**Szymon Szczerbiński**

*Testy Penetracyjne, gr. 1*

Projekt w całości został przeprowadzony w środowisku przygotowanym uprzednio na potrzeby laboratoriów. Cała jego struktura (urządzenia, sieć, adresacja, ustawienia) jest identyczna w stosunku do poprzednio wykonywanych sprawozdań. Do środowiska została dodana jedynie maszyna zawierająca system Metasploitable 2, która została odpowiednio skonfigurowana, tak aby mogła komunikować się z innymi hostami w specjalnie przeznaczonej do tego odrębnej sieci.

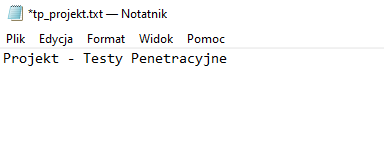
**Zad. 1**

W porozumieniu z prezesem firmy BitWise Dynamics Markiem Nowakiem została sporządzona umowa, na podstawie której wykonany będzie całkowity test penetracyjny, w ramach którego:

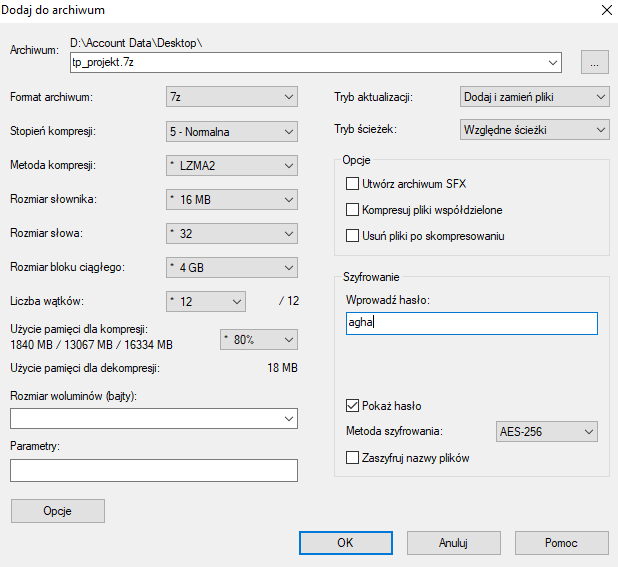
* Zidentyfikowane zostaną potencjalne luki w zabezpieczeniach
* Oceniona zostanie skuteczność systemu w reakcji na ataki oraz ogólna ocena poziomu bezpieczeństwa
* Test będzie odbywał się etapowo, tak aby nie zakłócić normalnego funkcjonowanie systemu
* Ustalone zostały ograniczenia i środki ostrożności
* Dozwolone zostały próby fizycznego dostępu
* Przeskanowane zostaną adresy i usługi dostępne publicznie
* Zostanie sprawdzona możliwość zdalnego przejęcia kontroli nad systemem
* Testom zostaną poddane zabezpieczenia kryptograficzne
* Rozważone zostaną metody obrony oraz potencjalnych ataków typu Injection, głównie SQL

**Zad. 2**

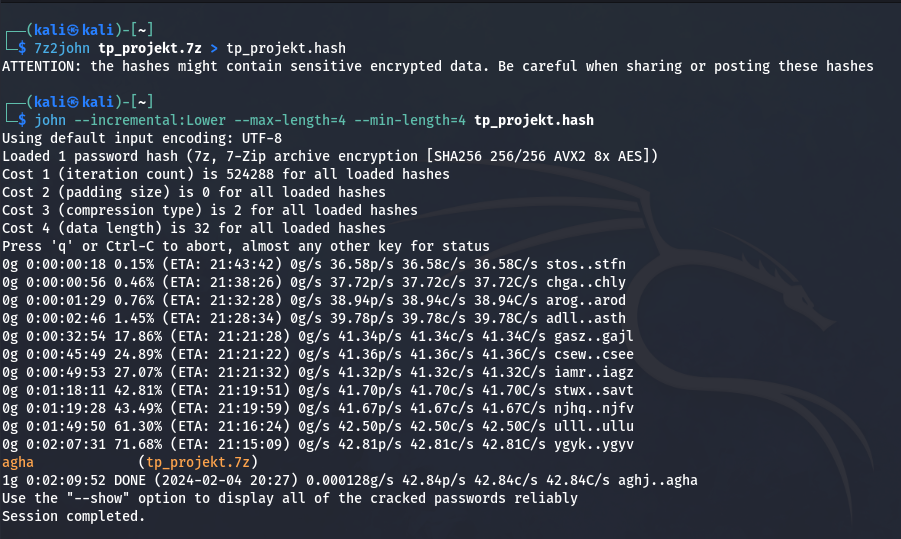
Stworzenie przykładowego pliku testowego w systemie Windows “tp\_projekt.txt” o zawartości “Projekt - Testy Penetracyjne”:



Kompresja wcześniej utworzonego pliku, za pomocą programu 7zip, do której wykorzystane zostało hasło “agha”. Przygotowany plik .7z został następnie przeniesiony na maszynę z systemem Kali Linux, na której zostanie poddany próbie ataku metodą brute-force.

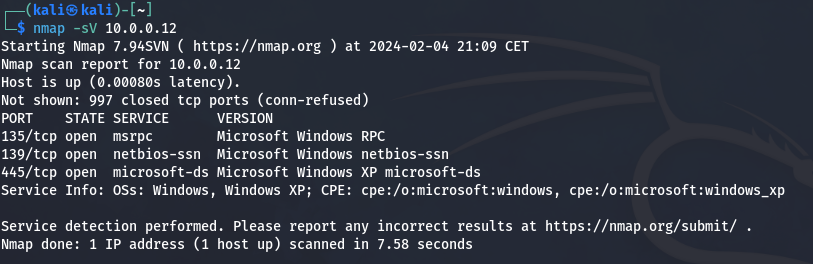


Po przeniesieniu pliku do systemu Kali Linux, wyciągam jego hash, jednocześnie dostosowując go do postaci przystępnej dla JTR za pomocą narzędzia 7z2john. Następnie pomyślnie dokonuję ataku metodą brute-force wykorzystując JTR w trybie incremental. Ze względu na fakt, że tym razem atak został przeprowadzany na słabszej maszynie zajął on trochę czasu oraz byłem zmuszony do ograniczenia liczby kombinacji haseł manipulując zestawem znaków oraz parametrami długości.

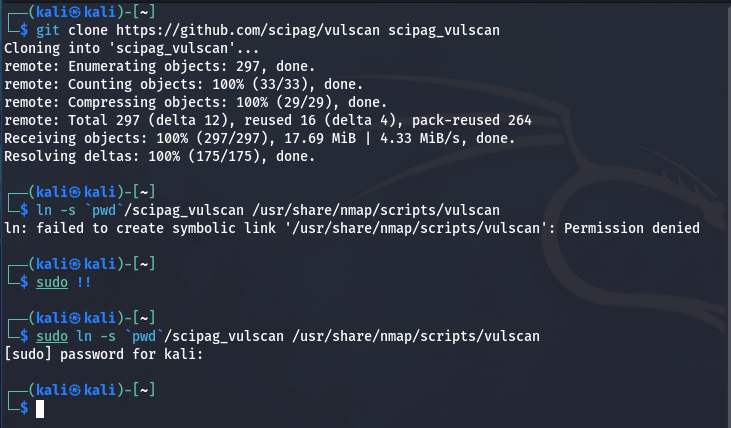


**Zad. 3**

Zadanie rozpocząłem od skanu otwartych portów atakowanego hosta za pomocą narzędzia nmap z wykorzystaniem parametru –sV. Program wykrył trzy otwarte porty tcp: 135, 139, 445, a więc postanowiłem wykorzystać właśnie te porty podczas dalszej analizy.

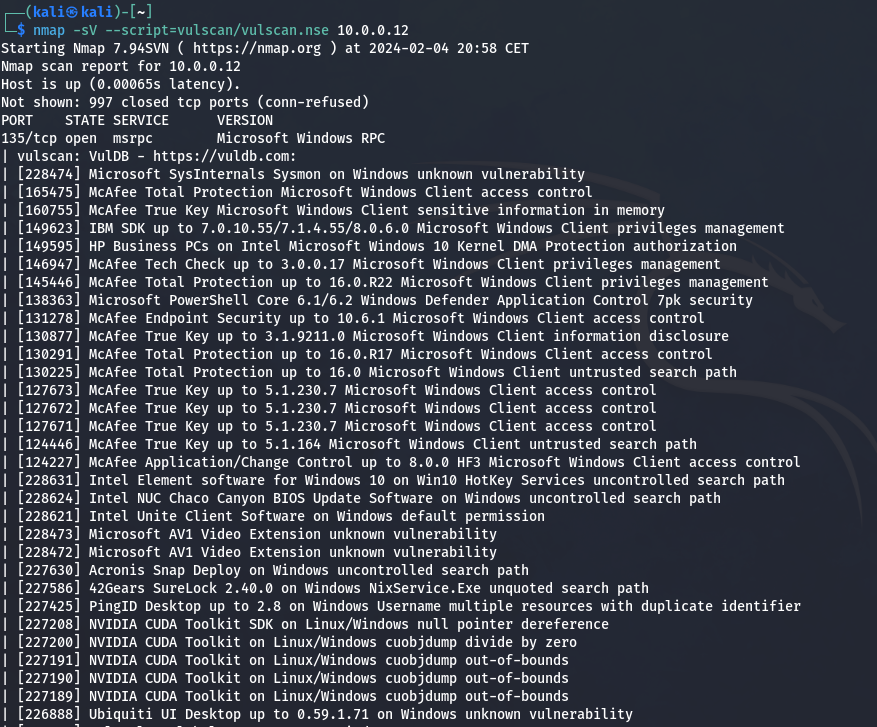


Następnym krokiem było pobranie skryptu vulscan z portalu github. Skrypt ten pozwala rozszerzyć działanie polecenia nmap –sV – po wykryciu otwartego portu oraz działającej na nim usługi skrypt wypisuje wszystkie adekwatne podatności. Podczas działania wykorzystuje on wiele źródeł takich jak zbiory klasyfikowane za pomocą CVE, zbiory OpenVas, VulDB, czy portalu securityfocus.



Skanowany jest system Windows XP, a więc podatności jest znacznie więcej niż być powinno. W rzeczywistości otwarte są jedynie 3 porty, a lista podatności wskazanych przez skrypt jest o wiele za długa, aby została zamieszczona w całości. Z tego powodu poniżej znajdują się zrzuty prezentujące jedynie początkowe rekordy.

Wykrycie otwartego portu 135 /tcp:



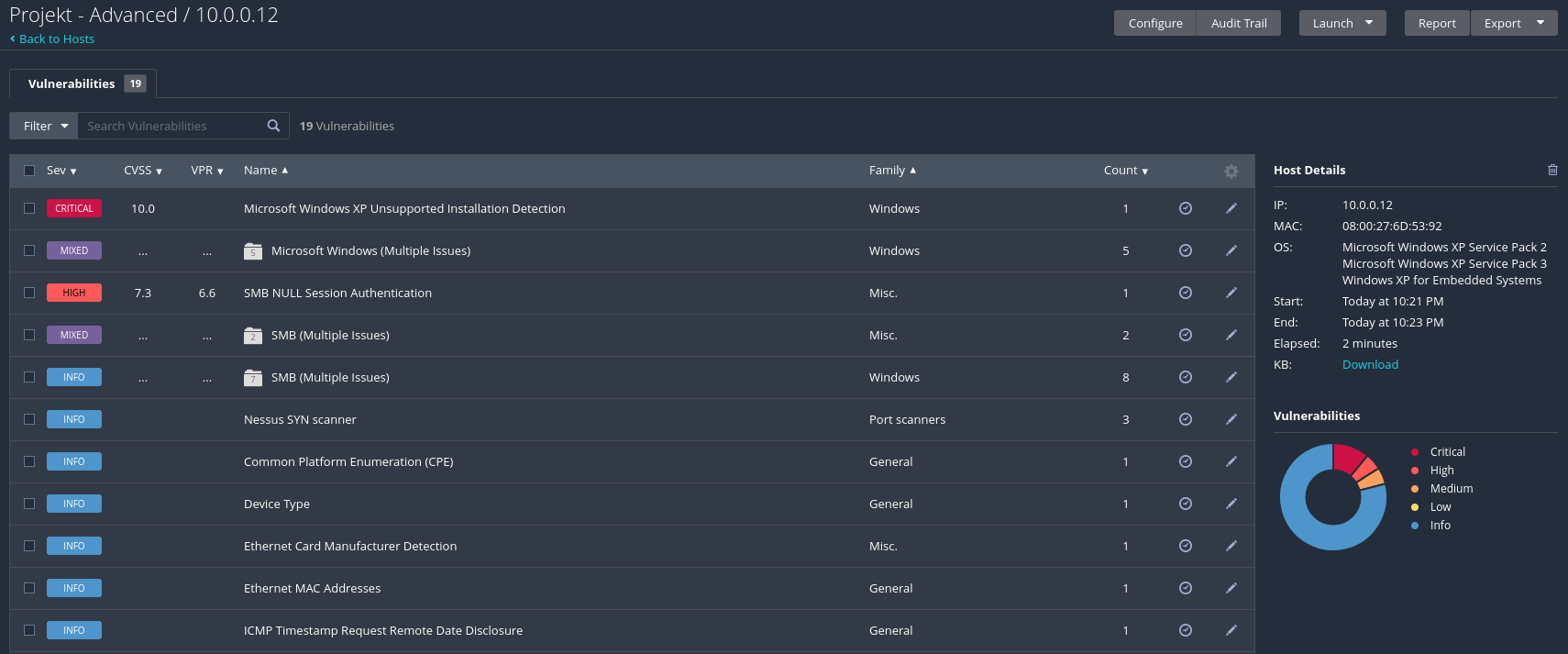
Podatności dla portu 137 /tcp:



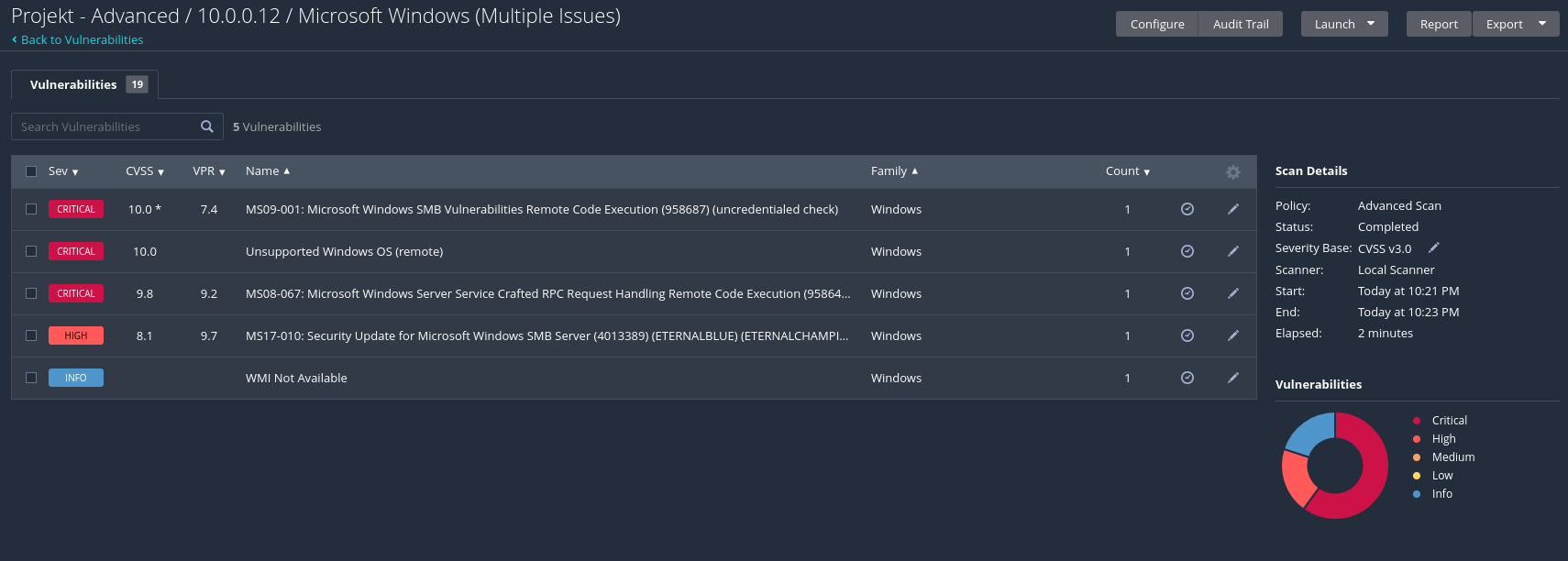
Po wykorzystaniu nmapa postanowiłem użyć także skanera Nessus. W tym celu utworzyłem nowy skan zaawansowany zawierający wszystkie pluginy podatności, a następnie skonfigurowałem go tak, aby przeskanował on jedynie maszynę z systemem Windows XP.

W wyniku otrzymałem zbiór informacji o systemie, a także parę rekordów potencjalnych podatności. Część z nich została oflagowana jako krytyczna oraz pogrupowana z innymi podatnościami z rodziny.

Wynik skanu Nessus’em:



Wykryte podatności krytyczne, powiązane z systemem Windows:



Wymienione zostały m.in podatności SMB: MS08-067 oraz MS09-001, a także MS17-010.

Jako że wszystkie z tych podatności otrzymały wysokie oceny ryzyka, oraz wymienione zostały w ramach skanu zarówno nmap’em jak i Nessus’em, to im postanowiłem poświęcić największą uwagę. Zdecydowanie rzetelność ocen potwierdzona została przez firmę Microsoft, która wydając łatki naprawiające te podatności oflagowała każdą z aktualizacji jako wymagana oraz krytyczna.

**MS08-067**

Jest to podatność usługi serwerowej, która dotyczyła systemów Windows XP, Vista, Server 2003 oraz Server 2008. Pozwalała ona na wykorzystanie sposobu w jaki wykorzystywane były żądania RPC. Odpowiednie przygotowanie żądania pozwalało atakującemu na zdalne wykonywanie arbitralnego kodu (RCE - Remote Code Execution). Podatność ta została przez nas wykorzystywana na potrzeby poprzednich laboratoriów, a samo jej użycie zajmuje jedynie 5s. Dzięki niej atakujący jest w stanie otrzymać całkowitą kontrolę nad systemem.

**MS09-001**

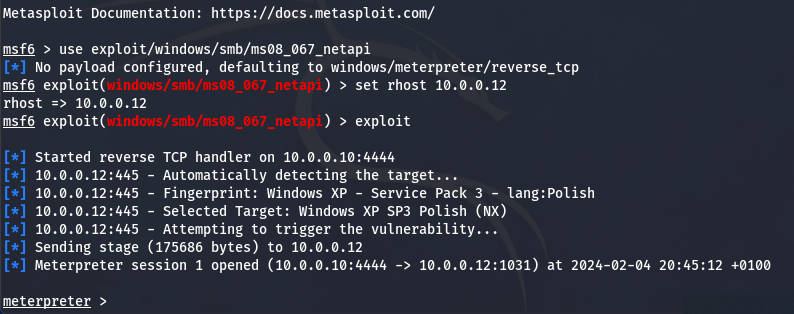
Podatność ta, jest całkiem podobna pod względem zastosowania do poprzedniej. Pozwala ona na nieautoryzowane wykonanie kodu poprzez odpowiednio spreparowane pakiety SMB. Skuteczne jej wykorzystanie pozwala na przeprowadzenie ataków DOS na system, a samo wykonanie arbitralnego kodu zostało oznaczone jako “teoretycznie możliwe” w tamtych czasach. Dotyczy ona systemów Windows 2000, XP, Vista, Server 2003 oraz Server 2008.

**MS17-010**

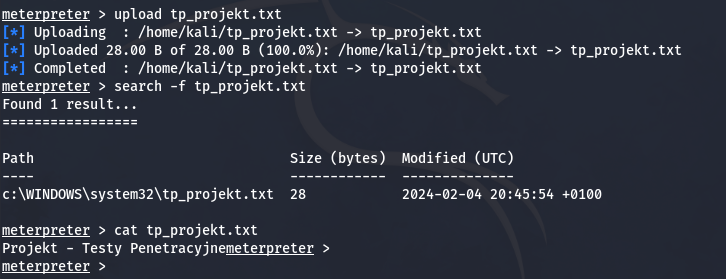
Tak jak w poprzednich przypadkach, podatny jest sposób w jaki używane są niektóre zapytania SMBv1. Jej wykorzystanie także pozwala na zdalne wykonanie kodu na atakowanym systemie, czyli de facto na otrzymanie pełnych uprawnień oraz całkowitej kontroli. Podatność ta dotyczy wszystkich systemów, aż do Windowsa 10 oraz Windows Server 2016. Exploit wykorzystywany był podczas ataku ransomware Wannacry, znana jest bardziej pod nazwą EternalBlue.

**Zad. 4**

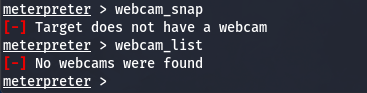
W ramach ataku wykorzystuję framework metasploit oraz moduł wykorzystujący wyżej opisaną podatność MS08-067. Odpowiednio konfiguruję zmienną rhost, ustawiając ją na adres hosta z systemem Windows XP; payload wybierany jest domyślnie jako reverse tcp, co w naszym przypadku jest odpowiednie. Następnie przechodzę do uruchomienia exploita oraz przejęcia systemu.



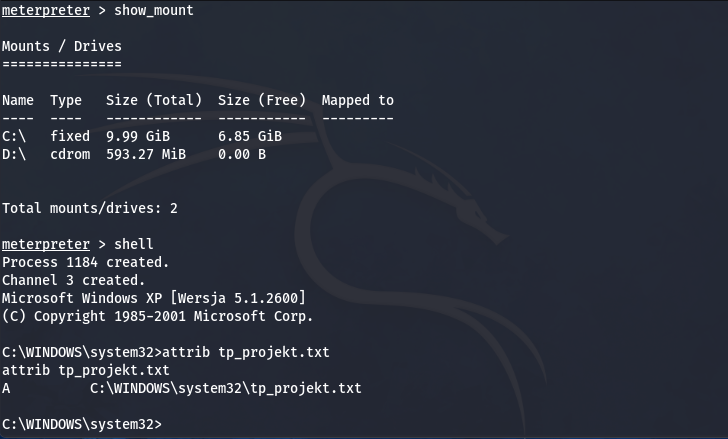
Po uzyskaniu kontroli nad systemem, moim pierwszym działaniem jest przesłanie wcześniej odszyfrowanego pliku tp\_projekt.txt na komputer ofiary. Wykorzystuję do tego polecenia upload, bez wskazania konkretnej ścieżki, czyli został on umieszczony w katalogu System32, co prezentuje komenda search –f tp\_projekt.txt. Zdalnie wyświetlając zawartość pliku, możemy zaobserwować, że operacja zakończyła się pomyślnie bez problemów.



Kolejnym celem jest zdalne włączenie kamery. Niestety system hosta nie jest w nią wyposażony (jest to maszyna wirtualna, bez dodatkowych urządzeń) dlatego pokazuję jedynie proof of concept. Aby wykonać zrzut z kamery należy wykorzystać polecenie webcam\_snap. Z wcześniej wymienionego powodu dostajemy informację o braku urządzenia w systemie, co pokazuje także webcam\_list.

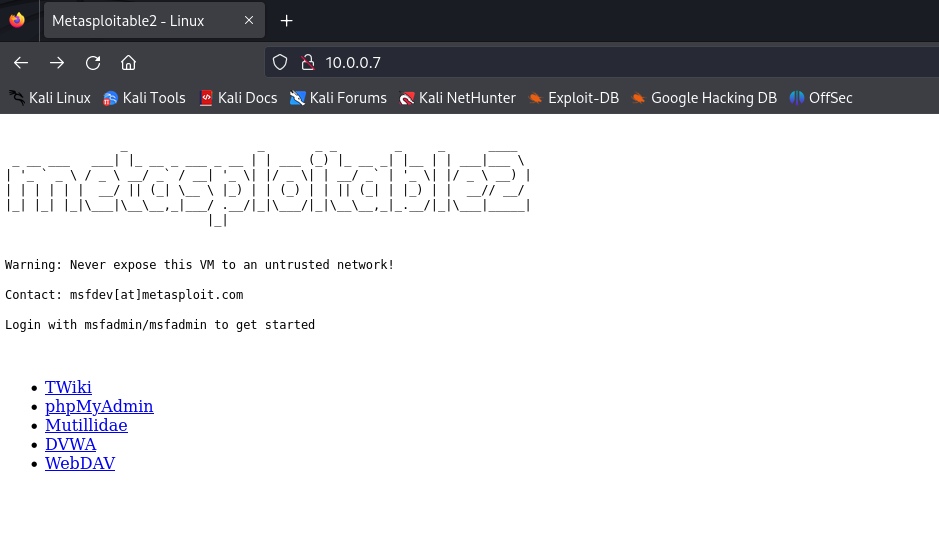


Ostatnimi poleceniami jakich używam w meterpreterze są: show\_mount, służące do wylistowania wszystkich dysków w systemie wraz z danymi o ich pojemności oraz typu – stacjonarny, przenośny, stacja cdrom, oraz polecenie shell, które pozwala na przeniesienie się z meterpretera do poziomu konsoli cmd z uprawnieniami użytkownika, którego sesja została przejęta. W naszym przypadku atakowana została usługa systemowa, a więc nasze akcje wykonywane są jako system.



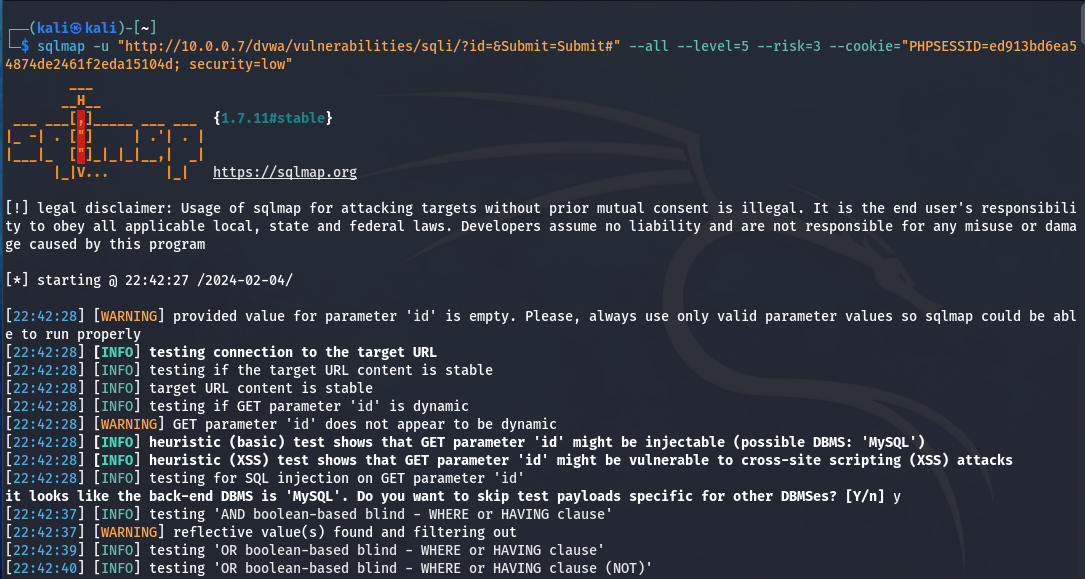
**Zad. 5**

Ostatnim punktem testu penetracyjnego jest instalacja systemu Metasploitable oraz wykorzystanie go do zaprezentowania podatności SQL Injection. W tym celu użyję domyślnie zainstalowanej aplikacji DVWA – Damn Vulnerable Web Application, która służy do wykorzystywania różnych podatności na wielu poziomach trudności.



Po uruchomieniu strony DVWA oraz przejścia do modułu poświęconego SQL Injection, kopiuję adres URL, a także parametry sesji zapisanej w ciasteczkach. Dzięki temu podczas automatycznego wykrywania podatności SQLi program nie będzie błędnie przekierowywany do strony logowania.

W celu prezentacji samego ataku SQLi wykorzystam narzędzie sqlmap, które automatycznie testuje wiele zapytań, w zależności od złożoności oraz skali ryzyka zdefiniowanej przez atakującego. Podczas ataku wykorzystuję najwyższy poziom złożoności, pozwalając tym na dokładne wykrycie podatności kosztem ryzyka, które można pominąć, ponieważ aplikacja działa lokalnie oraz posiadam jej backup.



Już po chwili od uruchomienia narzędzie wykryło, że parametr “id” jest podatny na atak SQLi, co zostało dokładnie opisane podczas działania aplikacji.



Po zakończeniu procesu testowania program exportuje wszystkie możliwe dane oraz zapisuje je lokalnie na dysku atakującego. Dzieje się tak na wskutek użycia parametru –all. Finalnie po zakończeniu ataku mam dostęp do całej bazy danych wykorzystywanej przez aplikację DVWA. Wyeksportowane zostały nie tylko tabele dotyczące modułu o SQLi, ale cała baza aplikacji. Atakujący w tej chwili posiada pełną wiedzę o jej działaniu oraz ma dostęp do wszystkich danych. Przykładowe tabele zamieściłem na zrzutach poniżej.

Tabela zawierająca wszystkie schematy wykorzystywane w bazie oraz jej kodowanie:

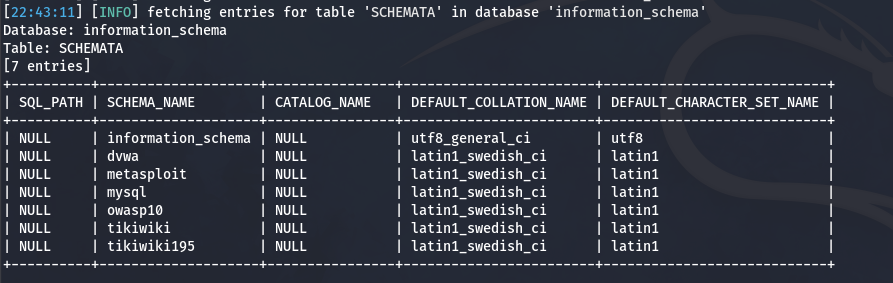


Tabela użytkowników bazy danych “dvwa” zawierająca nazwę użytkownika, hasło, imię oraz nazwisko:



Tabela “accounts” w baziw danych “owasp10” zawierająca nazwę użytkownika, nieszyfrowane hasło, cytat, a także zmienną typu boolean informującą o posiadanych uprawnieniach administratora:



Tabela “blogs\_table” w bazie danych “owasp10” zawierająca treść posta na blogu, datę opublikowania, a także autora wpisu, wykorzystywaną w ramach innego modułu DVWA:

