**Raport dotyczący bezpieczeństwa aplikacji „[Damn Small Vulnerable Web](https://github.com/stamparm/DSVW)”**

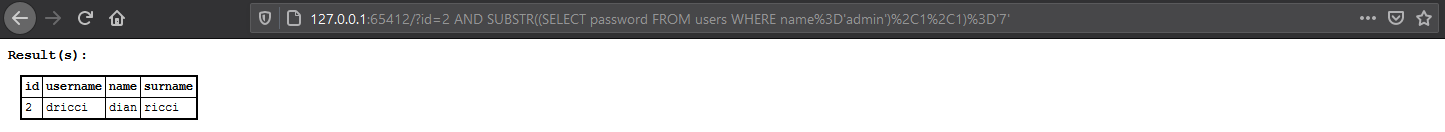
**Damn Small Vulnerable Web** (DSVW) to celowo podatna na ataki aplikacja internetowa napisana w mniej niż 100 wierszach kodu, stworzona do celów edukacyjnych. Obsługuje większość (najpopularniejszych) luk w zabezpieczeniach aplikacji internetowych wraz z odpowiednimi atakami. W przedstawionym raporcie, korzystając z wcześniej wspomnianej aplikacji przedstawię wybrane ataki, luki w zabezpieczeniach oraz sposoby ich zapobiegania.

1) **Blind SQL Injection**

Atak typu SQL Injection polega na „wstrzyknięciu” zapytania SQL za pośrednictwem danych wejściowych od klienta do aplikacji. Pozwala to na odczyt wrażliwych danych bazy oraz wykonywaniu na niej operacji wstawiania, usuwania lub modyfikacji istniejących rekordów.

Hasło administratora:





[**http://127.0.0.1:65412/?id=2%20AND%20SUBSTR((SELECT%20password%20FROM%20users%20WHERE%20name%3D%27admin%27)%2C1%2C1)%3D%277%27**](http://127.0.0.1:65412/?id=2%20AND%20SUBSTR((SELECT%20password%20FROM%20users%20WHERE%20name%3D%27admin%27)%2C1%2C1)%3D%277%27)

(%20 = „SPACE”, %3D = „=” , %27 = „ **’** ”, %2C = „ **,** ”)

?id=2 AND SUBSTR((SELECT password FROM users WHERE name = ‘admin’),1,1)=’7’

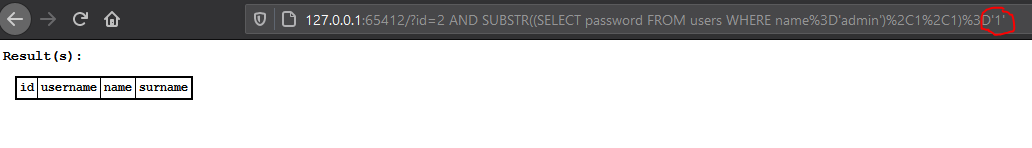
SUBSTR = wycięcie części textu

SELECT SUBSTR("SQL Tutorial", 5, 3) -> „Tut”

Przedstawiony przykład wysyła na serwer zapytanie o treści:

„Pobierz użytkownika o id = 2 i czy pierwsza litera z hasła użytkownika o nazwie admin równa jest ‘7’ ?”

Jeśli te dwa warunki zostaną spełnione tzn. użytkownik o id=2 istnieje oraz pierwsza litera użytkownika o nazwie admin równa jest ‘7’ (True and True) to serwer zwróci odpowiedni wynik wyszukiwania. Jeśli chcielibyśmy sprawdzić czy pierwszą literą jest cyfra 1 to otrzymamy wynik negatywny bo jeden z przedstawionych warunków jest fałszem.

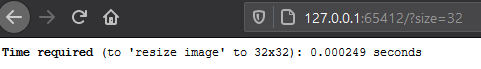


2) **Denial of Service**

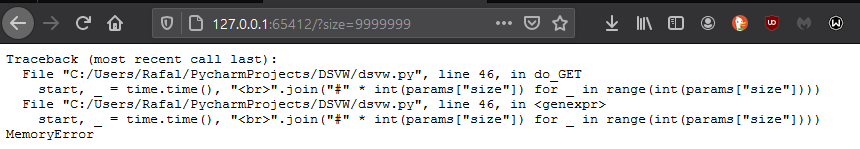
Atak polega zwykle na przeciążeniu aplikacji serwującej określone dane czy obsługującej danych klientów (np. wyczerpanie limitu wolnych gniazd dla serwerów FTP czy WWW), zapełnienie całego systemu plików tak, by dogrywanie kolejnych informacji nie było możliwe (w szczególności serwery FTP), czy po prostu wykorzystanie błędu powodującego załamanie się pracy aplikacji.

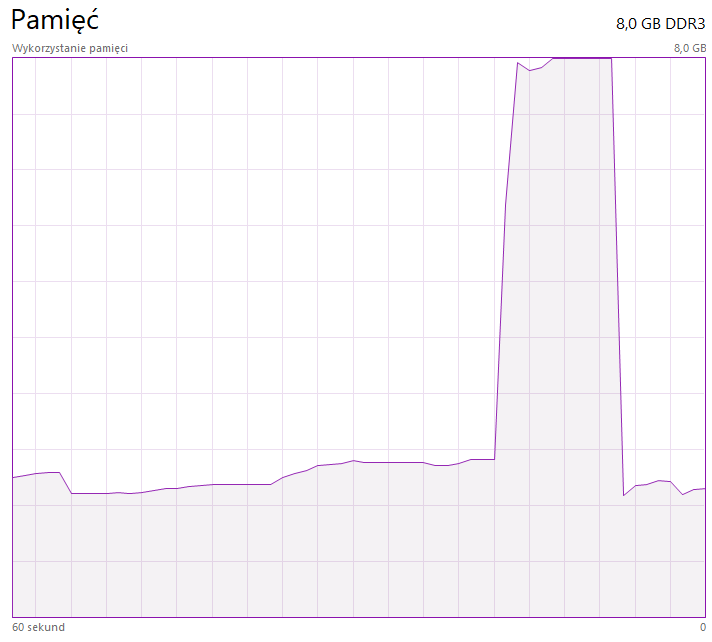
W aplikacji [**Damn Small Vulnerable Web**](https://github.com/stamparm/DSVW) atak Denial of Service przedstawiony jest jako oferowana przez aplikację usługa zmiany rozmiaru obrazka.

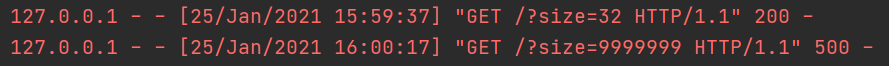
W przypadku podania w miarę „zwyczajnych” rozmiarów otrzymujemy następujący wynik:



Brak zastosowania warunków określających maksymalny rozmiar obrazka umożliwia klientom wprowadzanie rozmiarów skalowania przekraczających możliwości obliczeniowe serwera.



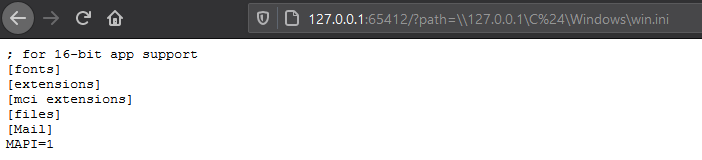




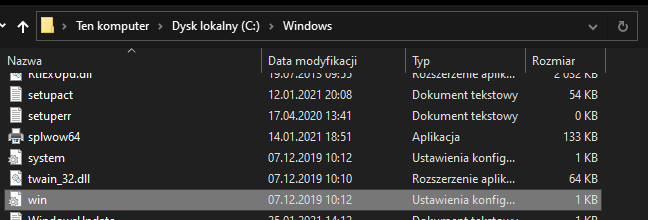
3) **Server Side Request Forgery**

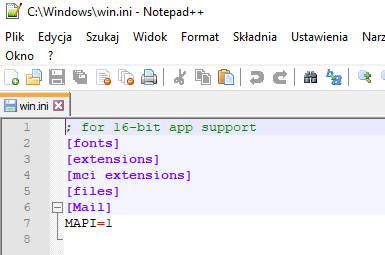
W przypadku ataku polegającego na fałszowaniu żądań po stronie serwera (SSRF) osoba atakująca może nadużywać funkcji serwera w celu odczytu lub aktualizacji zasobów wewnętrznych. Atakujący może podać lub zmodyfikować adres URL, który kod działający na serwerze będzie czytać lub przesyłać dane. Poprzez staranny wybór adresów URL atakujący może być w stanie odczytać konfigurację serwera, taką jak metadane AWS, połączyć się z usługami wewnętrznymi, takimi jak protokół http baz danych lub wysyłać żądania do usług wewnętrznych, które nie mają być ujawniane.

Przykład odczytania pliku win.ini przez podanie odpowiednio zmodyfikowanego adresu URL:



Ten sam plik na dysku:





**Zabezpieczania przed atakami SQL Injection:**

* **Zabezpieczanie na poziomie aplikacji**

Podstawowym sposobem zabezpieczania przed SQL injection jest niedopuszczenie do nieuprawnionej zmiany wykonywanego zapytania. W PHP można to zrobić, poprzez wykonanie na każdym tekstowym parametrze wykorzystywanym do budowy zapytania wbudowanej funkcji addslashes(), która dodaje backslash przed znakami, takimi jak ', " czy \, dzięki czemu znaki te nie są traktowane jak znaki specjalne.

* **Zabezpieczenie na poziomie bazy danych**

Udostępnienie użytkownikowi bazy tylko niezbędnych uprawnień nie da całkowitej ochrony, jednak pozwoli na minimalizację szkód. Przykładowo – niewiele aplikacji potrzebuje uprawnienia do kasowania tabel z bazy danych. Podobny efekt – czyli zmniejszenie możliwości atakującego – uzyskać można wyłączając niepotrzebną funkcjonalność na poziomie samego silnika. Pomocą mogą służyć również procedury składowane, dzięki którym zapytanie budowane jest po stronie bazy danych i aplikacja nie ma bezpośredniego wpływu na jego postać, chociaż i w tym przypadku skonstruowanie ataku nie jest niemożliwe.

* **Zabezpieczanie na poziomie serwera aplikacji/www**

Możliwa jest instalacja dodatkowych modułów do serwera warstwy aplikacyjnej (np. mod\_security do serwera Apache) filtrujących według zdefiniowanych reguł przychodzące żądania i blokujących te potencjalnie groźne. Reguły pozwalają na wychwycenie typowych uniwersalnych ataków lub znanych luk w popularnych aplikacjach. Wadą rozwiązania jest możliwość zablokowania także pożądanych wywołań (np. w aplikacji do zarządzania bazą danych).

**Zabezpieczania przed atakami Denial of Service:**

* **Blokowanie niepotrzebnego ruchu sieciowego**

Zmniejszenie powierzchni ataku najłatwiej uzyskać przez zablokowanie takiego rodzaju ruchu, co do którego jesteśmy pewni, że nie będzie przechodził przez dane urządzenie pośredniczące.

* **Blokowanie nieprawidłowego ruchu sieciowego**

Firewall oraz urządzenia pośredniczące powinny wykrywać podstawowe anomalie w ruchu sieciowym, analizując dane  przychodzące oraz wychodzące. Bardzo pomocne we wdrożeniach okażą się dokumenty BCP 38, BCP 84 oraz BCP 140, które m.in. podpowiedzą, dlaczego i w jaki sposób ograniczyć *IP Spoofing*

* **wykonanie hardeningu serwerów sieciowych i urządzeń pośredniczących** — „utwardzanie” konfiguracji systemów powinno być nastawione na ograniczanie liczby usług oraz na zamykanie niepotrzebnych portów na publicznych interfejsach sieciowych, wprowadzanie list kontroli dostępu oraz silnych metod uwierzytelniania. Warto też wdrożyć konfigurację, która pozwoli w mniej zachłanny sposób używać pamięci i mocy obliczeniowej urządzeń

**Zabezpieczania przed atakami SQL Injection:**

* **Podstawowe zabezpieczenia**

podstawową metodą zabezpieczenia się przed tego typu atakami jest sprawdzanie, czy podany przez użytkownika URL zaczyna się od http: lub https: (ewentualnie jeszcze ftp:)

* **Czarne listy**

Najczęściej wprowadzana są czarne listy takich wartości adresów URL, które nie są dopuszczalne. Na przykład, chcąc uniemożliwić skanowanie localhosta, twórca web aplikacji mógł pomyśleć o odrzucaniu prób żądań do URL-i zaczynających się od:

<http://localhost> , http://127.0.0.1

* **Filtry**

Innym często stosowanym filtrem przez twórców aplikacji webowych jest sprawdzanie rozszerzenia plików. Najczęstszym sposobem implementacji tego typu filtru jest po prostu sprawdzanie ostatnich znaków w URL-u. Jeśli URL kończy się np. na „.png”, uznajemy, że wskazuje on do pliku z obrazkiem.

Źródła:

1. <https://owasp.org/www-community/attacks/SQL_Injection#Boolean_Exploitation_Technique>
2. <https://pl.wikipedia.org/wiki/SQL_injection>
3. <https://owasp.org/www-community/attacks/Server_Side_Request_Forgery>
4. <https://pl.wikipedia.org/wiki/Blokada_us%C5%82ug>
5. <https://sekurak.pl/czym-jest-atak-ddos-cz-3-obrona/>
6. <https://sekurak.pl/czym-jest-podatnosc-ssrf-server-side-request-forgery/>