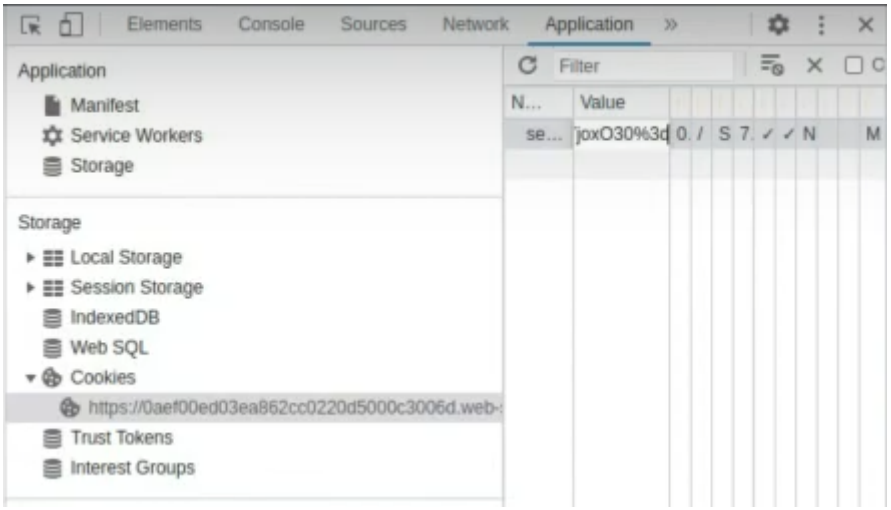
Przykład ataku:
Jeśli osoba atakująca wykryje serializowany obiekt w żądaniu HTTP, może go zdekodować i uzyskać strumień bajtów jak w powyższym przykładzie.



Widzimy, że w sesyjnym pliku cookie są zapisane jakieś informacje, po dekodowaniu uzyskujemy obiekt:
`O:4:"User":2:{s:8:"username";s:6:"wiener";s:5:"admin";b:0;}`



W naszym przykładzie możemy zmienić atrybut admin na wartość 1, ponownie zakodować obiekt i nadpisać swój aktualny plik cookie zmodyfikowaną wartością w polu Value.



Uzyskamy wtedy wszystkie uprawnienia admina m.in. możliwość usunięcia użytkowników.

Jak zapobiegać lukom niezabezpieczonej deserializacji:

1. Unikaj deserializacji danych wejściowych użytkownika, chyba że jest to absolutnie konieczne.

2. Jeśli musisz deserializować dane z niezaufanych źródeł, zastosuj solidne środki, aby upewnić się, że dane nie zostały naruszone.

3. Zaimplementuj podpis cyfrowy, aby sprawdzić integralność danych, ale upewnij się, że wszelkie kontrole odbywają się przed rozpoczęciem procesu deserializacji.

Bibliografia:

https://owasp.org/www-project-top-ten/
https://www.hahwul.com/cullinan/history-of-owasp-top-10/
https://cyolo.io/blog/owasp-top-10/owasp-top-10-injection/
https://sekurak.pl/podatnosc-server-side-template-injections/
https://portswigger.net/web-security/server-side-template-injection
https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection
https://www.imperva.com/learn/application-security/server-side-template-injection-ssti/
https://owasp.org/www-community/attacks/xss/
https://www.softwaretestinghelp.com/cross-site-scripting-xss-attack-test/
https://portswigger.net/web-security/deserialization/exploiting