Ok 2. Środowiska wirtualne

Środowisko wirtualne (ang. **Virtual Environments**, VE) to oprogramowanie symulujące działanie sprzętu komputerowego. W odróżnieniu od tradycyjnych systemów fizycznych umożliwia ono jednoczesne funkcjonowanie wielu niezależnych systemów operacyjnych lub aplikacji na jednej maszynie [1]. Działają one niezależnie od fizycznej struktury sprzętu. Tworzenie środowisk wirtualnych możliwe jest dzięki wykorzystaniu technologii wirtualizacji. Jest to technologia, wykorzystująca środowisko logiczne do przekroczenia fizycznych ograniczeń sprzętowych [2].

Podstawowym elementem środowisk wirtualnych jest maszyna wirtualna (ang. **Virtual Machine**, VM). Jest to aplikacja, wykonujące program tak, jakby była fizycznym urządzeniem, więc można byłoby powiedzieć, że jest to „komputer”, działający wewnątrz fizycznego komputera. Aplikacja VM (nazywana „gościem”) uruchamia swój własny system operacyjny na rzeczywistej maszynie (zwanej „gospodarzem”). Wirtualny system operacyjny może być dowolny, np. Windows lub MacOS, i nie jest ograniczony do jednego systemu operacyjnego na maszynie gospodarza [3]. Każda maszyna wirtualna działa niezależnie i nie ma wpływu na działanie innych VM-ów.

2.1. Charakterystyka wirtualizacji

Wirtualizacja to technologia umożliwiająca tworzenie wielu odizolowanych środowisk komputerowych – zwanych maszynami wirtualnymi (VM) – na jednym fizycznym urządzeniu. Dzięki warstwie pośredniczącej, zwanej hipernadzorcą (**hypervisor**), każda maszyna wirtualna może działać jak odrębny komputer z własnym systemem operacyjnym i aplikacjami, niezależnie od innych instancji. To podejście pozwala na efektywne wykorzystanie zasobów sprzętowych, zwiększenie skalowalności, uproszczenie zarządzania oraz ograniczenie kosztów operacyjnych [4].

Hipernadzorca (hypervisor) to oprogramowanie, które umożliwia tworzenie i zarządzanie maszynami wirtualnymi poprzez oddzielenie systemów operacyjnych gości od fizycznej infrastruktury sprzętowej. W zależności od sposobu działania, wyróżniamy dwa główne typy:

* typ 1: natywny (bare-metal) – działa bezpośrednio na sprzęcie, bez potrzeby instalowania systemu operacyjnego gospodarza. Przykładowo: Microsoft Hyper-V,
* typ 2: hostowany – funkcjonuje jako aplikacja zainstalowana w ramach istniejącego systemu operacyjnego. Przykładowo: VirtualBox, VMware Workstation.

Dodatkowo, hypervisory można klasyfikować ze względu na sposób wirtualizacji:

* Pełna wirtualizacja – system gościa działa bez konieczności modyfikacji, nieświadomy, że funkcjonuje w środowisku wirtualnym, jest w pełni niezależny.
* Wirtualizacja wspierana sprzętowo – wykorzystuje specjalne funkcje procesora, takie jak Intel VT-x czy AMD-V, w celu optymalizacji pracy maszyn wirtualnych.
* Parawirtualizacja – wymaga modyfikacji systemu gościa, który jest świadomy, że działa w środowisku wirtualnym i potrafi efektywnie współpracować z hipernadzorcą [5].

2.2. VMware jako platforma testowa

Vmware Workstation Pro (od firmy Broadcom) jest w pełni zwirtualizowanym środowiskiem sprzętowym dla systemu operacyjnego gościa. Program ten obsługuje wiele systemów operacyjnych gospodarza, w tym Windows, Linux oraz macOS, i został zaprojektowany z myślą o maksymalnym wykorzystaniu fizycznych zasobów komputera, co przekłada się na wysoką wydajność działania maszyn wirtualnych [4].

Oprogramowanie to stara się wykonywać instrukcje bezpośrednio na procesorze gospodarza, o ile jest to możliwe. W przypadkach, gdy bezpośrednie wykonanie kodu nie jest wspierane przez sprzęt, oprogramowanie korzysta z techniki dynamicznego tłumaczenia binarnego, która pozwala na przekształcanie instrukcji w czasie rzeczywistym. Tak przetworzony kod jest przechowywany w pamięci RAM, co umożliwia dalsze działanie maszyny wirtualnej z dużą szybkością – według producenta, z wydajnością sięgającą ponad 80% względem natywnego systemu [4].

VMware emuluje podstawowe elementy sprzętowe, takie jak karta graficzna (czego nie oferuje np., VirtualBox od firmy Oracle), karta sieciowa czy kontrolery dysków, a także umożliwia dostęp do urządzeń USB, portów szeregowych i równoległych za pomocą sterowników pośredniczących. Warto jednak zauważyć, że przenoszenie maszyn wirtualnych między różnymi komputerami, szczególnie z odmiennymi architekturami procesora lub liczbą rdzeni, może wymagać dodatkowej konfiguracji ze względu na różnice w zestawach instrukcji [4].

Dzięki powyższym rozwiązaniom, VMware Workstation Pro wyróżnia się wysoką wydajnością i wszechstronnością, co czyni go solidnym narzędziem do prowadzenia badań nad bezpieczeństwem w środowiskach wirtualnych.

A screenshot of a computer

AI-generated content may be incorrect.

Fot. 1 Widok podstawowy Vmware Workstation w wersji 17.6.3

Podstawowymi funkcjami dostarczanymi przez środowisko Vmware Workstation są m.in.:

- tworzenie maszyn wirtualnych z wybraną konfiguracją systemu operacyjnego, pamięci RAM, przestrzeni dyskowej,

- snapshoty, czyli zapisywanie stanu maszyny wirtualnej w wybranym przez użytkownika momencie, co daje możliwość przywrócenia zapisanego obrazu,

- klonowanie maszyn wirtualnych,

- uruchamianie maszyn wirtualnych jednocześnie przy jednoczesnym ich izolowaniu.

- zaawansowana konfiguracja sieciowa, pozwalająca na definiowanie topologii sieci wirtualnych (m.in. NAT, bridge, host-only)

- połączenia ze zdalnymi serwerami, np. z Vmware ESXi.

2.3 Konfiguracja środowiska testowego

Do przeprowadzenia badań wykorzystano komputer stacjonarny wyposażony w procesor AMD Ryzen 5 3600 (6 rdzeni, 12 wątków), 64 GB pamięci RAM DDR4 (3600 MHz), dysk SSD M.2 o pojemności 2 TB oraz kartę graficzną NVIDIA GeForce RTX 3060 Ti. Systemem operacyjnym gospodarza był Windows 11 Pro (wersja 24H2).

Środowisko wirtualne zostało zrealizowane przy użyciu VMware Workstation Pro 17.6.3. W ramach testów utworzono dwie maszyny wirtualne: jedną z systemem Windows 10 Pro (wersja 22H2, 64-bit), drugą z systemem Ubuntu 24.04.2 LTS. Obie maszyny zostały skonfigurowane zgodnie z poniższymi założeniami:

* **Procesor**: przypisano 4 rdzenie wirtualne w celu zapewnienia stabilności i wydajności przy jednoczesnym zachowaniu zasobów hosta.
* **Pamięć RAM**: każdej maszynie przydzielono 8 GB pamięci operacyjnej.
* **Typ połączenia sieciowego**: zastosowano tryb mostkowany (bridged), umożliwiający pełną komunikację z innymi urządzeniami w sieci lokalnej, co było niezbędne dla testów typu Man-in-the-Middle czy sniffing.
* **Typ dysku**: dla systemu Windows wybrano nośnik NVMe, natomiast dla Ubuntu – SCSI, zgodnie z rekomendacjami instalatorów i wymaganiami kompatybilności.
* **Typ oprogramowania układowego**: wybrano UEFI ze względu na nowoczesne mechanizmy bezpieczeństwa oraz zgodność z funkcjami systemowymi, takimi jak Microsoft Defender Credential Guard.
* **VMware Tools**: zainstalowano na obu maszynach w celu poprawy wydajności i zapewnienia pełnej integracji ze środowiskiem wirtualnym.

Maszyny wirtualne zostały przygotowane w sposób umożliwiający swobodną realizację zaplanowanych scenariuszy testowych. Ich konfiguracja została dostosowana do wymagań badań nad bezpieczeństwem systemów operacyjnych, z uwzględnieniem typowych ustawień spotykanych w warunkach rzeczywistych.

2.4 Kali Linux – narzędzie do przeprowadzania testów

Na osobnym komputerze podłączonym do tej samej sieci zainstalowany został system Kali Linux w wersji 2025.1c. Jest to dystrybucja typu open-source oparta na systemie Debian, stworzona przez Offensive Security w celach przeprowadzania testów penetracyjnych oraz audytów bezpieczeństwa. Posiada on wbudowane narzędzia dedykowane testom bezpieczeństwa, a także dostosowane jądro systemu do wstrzykiwania różnego rodzaju pakietów [8].

System ten został wybrany do testów, ze względu na swoją popularność, stabilność oraz szeroką społeczność skupioną wokół oprogramowania. Jest to adekwatne narzędzie do przeprowadzenia testów bezpieczeństwa w środowiskach wirtualnych.

4. Testowanie bezpieczeństwa - ataki sieciowe

4.1 Sniffing

Celem testów było sprawdzenie czy możliwe jest pasywne podsłuchiwanie ruchu sieciowego (sniffing) z systemu Kali Linux zainstalowanego na fizycznym komputerze, przy użyciu narzędzia Wireshark. Wszystkie urządzenia były połączone do tej samej sieci lokalnej (Wi-Fi). Maszyny testowe – zarówno fizyczne, jak i wirtualne – pracowały w trybie mostkowanym (bridged), co zapewniało im bezpośrednią obecność w tej samej podsieci IP co komputer atakujący.

W ramach badania przeprowadzono cztery testy: dwa z systemem Windows 10 (na maszynie fizycznej i wirtualnej) oraz dwa z systemem Ubuntu 22.04 (również fizycznie i w VM). We wszystkich przypadkach wyłączono domyślne zabezpieczenia systemowe, takie jak zapora Windows Firewall czy ufw, aby sprawdzić, czy ruch HTTP da się przechwycić w warunkach minimalnej ochrony.

Tabela poniżej przedstawia zestawienie przeprowadzonych testów:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | System atakowany | Środowisko | Zabezpieczenia | Ruch HTTP przechwycony |
| WV | Windows 10 | VMware | brak | nie |
| WB | Windows 10 | Bare-metal | brak | nie |
| UV | Ubuntu 22.04 | VMware | brak | nie |
| UB | Ubunu 22.04 | Bare-metal | brak | nie |

Tab. X. Testy przeprowadzone w ramach ataku typu sniffing

Program Wireshark w wersji 4.4.4 został uruchomiony na interfejsie sieciowym wlan0[[1]](#footnote-1). Włączony został tryb promiscous.[[2]](#footnote-2) W celu zawężenia analizy, w Wiresharku ustawiono filtr ograniczający rejestrowane pakiety wyłącznie do protokołu HTTP. Test polegał na odwiedzeniu strony HTTP (http://testphp.vulnweb.com/login.php)[[3]](#footnote-3) na maszynie ofiary oraz zasymulowaniu ruchu poprzez wprowadzenie i wysłanie przykładowych danych logowania.

A screenshot of a computer

AI-generated content may be incorrect.  
Fot. Konfiguracja filtrowania ruchu HTTP w narzędziu Wireshark na interfejsie wlan0.

Po uruchomieniu przechwytywania i wygenerowaniu ruchu z maszyn ofiar Wireshark nie zarejestrował żadnego ruchu HTTP w żadnym z testowanych przypadków.

A screenshot of a computer

AI-generated content may be incorrect.  
Fot. X. Brak przechwyconego ruchu HTTP po stronie atakującego w przypadku ruchu z maszyny ofiary.

Dla porównania i zobrazowania ataku, a także sprawdzenia metodologii, przeprowadzony został dodatkowy test, gdzie połączenie http odbyło się na komputerze atakującym. Wireshark odnotował ruch sieciowy.

A screenshot of a computer

AI-generated content may be incorrect.  
Fot. Przechwycony ruch HTTP po wygenerowaniu go lokalnie na komputerze atakującym.

Dane przesyłane w formularzu logowania zostały przeanalizowane przy użyciu funkcji „Follow HTTP Stream”, która umożliwiła pełny podgląd przesłanych treści, w tym danych logowania.

A screenshot of a computer

AI-generated content may be incorrect.  
Fot. Podgląd danych logowania uzyskanych przez narzędzie „Follow HTTP Stream” w Wireshark na komputerze atrakującym.

****

**Rys. x.** Schemat topologii środowiska testowego w ataku sniffingu.

Przygotowany został schemat przedstawiający topologię środowiska testowego. W jego skład wchodzą: maszyny testowe (fizyczne oraz VM), router Wi-Fi (pełniący rolę przełącznika sieciowego) oraz komputer atakujący z Kali Linux i Wiresharkiem. Ruch sieciowy z maszyn ofiar nie był kierowany do atakującego, ponieważ przełącznik przekazuje pakiety wyłącznie do docelowego adresata, eliminując możliwość pasywnego podsłuchu.

Brak przechwyconego ruchu HTTP nie świadczy o błędach w konfiguracji testu, lecz o skuteczności współczesnej architektury sieci w ograniczaniu dostępu do pakietów przez nieuprawnione hosty. W kontekście bezpieczeństwa oznacza to, że nawet przy wyłączonych zaporach systemowych, izolacja sieciowa skutecznie chroni przed pasywnym sniffingiem. W środowiskach wykorzystujących przełączniki sieciowe (np. domowe routery Wi-Fi) standardowy sniffing z użyciem narzędzi takich jak Wireshark nie pozwala na podsłuchiwanie ruchu pomiędzy innymi urządzeniami w sieci. Wynika to z faktu, że przełączniki przekazują pakiety tylko do właściwego odbiorcy, co uniemożliwia pasywne przechwytywanie pakietów przez inne hosty [9].

Z przeprowadzonych badań wynika, że w przypadku testów sniffingu maszyny wirtualne nie wykazują różnicy w kontekście bezpieczeństwa w porównaniu do maszyn fizycznych. Ustawienie urządzeń w tej samej sieci (typu bridged) nie wystarczyło do przechwycenia ruchu http z maszyny ofiary.

Testy w wariantach z aktywnymi zabezpieczeniami nie zostały przeprowadzone – skoro przechwycenie ruchu było niemożliwe w najmniej chronionym scenariuszu, dalsze zwiększanie ochrony nie miałoby wpływu na wynik. Wnioskiem praktycznym jest to, że skuteczne przechwytywanie ruchu HTTP w takich warunkach wymaga zastosowania ataku aktywnego, np. typu ARP spoofing lub Man-in-the-Middle.

4.2 ARP spoofing

4.2.1 Teoretyczne podstawy ataku ARP spoofing

**ARP spoofing** to technika ataku w sieciach lokalnych, polegająca na wysyłaniu fałszywych komunikatów ARP (Address Resolution Protocol), które mają na celu przekonanie urządzenia ofiary, że atakujący jest innym zaufanym hostem — najczęściej bramą sieciową. Dzięki temu możliwe jest przechwycenie, zmodyfikowanie lub przekierowanie ruchu sieciowego [10].

Adresy IP służą do identyfikacji urządzeń w sieci, ale na poziomie fizycznym komunikacja odbywa się za pomocą adresów MAC. Aby powiązać adres IP z adresem MAC, urządzenia korzystają z ARP, które dynamicznie buduje tzw. tablicę ARP — mapującą IP do MAC. Atakujący może wykorzystać ten mechanizm, wysyłając spreparowaną odpowiedź ARP, która podszywa się pod router lub inne urządzenie. Celem jest wpisanie fałszywego powiązania IP z MAC w tablicy ARP ofiary [10].

A computer screen shot of a router

AI-generated content may be incorrect.

**Rys. x.** Schemat ataku ARP spoofing (MITM).

Atak przebiega zazwyczaj w dwóch etapach:

1. Atakujący wysyła fałszywe odpowiedzi ARP do ofiary, informując, że adres IP routera (bramy) należy do jego (atakującego) adresu MAC.
2. Ofiara aktualizuje swoją tablicę ARP i zaczyna przesyłać dane do atakującego zamiast do faktycznej bramy. Atakujący może przechwycić dane, a następnie przesłać je dalej (atak typu Man-in-the-Middle).

Ataki typu ARP spoofing są popularne nie tylko ze względu na swoją skuteczność, ale również z powodu prostoty implementacji. Wystarczy, że atakujący znajdzie się w tej samej sieci lokalnej co ofiara i będzie w stanie wysyłać pakiety ARP – nie są potrzebne żadne specjalne uprawnienia [11].

Z perspektywy bezpieczeństwa, ARP spoofing stanowi zagrożenie, ponieważ może prowadzić do:

* kradzieży tożsamości użytkownika (np. przejęcie sesji logowania),
* instalacji złośliwego oprogramowania (poprzez modyfikację ruchu),
* ataków typu DoS (gdy dane nie są przekazywane dalej po przejęciu).

Atak ARP spoofing pozwala nie tylko na pasywne podsłuchiwanie ruchu, ale także na jego modyfikowanie. Atakujący może przechwycić dane logowania, sesje HTTP, a nawet wstrzykiwać złośliwy kod [10].

4.2.2 Przebieg i wyniki ataku ARP spoofing w środowisku testowym

Przeprowadzone próby ataku ARP spoofing miały na celu przechwycenie danych logowania na testowanych systemach operacyjnych, z wykorzystaniem techniki **Man-in-the-Middle (MITM)**. Atak realizowany był za pomocą narzędzia **Ettercap**, a do analizy przechwyconego ruchu użyto dodatkowo **Wiresharka**.

**Rys. x.** Widok aplikacji Ettercap podczas ataku ARP spoofing. W polach *Target 1* oraz *Target 2* zdefiniowano adres IP ofiary oraz bramy, co umożliwia przeprowadzenie ataku typu Man-in-the-Middle. W dolnej części okna widoczne są przechwycone dane logowania przesyłane w postaci niezaszyfrowanej (HTTP POST).

**Rys. x.** Widok przechwyconego ruchu HTTP w narzędziu Wireshark. Widoczne jest żądanie POST do strony logowania /userinfo.php, zawierające dane uwierzytelniające przesyłane w formacie application/x-www-form-urlencoded. Przechwycenie pakietu potwierdza skuteczność ataku MITM.

Mechanizm ataku opierał się na fałszowaniu odpowiedzi ARP w lokalnej sieci, dzięki czemu maszyna atakująca (Kali Linux) była rozpoznawana przez system ofiary jako domyślna brama sieciowa. W wyniku tego cały ruch sieciowy ofiary był przekierowywany przez komputer atakującego. Pozwoliło to na przechwytywanie żądań HTTP, a w niektórych przypadkach – danych logowania przesyłanych w postaci nieszyfrowanej.

W trakcie każdego testu użytkownik na maszynie ofiary otwierał stronę <http://testphp.vulnweb.com/login.php> i ręcznie wpisywał dane logowania (np. admin:test). Przeprowadzano **po 15 prób dla każdego scenariusza** testowego, z rejestracją, czy dane zostały skutecznie przechwycone, czy tylko widoczny był ruch sieciowy, czy też nie odnotowano żadnej aktywności.

W przypadku Ettercapa, dane logowania (login i hasło) były często wyświetlane bezpośrednio w dolnym panelu aplikacji, w zakładce *"Messages"* lub *"Connections"* – jako fragment przesyłanego żądania HTTP POST. Narzędzie to potrafi również filtrować pakiety według treści i rozpoznawać typowe formularze logowania, dzięki czemu nawet bez szczegółowej analizy można było szybko ocenić, czy atak zakończył się sukcesem.

W Wiresharku natomiast dane logowania były identyfikowane jako treść żądania HTTP – widoczne w zakładce *"Follow HTTP Stream"*, najczęściej w formacie:

*username=admin&password=test*

lub jako fragment application/x-www-form-urlencoded w pakiecie typu POST. Widoczność tych danych była uzależniona od warunków testu – systemu, środowiska uruchomieniowego (VM/bare-metal) oraz aktywnych zabezpieczeń.

Scenariusze testowe obejmowały osiem konfiguracji: Windows 10 oraz Ubuntu 22.04, każdorazowo uruchomione jako maszyna wirtualna oraz jako system fizyczny (bare-metal), z włączonymi i wyłączonymi domyślnymi zaporami sieciowymi (firewall i ufw).

W ramach testów za **skuteczny atak** uznawano wyłącznie przypadki, w których dane logowania (login i hasło) zostały przechwycone w formie czytelnej. Samo odnotowanie aktywności sieciowej, bez możliwości pozyskania danych uwierzytelniających, traktowane było jako **niepowodzenie**, gdyż nie stanowi realnego zagrożenia dla poufności danych.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| System | Środowisko | Zabezpieczenia | Powodzenie | Przechwycenie samego ruchu | Brak wyników |
| Windows 10 | fizyczny komputer | włączone | 11 | 4 | 0 |
| Windows 10 | fizyczny komputer | wyłączone | 9 | 4 | 2 |
| Windows 10 | maszyna wirtualna | włączone | 11 | 4 | 0 |
| Windows 10 | maszyna wirtualna | wyłączone | 12 | 3 | 0 |
| Ubuntu 22.04 | fizyczny komputer | włączone | 9 | 6 | 0 |
| Ubuntu 22.04 | fizyczny komputer | wyłączone | 6 | 7 | 2 |
| Ubuntu 22.04 | maszyna wirtualna | włączone | 6 | 8 | 1 |
| Ubuntu 22.04 | maszyna wirtualna | wyłączone | 9 | 5 | 1 |

**Tab. x.** Wyniki testów ARP spoofing dla różnych konfiguracji systemów i środowisk.

Łącznie przeprowadzonych zostało 120 prób ataku ARP spoofing, z czego 73, zakończyło się powodzeniem, co daje ok. 61% skutecznych ataków. Dane pokazują, że zabezpieczenia typu firewall oraz ufw nie mają wpływu na ochronę przed atakami typu ARP spoofing, ponieważ w niektórych przypadkach udział skutecznych ataków przy włączonych zabezpieczeniach był większy niż wyłączonych ufw oraz firewall. Z uwagi na brak spójnych korelacji, szczegółowy podział skuteczności względem statusu zapory sieciowej nie został objęty osobną analizą porównawczą.

Wykres x. przedstawia procentową skuteczność przechwycenia danych logowania (pełny sukces ataku) w scenariuszach obejmujących systemy Windows 10 i Ubuntu 22.04, uruchamiane jako maszyny fizyczne oraz wirtualne.

**Rys. x.** Skuteczność ataków ARP spoofing (MITM) w zależności od systemu operacyjnego i środowiska uruchomieniowego.

Poniższe zestawienie pokazuje uśrednioną skuteczność przechwycenia danych logowania dla wszystkich testów przeprowadzonych na systemach uruchomionych bezpośrednio na sprzęcie oraz w środowisku wirtualnym.

**Rys. x.** Średnia skuteczność ataków ARP spoofing w zależności od środowiska uruchomieniowego (maszyna fizyczna vs wirtualna), niezależnie od systemu operacyjnego.

Z przedstawionych danych wynika, że system Windows był zauważalnie bardziej podatny na atak ARP spoofing zakończony przechwyceniem danych logowania niż Ubuntu. Większe bezpieczeństwo systemu Linux może wynikać np. z:

* Różnic w sposobie aktualizacji i przechowywania wpisów ARP (np. reakcja na różne typy pakietów ARP) [12],
* Różnic w domyślnych konfiguracjach systemowych wpływających na obsługę ARP (cache timeout, filtrowanie pakietów ARP) [12].
* Różnic w protokołach (np. POST, kodowanie URL).

Środowisko uruchomieniowe nie miało widocznego wpływy na wyniki. Różnice w skuteczności ataków między środowiskiem fizycznym a wirtualnym były niewielkie, co potwierdza dobrą zgodność odwzorowania warunków sieciowych przez tryb bridged w VMware.

Badanie pokazuje, że ARP spoofing umożliwia skuteczny atak typu MITM w sieciach lokalnych, niezależnie od tego czy system uruchomiony jest na fizycznym komputerze, czy maszynie wirtualnej. Większe znaczenie ma jednak system operacyjny.

Potencjalnymi środkami zaradczymi wobec podatności obu systemów na ataki typu ARP spoofing mogą być m.in.:

* dynamiczne monitorowanie tablicy ARP (np. Arpwatch) [13],
* Statyczne wpisy ARP w sieciach krytycznych [14],
* Segmentacja sieci (VLAN) [15].

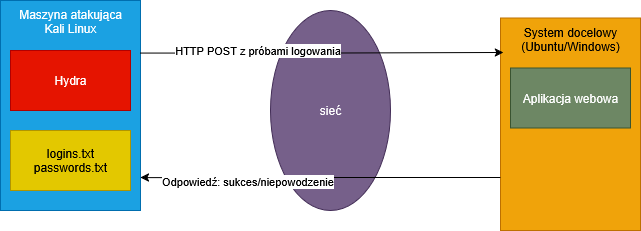
4.3 Brute-force

4.3.1 Charakterystyka ataku brute-force

Atak brute force polega na systematycznym testowaniu wszystkich możliwych kombinacji danych uwierzytelniających, aż do momentu znalezienia poprawnego loginu lub hasła. Podstawą skuteczności tego typu ataków jest automatyzacja – atakujący wykorzystuje odpowiednie narzędzia, takie jak skrypty lub specjalistyczne oprogramowanie (np. Hydra), które wykonują setki lub tysiące prób w krótkim czasie. Metoda ta może być skuteczna zwłaszcza w przypadkach, gdy użytkownicy stosują słabe, krótkie lub przewidywalne hasła. Współczesne ataki brute force są coraz częściej wspomagane przez dane statystyczne, takie jak zbiory haseł pochodzące z wycieków czy popularne kombinacje zebrane z otwartych źródeł. W takich scenariuszach atak może zostać zoptymalizowany poprzez użycie ataku słownikowego (dictionary attack), który skraca czas potrzebny do złamania hasła.

Czas potrzebny do skutecznego przeprowadzenia ataku rośnie wykładniczo wraz z długością i złożonością hasła. Dlatego silne mechanizmy uwierzytelniające – takie jak długie, losowe hasła, wieloskładnikowe uwierzytelnianie (MFA), limity logowań czy systemy CAPTCHA – znacząco podnoszą poziom odporności systemu na tego rodzaju zagrożenia [16].

Do ataków wykorzystano narzędzie Hydra. To jeden z najpopularniejszych programów służących do przeprowadzania ataków typu brute force na różne usługi sieciowe, w tym m.in. protokoły HTTP, FTP, SSH, Telnet, SMB czy POP3. Działa poprzez systematyczne próby logowania przy użyciu dostarczonych list loginów i haseł, aż do znalezienia poprawnej kombinacji.

**Rys. X.** Schemat ataku brute force z wykorzystaniem Hydry – wysyłanie żądań HTTP POST z próbami logowania do aplikacji webowej w systemie docelowym.

4.3.2 Analiza testów bezpieczeństwa systemów

Celem testów było sprawdzenie, czy systemy operacyjne (Windows 10 oraz Ubuntu 22.04), uruchomione zarówno jako maszyny wirtualne, jak i fizyczne, wykazują odporność na atak siłowy (brute-force) na aplikację webową z formularzem logowania. Test miał zweryfikować:

* czy systemy w jakikolwiek sposób reagują na dużą liczbę żądań logowania (np. logami, blokadą adresu IP, opóźnieniem odpowiedzi),
* czy wbudowane mechanizmy bezpieczeństwa (firewall/UFW) mają wpływ na skuteczność ataku,
* czy środowisko uruchomienia systemu (fizyczny komputer oraz maszyna wirtualna) zmienia jego podatność.

Test nie koncentrował się na łamaniu silnych haseł, lecz na **technicznej reakcji systemów operacyjnych na próbę ataku** — co istotne z perspektywy ich bezpieczeństwa w środowisku wirtualnym.

W celu przeprowadzenia testów ataków brute force przygotowano lekką aplikację webową z formularzem logowania, uruchamianą lokalnie w kontenerze Docker. Aplikacja została zaprojektowana w prosty sposób – umożliwia przesłanie loginu i hasła metodą POST oraz weryfikację poprawności danych na podstawie zapisanych wartości.

W skład rozwiązania wchodziły m.in. pliki login.php oraz users.php, zawierające odpowiednio logikę logowania oraz zestaw przykładowych danych uwierzytelniających. Środowisko zostało uruchomione lokalnie z wykorzystaniem docker-compose.

Pełna struktura aplikacji i jej kod źródłowy zostały przedstawione w załączniku (**Załącznik A**). Poniżej znajduje się przykładowy wygląd formularza logowania.

A screenshot of a computer

AI-generated content may be incorrect.

**Rys. x.** Interfejs aplikacji logowania, wykorzystywanej do testów brute force.

Aplikacja testowa wykorzystuje formularz logowania przesyłający dane metodą POST, w której login i hasło są przekazywane jako parametry żądania. Narzędzie Hydra automatyzuje ten proces, podstawiając kolejne kombinacje nazw użytkowników i haseł, a następnie analizuje odpowiedzi serwera pod kątem wystąpienia określonego komunikatu błędu, co pozwala na identyfikację poprawnych danych logowania.

Przykładowa komenda użyta w testach:

hydra -L logins.txt -P passwords.txt -s 8080 [adres\_IP] http-post-form "/login.php:username=^USER^&password=^PASS^:Nieprawidłowe dane"

**Parametry:**

* -L, -P – pliki z loginami i hasłami.
* -s – port serwera HTTP (tu: 8080).
* http-post-form – moduł ataku na formularze POST.
* "/login.php:...:Nieprawidłowe dane" – format:
  + ścieżka do formularza (/login.php),
  + parametry POST z loginem i hasłem (^USER^, ^PASS^),
  + tekst rozpoznający błąd logowania (Nieprawidłowe dane).

Na rysunku poniżej przedstawiono przykład skutecznego ataku brute force wykonanego z użyciem narzędzia Hydra. Kombinacja loginu admin oraz hasła password została poprawnie odgadnięta i zidentyfikowana przez narzędzie jako prawidłowa.



**Rys. X.** Przykład udanego ataku brute force z użyciem Hydra (znaleziono poprawną parę login/hasło).

Za **sukces** uznano poprawne dopasowanie loginu i hasła, potwierdzone odpowiedzią serwera inną niż komunikat o błędzie logowania.

W trakcie testów wykorzystano:

* **10 loginów** oraz **1000 haseł**, co daje 10 000 prób,
* Listy zostały utworzone w oparciu o najpopularniejsze kombinacje (np. admin, user, 123456, password, itp.),
* Ich rozmiar dobrano tak, aby atak trwał krótko, lecz był wystarczająco realistyczny do celów badawczych.

Tabela przedstawia zestawienie rezultatów testów ataków typu brute force w różnych konfiguracjach systemów operacyjnych oraz środowisk uruchomieniowych. We wszystkich przypadkach uzyskano poprawne uwierzytelnienie, co potwierdza skuteczność ataku niezależnie od zastosowanego systemu, rodzaju środowiska (maszyna wirtualna lub fizyczna) oraz stanu zapory sieciowej.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| System operacyjny | Środowisko | Zabezpieczenia | Rezultat | Czas | Szybkość (tries/min) |
| Windows 10 | fizyczny komputer | włączone | powodzenie | ok. 2 minuty | 3795 |
| Windows 10 | fizyczny komputer | wyłączone | powodzenie | ok. 2 minuty | 3194 |
| Windows 10 | maszyna wirtualna | włączone | powodzenie | ok. 2 minuty | 3680 |
| Windows 10 | maszyna wirtualna | wyłączone | powodzenie | ok. 2 minuty | 3795 |
| Ubuntu 22.04 | fizyczny komputer | włączone | powodzenie | ok. 2 minuty | 3770 |
| Ubuntu 22.04 | fizyczny komputer | wyłączone | powodzenie | ok. 2 minuty | 3195 |
| Ubuntu 22.04 | maszyna wirtualna | włączone | powodzenie | ok. 2 minuty | 3750 |
| Ubuntu 22.04 | maszyna wirtualna | wyłączone | powodzenie | ok. 2 minuty | 3683 |

Tab. X. Wyniki testów ataku brute force w zależności od środowiska i konfiguracji zabezpieczeń.

Wszystkie testy zakończyły się powodzeniem, rozumianym jako poprawne odgadnięcie pary login–hasło na potrzeby logowania do testowej aplikacji webowej. Czasy trwania ataku oraz szybkość (tr/min) pozostają w większości przypadków bardzo zbliżone – różnice rzędu kilkudziesięciu prób/min są naturalne i wynikają z losowych czynników środowiskowych (obciążenie CPU, opóźnienia sieciowe itp.). Wszystkie scenariusze zakończyły się **pełnym powodzeniem** – login i hasło zostały odgadnięte, bez względu na konfigurację. Żaden z systemów nie odnotował również prób włamania się do aplikacji (brak informacji w logach systemu oraz brak np. powiadomień systemowych).

**Obecność zapory sieciowej** (zarówno Windows Firewall, jak i UFW w Ubuntu) **nie wpłynęła istotnie** na skuteczność ataku brute force. Jedyną zauważalną różnicą był nieznaczny spadek wydajności w niektórych konfiguracjach z aktywną zaporą, co może wynikać z opóźnień w przetwarzaniu pakietów.

**System operacyjny nie miał istotnego wpływu** na rezultat testów – zarówno Windows 10, jak i Ubuntu 22.04 pozwoliły na skuteczne przeprowadzenie ataku przy wykorzystaniu identycznych parametrów i list loginów/haseł.

Nie odnotowano również różnic między testami przeprowadzonymi na systemach postawionych na maszynach wirtualnych oraz fizycznym komputerze. W rezultacie można stwierdzić, że testowana aplikacja pozbawiona jakichkolwiek mechanizmów obronnych (np. limitów logowań, CAPTCHA, opóźnień czasowych) jest całkowicie podatna na ataki słownikowe, niezależnie od konfiguracji systemu, środowiska czy zapory sieciowej.

Analiza przeprowadzonych testów wykazała, że ataki typu brute force były skuteczne we wszystkich badanych konfiguracjach systemowych — niezależnie od używanego systemu operacyjnego (Windows 10 lub Ubuntu 22.04), typu środowiska (maszyna fizyczna lub wirtualna), jak również stanu zapory sieciowej (aktywna lub nieaktywna). W każdym przypadku narzędzie Hydra zdołało uzyskać poprawne dane uwierzytelniające w czasie nieprzekraczającym kilku minut.

Powyższe wyniki wskazują, że domyślna konfiguracja systemów operacyjnych, szczególnie w kontekście ochrony hostowanych aplikacji webowych, nie zawiera skutecznych mechanizmów przeciwdziałających atakom słownikowym. Brak reakcji systemu na liczne nieudane próby logowania oznacza, że zabezpieczenia tego typu muszą być wdrażane świadomie przez administratora systemu.

W kontekście systemowym, rekomendowane są następujące środki zaradcze:

* **Konfiguracja limitów połączeń i filtrowanie ruchu (np. za pomocą iptables, fail2ban, Windows Defender Firewall z zasadami niestandardowymi)** – umożliwia blokowanie adresów IP generujących dużą liczbę prób logowania.
* **Aktywne monitorowanie logów systemowych (np. auth.log, Windows Event Viewer)** – pozwala na wykrycie anomalii i wczesne reagowanie na potencjalne zagrożenia.
* **Integracja z systemami IDS/IPS (np. Snort, Suricata)** – może pozwolić na identyfikację charakterystycznych wzorców ataków brute force.
* **Stosowanie uwierzytelniania dwuskładnikowego na poziomie systemu operacyjnego lub aplikacji (jeśli wspierane)** – istotnie podnosi poziom bezpieczeństwa, nawet przy skutecznym odgadnięciu loginu i hasła.

Reasumując, choć testowana aplikacja webowa była celowo uproszczona i pozbawiona zabezpieczeń, kluczowym wnioskiem z badań jest brak systemowych mechanizmów obrony przed zautomatyzowanym atakiem brute force. Wdrożenie odpowiednich zabezpieczeń po stronie systemu operacyjnego ma zatem istotne znaczenie w ochronie usług dostępnych w sieci.

**4.4 Eskalacja uprawnień**

4.4.1 Eskalacja uprawnień – teoria

Eskalacja uprawnień stanowi jedną z najistotniejszych kategorii zagrożeń w kontekście bezpieczeństwa systemów operacyjnych. Polega na nieuprawnionym uzyskaniu dostępu do poziomu uprawnień wyższego niż przypisany użytkownikowi – na przykład przejęciu uprawnień administratora (root) przez zwykłego użytkownika. Z perspektywy analizy systemów w środowiskach wirtualnych, eskalacja uprawnień jest szczególnie istotna, ponieważ umożliwia pełne przejęcie kontroli nad maszyną wirtualną, a w niektórych przypadkach – również nad systemem hosta [17].

Wyróżnia się dwa podstawowe typy ataków eskalacyjnych:

* **Eskalacja pionowa (vertical privilege escalation)** – polega na uzyskaniu przez użytkownika wyższych uprawnień, niż zostały mu nadane, np. przejęcie roli administratora przez użytkownika o uprawnieniach ograniczonych [18].
* **Eskalacja pozioma (horizontal privilege escalation)** – polega na uzyskaniu dostępu do zasobów innego użytkownika posiadającego ten sam poziom uprawnień, np. dostęp do konta współużytkownika [18].

W systemach Linux i Windows występuje szereg technik eskalacyjnych, z których najczęściej wykorzystywane obejmują:

* błędne konfiguracje uprawnień sudo [18];
* pliki wykonywalne z ustawionym bitem SUID;
* zadania cron oraz skrypty uruchamiane z uprawnieniami administratora;
* znane podatności jądra systemu operacyjnego (np. Dirty COW, CVE-2016-5195);
* niezabezpieczone skrypty startowe oraz pliki z nadanymi niepoprawnymi uprawnieniami [18].

W celu automatycznego rozpoznawania podatności eskalacyjnych stosuje się m.in.:

* linpeas – skaner bezpieczeństwa analizujący konfigurację systemu;
* GTFOBins – baza operacji możliwych do wykorzystania w atakach eskalacyjnych;
* Linux Exploit Suggester – narzędzie rekomendujące znane exploity jądra;
* sudo -l – manualne narzędzie do przeglądu dostępnych komend z uprawnieniami administratora [18].

W środowiskach wirtualnych (VM) eskalacja uprawnień może prowadzić do eskalacji między maszynami lub nawet do przejęcia kontroli nad systemem hosta.

W celu ograniczenia ryzyka eskalacji uprawnień zaleca się stosowanie następujących praktyk:

* wykorzystanie narzędzi auditujących, np. PriviLynis;
* stosowanie mechanizmów Mandatory Access Control (MAC), takich jak SELinux czy AppArmor;
* regularna aktualizacja systemu operacyjnego i bibliotek;
* stosowanie zasady minimalnych uprawnień (PoLP);
* implementacja nowoczesnych technik obronnych, takich jak Kernel Data Relocation Mechanism (KDRM), które utrudniają wykorzystanie exploitów jądra [19].

4.4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nr** | **Technika / Wektor Ataku** | **Opis testu** | **Kompatybilność systemowa** | **Wymagane narzędzia / warunki** | **Oczekiwany efekt** |
| 1 | sudo -l – błędna konfiguracja sudo | Sprawdzenie, czy użytkownik może wykonać komendę jako root bez hasła | Ubuntu VM / bare metal | Dostęp do lokalnego konta użytkownika | Uzyskanie shella jako root |
| 2 | Pliki SUID | Wyszukiwanie plików z bitem SUID i analiza ich wykorzystania | Ubuntu VM / bare metal | find, GTFOBins | Uzyskanie podniesionych uprawnień |
| 3 | cron / zadania cykliczne | Zbadanie zadań cyklicznych wykonywanych z uprawnieniami root | Ubuntu VM / bare metal | crontab, dostęp do logów | Wstrzyknięcie polecenia z uprawnieniami root |
| 4 | Niezabezpieczone skrypty startowe | Sprawdzenie, czy istnieją skrypty uruchamiane z uprawnieniami root, które można modyfikować | Ubuntu VM / bare metal | ls, find, ps, grep | Przejęcie kontroli nad wykonywanym kodem |
| 5 | linPEAS automatyczny skan | Wykrywanie wektorów przy pomocy linPEAS.sh | Ubuntu VM / bare metal | linPEAS | Raport podatności i możliwości |
| 6 | Abuse service permissions (Windows) | Sprawdzenie, czy użytkownik może edytować konfigurację usługi systemowej | Windows VM / bare metal | accesschk, sc.exe, powershell | Uruchomienie własnego kodu jako SYSTEM |
| 7 | DLL Hijacking | Analiza załadowanych DLL i podstawienie złośliwej biblioteki | Windows VM / bare metal | procmon, procmon-log, własna DLL | Wykonanie kodu z uprawnieniami SYSTEM |
| 8 | WinPEAS automatyczny skan | Skanowanie systemu Windows w poszukiwaniu wektorów | Windows VM / bare metal | WinPEAS | Raport potencjalnych możliwości |
| 9 | CVE exploit (np. Dirty COW / PrintNightmare) | Wykorzystanie znanej podatności jądra lub systemu | Ubuntu / Windows VM / bare metal | Exploit code (lokalny) | Uzyskanie uprawnień root / SYSTEM |

Tab. X

Test 1 – Eskalacja uprawnień przez plik /bin/bash z ustawionym bitem SUID

Celem testu jest sprawdzenie, czy niepoprawne nadanie bitu SUID plikowi /bin/bash umożliwi użytkownikowi o ograniczonych uprawnieniach uzyskanie dostępu do powłoki z uprawnieniami administratora (root). Scenariusz symuluje błąd konfiguracyjny, który może wystąpić w środowiskach produkcyjnych wskutek nieprawidłowej administracji lub działania złośliwego oprogramowania. **Założenia testowe:**

* System: Ubuntu 22.04 LTS
* Użytkownik testowy: testuser1 (zwykły użytkownik)
* SUID ustawiono ręcznie: sudo chmod u+s /bin/bash
* Metoda ataku: uruchomienie /bin/bash -p przez użytkownika nieuprzywilejowanego

**Przebieg testu**

1. **Zalogowano się na konto root**, nadano plikowi /bin/bash bit SUID:

sudo chmod u+s /bin/bash

1. **Zalogowano się na konto testuser1**, uruchomiono nową powłokę z opcją -p (zachowanie efektywnego UID):

/bin/bash -p

1. **Sprawdzono efektywne ID użytkownika**:

id

# uid=1001(testuser1) gid=1001(testuser1) euid=0(root) groups=1001(testuser1),100(users)

Obserwacja polecenia id potwierdziła skuteczną eskalację uprawnień – użytkownik testuser1 uzyskał efektywne uprawnienia roota (euid=0). Wynik był identyczny na maszynie wirtualnej i fizycznej.

| **Parametr** | **Ubuntu VM** | **Ubuntu Bare Metal** |
| --- | --- | --- |
| System operacyjny | Ubuntu 22.04 LTS | Ubuntu 22.04 LTS |
| Użytkownik testowy | testuser1 | testuser1 |
| Błąd konfiguracyjny | SUID na /bin/bash | SUID na /bin/bash |
| Polecenie atakujące | /bin/bash -p | /bin/bash -p |
| Efektywny UID po ataku | 0 (root) | 0 (root) |
| Wynik | Eskalacja uprawnień powiodła się | Eskalacja uprawnień powiodła się |

Tabela 4.2.1 – Wyniki testu 1: SUID + /bin/bash

Test wykazał, że niepoprawna konfiguracja pliku systemowego poprzez nadanie bitu SUID prowadzi do **natychmiastowej i pełnej eskalacji uprawnień**. Co istotne, wynik był niezależny od środowiska uruchomieniowego – zarówno maszyna wirtualna, jak i fizyczna zareagowały identycznie.

To potwierdza, że **środowisko wirtualne w tym przypadku nie wpływa na podatność** na ten typ ataku. Kluczowym czynnikiem jest błędna konfiguracja pliku wykonywalnego i brak kontroli nad nadanymi uprawnieniami.

**Zalecenia:**

* Niedopuszczalne jest ręczne nadawanie bitu SUID plikom takim jak /bin/bash.
* Rekomenduje się regularne audyty uprawnień plików z użyciem narzędzi takich jak find / -perm -4000 -type f.
* W środowiskach produkcyjnych powinno się monitorować zmiany uprawnień systemowych plików binarnych.

Test nr 2 – Eskalacja uprawnień przez błędną konfigurację sudo

Celem testu było zbadanie wpływu błędnej konfiguracji uprawnień w pliku /etc/sudoers na możliwość eskalacji uprawnień użytkownika lokalnego. Przetestowano scenariusz, w którym użytkownik ma możliwość uruchomienia programu nano jako root, bez konieczności podawania hasła.

**Założenia**

* Test został przeprowadzony na dwóch środowiskach z systemem Ubuntu: maszynie wirtualnej (VM) oraz systemie zainstalowanym bezpośrednio na fizycznym komputerze (bare metal).
* Utworzono użytkownika testuser1 z ograniczonymi uprawnieniami.
* W pliku konfiguracyjnym /etc/sudoers dodano następujący wpis:

testuser1 ALL=(ALL) NOPASSWD: /usr/bin/nano

Choć wpis ten został świadomie dodany na potrzeby testu, ma on **realne odniesienie do sytuacji występujących w rzeczywistych środowiskach produkcyjnych**. Administratorzy nierzadko udzielają użytkownikom dostępu do konkretnych programów (np. edytorów tekstowych) bez hasła, nie będąc świadomymi, że niektóre z tych aplikacji — takie jak nano, vim, czy less — umożliwiają wywołanie powłoki systemowej. Taki dostęp może prowadzić do pełnej eskalacji uprawnień. Test stanowi zatem **symulację błędnej, ale spotykanej konfiguracji bezpieczeństwa**.

**Przebieg testu:**

1. Zalogowano się do systemu jako użytkownik testuser1.
2. Użytkownik wywołał polecenie:

sudo /usr/bin/nano /etc/shadow

Dzięki wpisowi NOPASSWD, polecenie zostało wykonane bez żądania hasła.

1. Edytor nano został otwarty z uprawnieniami roota, umożliwiając podgląd oraz edycję wrażliwego pliku systemowego.
2. Następnie za pomocą skrótu Ctrl+R, a potem Ctrl+X, uruchomiono tryb poleceń, w którym wpisano:

reset; bash

W efekcie użytkownik otrzymał **pełną powłokę systemową z uprawnieniami roota**.

| **Parametr** | **Ubuntu VM** | **Ubuntu bare metal** |
| --- | --- | --- |
| Możliwość eskalacji uprawnień | Tak | Tak |
| Konieczność podania hasła | Nie | Nie |
| Dostęp do plików systemowych | Tak (np. /etc/shadow) | Tak (np. /etc/shadow) |

W obu środowiskach testowych (VM i bare metal) błędna konfiguracja uprawnień w pliku sudoers doprowadziła do skutecznej eskalacji uprawnień. Test potwierdził, że nawet pozornie bezpieczne i ograniczone polecenia (takie jak nano) mogą zostać wykorzystane do uzyskania pełnego dostępu do systemu.

**Zalecenia:**

* Należy unikać wpisów NOPASSWD dla programów umożliwiających interakcję z systemem, takich jak nano, vim, less, python, bash itd.
* Każdy wpis w pliku sudoers powinien być dodawany z użyciem visudo, który oferuje weryfikację składni i bezpieczeństwa.
* W razie potrzeby umożliwienia dostępu do pojedynczych poleceń – zaleca się korzystanie z wrapperów (np. script.sh z ograniczoną funkcjonalnością) oraz dodatkowych mechanizmów kontroli (SELinux, AppArmor).

Test nr 3 – Próba eskalacji uprawnień poprzez dostęp do /dev/mem

Celem testu było sprawdzenie, czy użytkownik o ograniczonych uprawnieniach może uzyskać dostęp do urządzenia /dev/mem, które odzwierciedla fizyczną pamięć systemu. Taki dostęp może potencjalnie umożliwić bezpośrednią manipulację danymi w pamięci, w tym również przejęcie kontroli nad procesami systemowymi lub eskalację do uprawnień administratora.

**Założenia:**

* Test przeprowadzono na dwóch platformach: systemie Ubuntu 22.04 działającym jako maszyna wirtualna (VM) oraz na identycznym systemie zainstalowanym na sprzęcie fizycznym (bare metal).
* Użytkownik testowy (testuser1 lub testuser2) został dodany do grupy kmem, która w teorii umożliwia dostęp do /dev/mem.
* Jądro systemu w obu przypadkach miało aktywną ochronę w postaci opcji CONFIG\_STRICT\_DEVMEM=y.

**Przebieg testu:**

1. Użytkownik sprawdził uprawnienia do /dev/mem:

ls -l /dev/mem

1. Następnie podjął próbę odczytu zawartości pamięci:

hexdump -C /dev/mem | head

1. Spróbował również zapisać dane do /dev/mem:

echo "AAAA" | dd of=/dev/mem bs=1 seek=0 count=4

1. We wszystkich przypadkach operacje zostały zablokowane z powodu braku odpowiednich uprawnień lub aktywnej ochrony jądra.

| **Parametr** | **Ubuntu VM** | **Ubuntu bare metal** |
| --- | --- | --- |
| Użytkownik w grupie kmem | Tak | Tak |
| Odczyt z /dev/mem możliwy | Nie (permission denied) | Nie (operation not permitted) |
| Zapis do /dev/mem możliwy | Nie | Nie |
| CONFIG\_STRICT\_DEVMEM aktywne | Tak | Tak |

Test wykazał, że dostęp do /dev/mem został skutecznie zablokowany w obu środowiskach, pomimo przypisania użytkownika do grupy kmem. Decydującym czynnikiem okazały się restrykcje jądra systemowego (CONFIG\_STRICT\_DEVMEM=y), które uniemożliwiają odczyt i zapis fizycznej pamięci przez procesy nieuprzywilejowane.

Należy jednak zauważyć, że potencjalne różnice między środowiskiem wirtualnym a fizycznym mogłyby się pojawić w przypadku odmiennych implementacji urządzeń /dev/mem przez hypervisor. W systemie bare metal /dev/mem odwzorowuje fizyczny adres przestrzeni RAM, podczas gdy w środowisku wirtualnym dostęp ten może być dodatkowo filtrowany lub w ogóle emulowany przez warstwę wirtualizacji (np. KVM, VMware). Pomimo braku różnic w wynikach testu, architekturalnie możliwe są rozbieżności w zachowaniu przy innym jądrze lub wersji hypervisora.

Bibliografia

[1] Virtualization Throughout the Software Lifecycle, Sarah N. Crutchfield

[2] Virtualization and Security Aspects: An Overview, Rui Filipe Pereira, Rui Miguel Silva & João Pedro Orvalho

[3] Virtualization and Forensics A Digital Forensic Investigator’s Guide to Virtual Environments

[4] VMware vSphere Essentials: A Practical Approach to vSphere Deployment and Management Luciano Patrão

[5] Optimal guest file system for type-2 hypervisorbased virtualization in Virtual box

[6] <https://uefi.org/specs/PI/1.8/V2_Overview.html>

[7] UEFI Memory Forensics: A Framework for UEFI Threat Analysis Kalanit Suzan Segal∗

[8] <https://www.kali.org/docs/introduction/what-is-kali-linux/>

[9] Patel, N. P., Patel, R. G., & Patel, D. R. (2009). Packet Sniffing: Network Wiretapping. W: 2009 IEEE International Advance Computing Conference (IACC 2009) (s. 2691-2696). Patiala, Indie: IEEE.

[10] Daniel G. Graham, *Ethical Hacking*, No Starch Press, 2021, rozdział 2.

[11] Alina Alina; Shipra Saraswat, *Understanding, Implementing and Combating Sniffing and ARP Spoofing*. 4th Int. Conf. on Recent Developments in Control, Automation & Power Engineering (RDCAPE), 2021.

[12] Sengupta, D. (2013, October 19). *ARP spoofing attack and detection using Wireshark*. In *Packet Analysis of Network Traffic*.

[13] Assegie, T. A., & Nair, P. S. (2019). *Comparative study on methods used in prevention and detection against Address Resolution Protocol spoofing attack*. Journal of Theoretical and Applied Information Technology.

[14] ELKILANI, Wail, AMIN, Khalid M. An automated approach for preventing ARP spoofing attack using static ARP entries. *International Journal of Advanced Computer Science and Applications*. 2014.

[15] BULL, Ronny L., MATTHEWS, Jeanna N. i TRUMBULL, Kaitlin A. VLAN hopping, ARP poisoning and Man-In-The-Middle attacks in virtualized environments. [Referat konferencyjny]. Sierpień 2016

[16] K. Swathi, “Brute Force Attack on Real World Passwords,” *International Journal of Research Publication and Reviews*, Nov. 2022.

[17] Happe, L., & Cito, J. (2024). *Got Root? A Linux Priv-Esc Benchmark*. arXiv preprint arXiv:2405.02106.

[18] Safe Security. (2021). *A Hands-on Approach to Linux Privilege Escalation*. Retrieved from <https://safe.security/wp-content/uploads/a-hands-on-approach-to-linux-privilege-escalation.pdf>

[19] SpringerLink. (2024). *Mitigation of privilege escalation attack using kernel data relocation mechanism (KDRM)*. Journal of Computer Security.

# Załącznik A. Kod aplikacji testowej do ataków brute force

## A.1 login.php

## Plik login.php realizuje podstawową walidację danych przesyłanych przez formularz. Sprawdzenie poprawności odbywa się na podstawie zdefiniowanych na sztywno danych uwierzytelniających.

## <?php

## $valid\_user = "admin";

## $valid\_pass = "password";

## if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

## $username = $\_POST["username"] ?? '';

## $password = $\_POST["password"] ?? '';

## if ($username === $valid\_user && $password === $valid\_pass) {

## echo "Zalogowano pomyślnie";

## } else {

## echo "Nieprawidłowe dane logowania";

## }

## }

## ?>

## A.2 login.html

## Plik login.html odpowiada za interfejs użytkownika.

## <!DOCTYPE html>

## <html lang="pl">

## <head>

## <meta charset="UTF-8">

## <title>Test logowania</title>

## </head>

## <body>

## <h2>Logowanie testowe</h2>

## <form action="login.php" method="post">

## Login: <input type="text" name="username"><br><br>

## Hasło: <input type="password" name="password"><br><br>

## <input type="submit" value="Zaloguj">

## </form>

## </body>

## </html>

## A.3 docker-compose.yml

Plik docker-compose.yml umożliwia łatwe i szybkie uruchomienie aplikacji lokalnie lub w środowisku testowym.

## version: '3.1'

## services:

## bruteforce:

## build: .

## ports:

## - "8080:80"

## container\_name: bruteforce\_test

## A.4 Dokckerfile

W celu uruchomienia serwera aplikacji wykorzystywany jest obraz PHP z Apache. Dockerfile definiuje środowisko uruchomieniowe.

FROM php:8.1-apache

COPY . /var/www/html/

EXPOSE 80

- podrozdział dlaczego win I linux i dlaczego w takich wersjach

- ‘wykorzystanie istniejących narzędzi, wyodrębnienie obszarów bezpieczeństwa, przeprowadzenie testów, podsumowanie i wnioski’

- Uzasadnienie czemu te a nie inne narzędzia

- Mocne wyodrębnienie obszarów bezpieczeństwa

- Kompleksowa ocena bezpieczeństwa systemów wirtualnych

- Które systemy na jaki typ ataków

Ubuntu: vmtestuser, password

Vmtest, password

student, password

Kali: kaliuser, password

Opis problemu, narzędzi, wprowadzenie ogólnikowe, co będzie robione – w jaki sposób, testy wydajnościowe, po wpr – krotki wstep teoretyczny

Najpierw narzędzia, przegląd literatury, istniejące rozwiązania, motywacje pracy,

Graficzna prezentacja platrofa testowa

20 sierpnia ALFA

Promotor uwagi:

~~- instrukcja obsługi – przerobić – wyrzucić zrzuty. Podstawowe informacje o maszynie i podzespołach – procesor, ram, grafika, dysk – na czym to było~~

- krótkie wprowadzenie

- układ: wprowadzenie (2str max – opis problemu, narzędzia istniejące, skoncentrowałem się na konkretnych elementach bezpieczeństwa tego systemu, będę testował w taki sposób; coś na styl streszczenia krótkiego) – co będzie robione, w jaki sposób, jakie będą efekty – czego się spodziewam w jakiej formie

- wstęp teoretyczny: teoria nt środowiska wirtualnego, bezpieczeństwa, jakich pojęć będę używał, opis narzędzi/języków (taki a taki system, czym się charakteryzuje)

- przegląd literatury, istniejące rozwiązania, MOTYWACJE PRACY

- teza i cel pracy (zakładamy tezę: np. bezpieczeństwo systemów operacyjnych jest bardzo wymagane i trudne do osiągnięcia, więc trzeba takie systemy zabezpieczyć; zakładam, że można za pomocą takiego a takiego narzędzia zaprezentować metody w jaki sposób zabezpieczyć swoje systemy; za pomocą testów opiszę na czym idea bezpieczeństwa polega) – max. pół strony (osobno teza i cel)

Cel: realizacja własnoręcznie wirtualnej maszyny z systemem takim i przeprowadzenie testów, żeby przedstawić w jaki sposób to bezpieczeństwo działa

Bezpieczenstwo sys oper jest trudje do osiagnieca w srod wirt. - teza

Realizacja wlasnorecznej maszyny z systemtem takim a takim – cel

- Czesc badawcza: jak najwięcej testów; też osobny rozdział stricte tylko testów na maszynie bez porównania też; dużo wyników i wniosków

- typy ataków teoretycznie – opisać! Zacząć od tego w części badawczej. Zrobić abstrakt graficzny – graficznie zaprezentować platformę testową (komputer z systemem rzeczywistym, drugi z VM – SCHEMAT (jednym z modułów było testowanie); nie trzeba się bawić w UML – wystarczy na kształtach – dużo kolorów – DUŻO SCHEMATÓW

Opisać na początku części badawczej z jakich elementów/modułów składa się platforma testowa – ogólny zarys żeby ktoś widział stanowisko oraz w jaki sposób wyniki były uzyskiwane

Też fajnie np. blokowo zaprezentować na czym atak polegał

- robić też testy porównawcze z zabezpieczeniami i bez nich

- opisywać wykresy

- wnioski: który system bezpieczniejszy, która opcja bezpieczniejsza – VM czy bare-metal

- zapora sieciowa na sieciach publicznych i prywatnych

- oprócz sieciowych jakiś malware

- każdy rozdział poświęcony innemu typowi testów

1. Interfejs wlan0 w systemach Linux odnosi się do połączenia bezprzewodowego i może być używany do analizy ruchu sieciowego, jeśli działa w trybie monitorowania. [↑](#footnote-ref-1)
2. Promiscius to tryb pracy interfejsu sieciowego, w którym przechwytywane są wszystkie pakiety przechodzące przez sieć. [↑](#footnote-ref-2)
3. Podana strona umożliwia przeprowadzanie testów bezpieczeństwa, poprzez przechwytywanie danych logowania przesływanych przez http. [↑](#footnote-ref-3)