**Akademia Górniczo-Hutnicza  
im. Stanisława Staszica w Krakowie  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Wydział Inżynierii Metali i Informatyki Przemysłowej**



**PRACA MAGISTERSKA**

**Temat: Bezpieczeństwo systemów operacyjnych w środowiskach wirtualnych  
- promotor Przemysław Marynowski**

Imię i nazwisko: Michał Kochanowicz

Kierunek studiów: Informatyka Techniczna

Opiekun pracy: dr inż. Przemysław Marynowski

Kraków, rok Akadem. 2024/2025

Spis treści

Streszczenie 3

1. Wstęp 4

2. Wprowadzenie teoretyczne 6

3. Środowisko testowe 11

4. Testy bezpieczeństwa 15

5. Zakończenie i wnioski 48

Literatura 51

Załącznik A. Kod aplikacji testowej do ataków brute-force 53

# Streszczenie

Praca polegała na przetestowaniu pod kątem bezpieczeństwa popularnych systemów operacyjnych (Windows 10 oraz Ubuntu 22.04 LTS), uruchomionych w środowisku wirtualnym. Analiza zakładała, tam gdzie to możliwe, porównanie bezpieczeństwa systemów wirtualnych z natywnymi, w możliwie zbliżonych konfiguracjach, a także zbadanie czy wirtualizacja generuje nowe podatności na ataki. Motywacją było stworzenie kompleksowego porównania, które pozwoli na ocenę wpływu wirtualizacji na poziom bezpieczeństwa oraz identyfikację różnic wynikających z samej specyfiki systemów operacyjnych.

Wyszczególnione zostały główne części pracy, czyli teoretyczna, w której przedstawiona została akademicka wiedza na temat poruszanych zagadnień, a także część badawcza, która opierała się na przeprowadzeniu testów bezpieczeństwa. Testy opierały się na atakach sieciowych (*sniffing, ARP spoofing, brute-force)*, eskalacji uprawnień oraz sprawdzeniu podatności kanałów integracyjnych między systemem gospodarza, a gościa (wspólny schowek, współdzielone foldery).

Głównym narzędziem wykorzystanym w badaniach jest VMware Workstation Pro, czyli platforma umożliwiająca tworzenie maszyn wirtualnych, udostępniająca jednocześnie zaawansowaną konfigurację. Do ataków sieciowych posłużyły narzędzia wbudowane w system Kali Linux (zainstalowany na osobnym komputerze), m.in. Wireshark. Eskalacja uprawnień odbywała się głównie wewnątrz badanych systemów, ale w różnych konfiguracjach przydzielonych uprawnień.

Spodziewanym rezultatem było potwierdzenie tezy, że bezpieczeństwo systemów operacyjnych w maszynach wirtualnych jest zasadniczo porównywalne z systemami natywnymi, natomiast różni się profil ryzyka. Środowisko wirtualne nie powinno mieć istotnego wpływu na ataki sieciowe oraz podatność na eskalację uprawnień. Różniący się profil ryzyka może wynikać ze specyfiki wirtualizacji, czyli większe bezpieczeństwo pod kątem dostępu do urządzeń USB/PCI, natomiast nowe podatności wynikające chociażby ze współdzielenia przestrzeni maszyny wirtualnej z systemem natywnym.

# Wstęp

Współczesne systemy informatyczne wymagają elastyczności, wysokiej wydajności oraz racjonalizacji kosztów eksploatacji i utrzymania. Istotną odpowiedzią na te potrzeby jest **wirtualizacja**, która oddziela warstwę usług od fizycznej infrastruktury i umożliwia jednoczesne uruchamianie wielu systemów operacyjnych oraz usług na wspólnym zasobie sprzętowym. Takie podejście zwiększa wykorzystanie zasobów, upraszcza zarządzanie i obniża koszty, przy zachowaniu wymaganych parametrów jakościowych [21]. Takie podejście generuje jednak nowe potencjalne zagrożenia, dotyczące bezpieczeństwa systemów, wykorzystujących techniki wirtualizacji.

## Cel, zakres pracy i teza

Celem pracy jest ocena bezpieczeństwa systemów operacyjnych uruchamianych w maszynach wirtualnych w porównaniu z maszynami fizycznymi. Podstawą jest założenie, że migracja do środowiska wirtualnego co do zasady nie usuwa istniejących podatności, lecz zmienia profil ryzyka i może wprowadzać dodatkowe wektory ataku lub korzyści (np. lepszą izolację) [20].

Zakres obejmuje zwięzły przegląd literatury i zagrożeń, opis użytej platformy oraz doboru systemów (Windows 10, Ubuntu LTS), opracowanie jednolitej metodyki porównawczej i wykonanie eksperymentów w parach „maszyna wirtualna–komputer fizyczny” (tam, gdzie to jest możliwe) dla kluczowych obszarów, czyli warstwy sieciowej (m.in. podsłuchiwanie ruchu, podszywanie się w sieci lokalnej, atak słownikowy) oraz eskalacji uprawnień (m.in. pliki SUID, konfiguracja *sudo*, dostęp do urządzeń peryferyjnych), a także kanałów typowych dla platformy VMware. Wyniki zostaną zestawione i przeanalizowane ze sformułowaniem wniosków oraz wskazaniem potencjalnych przyczyn uzyskanych rezultatów. Można zatem wyszczególnić część teoretyczną, badawczą oraz wnioski.

W pracy przyjęto następującą tezę: **bezpieczeństwo systemów uruchamianych w maszynach wirtualnych i na maszynach fizycznych jest porównywalne, natomiast różni się profil ryzyka.** Wirtualizacja z jednej strony może ograniczać skutki incydentów, z drugiej tworzy nowe wektory ataku [20].

## Motywacje podjęcia pracy

Podjęcie pracy na temat bezpieczeństwa systemów operacyjnych w środowiskach wirtualnych wynika z potrzeby zbadania jak, coraz bardziej popularne, maszyny wirtualne zachowują się w warunkach zagrożenia. Polskojęzyczna, a także zagraniczna literatura nie oferuje wielu pozycji, które kompleksowo opisywałyby to zagadnienie, szczególnie w oparciu o różne typy ataków, a także o porównanie najpopularniejszych systemów obecnie na rynku, czyli Windows i Linux, porównując również ich zachowanie do sytuacji, gdy są zainstalowane na fizycznym sprzęcie.

# Wprowadzenie teoretyczne

Wirtualizacja polega na uruchamianiu jednego lub więcej systemów gościa na sprzęcie wirtualnym, zarządzanym przez hipernadzorcę, który rozdziela zasoby i zapewnia izolację oraz mechanizmy współdzielenia [20]. W tym modelu dotychczasowe podatności usług i aplikacji pozostają aktualne, natomiast dochodzą wektory specyficzne, czyli m.in. ryzyka w warstwie zarządzania i dostępu administracyjnego do hipernadzorcy, kanały integracji hosta z gościem, a także utrata widoczności ruchu na wewnętrznych sieciach wirtualnych, co może wymagać alternatywnych metod monitorowania [20].

## Przegląd literatury i istniejących rozwiązań

Jak podkreślają Scarfone, Souppaya i Hoffman w przewodniku NIST *Guide to Security for Full Virtualization Technologies*, wirtualizacja nie usuwa zagrożeń właściwych systemom gości, lecz wprowadza ryzyka warstwy pośredniej (hipernadzorca), kanałów host–gość i sieci wirtualnych. Ramy odniesienia obejmują mechanizmy integracji narzędzi gościa czy też zarządzanie obrazami i migawkami, ze szczególnym uwzględnieniem poufności danych w migawkach i konieczności monitorowania ruchu wewnątrz hosta [20].

W ujęciu Pék, Buttyána i Bencsátha przedstawionym w artykule *A Survey of Security Issues in Hardware Virtualization* model zagrożeń dla środowisk wirtualnych grupuje wektory względem celu (maszyna wirtualna, hipernadzorca, interfejsy zarządzania, sieć, pamięć, w kontekście magazynowania danych) i przywilejów napastnika (zewnętrzny bez dostępu, uprzywilejowany w gościu i infrastrukturze oraz pełny dostęp administracyjny). Wektory te zostały przedstawione na Rys. 2.1. Autorzy podkreślają znaczenie technik wykrywania wirtualizacji jako kroków poprzedzających ataki specyficzne dla platformy [21].

A screenshot of a computer screen

AI-generated content may be incorrect.  
Rys. 2.1. Model zagrożeń w środowiskach wirtualnych wg Pék, Buttyána i Bencsátha – wektory ataku według celu i poziomu przywilejów napastnika.

Jak wskazują Mahipal i Ceronmani Sharmila w pracy *Virtual Machine Security Problems and Countermeasures for Improving Quality of Service in Cloud Computing*, ujęcia problemowe koncentrują się na trzech kategoriach trudnych zagrożeń: boczne kanały wynikające ze współdzielonego sprzętu, kompromitacja hipernadzorcy (np. błędy mapowania pamięci) oraz nadużycia migracji na żywo. Autorzy postulują podejście holistyczne łączące środki na poziomie maszyny wirtualnej i hipernadzorcy, takie jak np. monitorowanie liczników wydajności, analiza zaufanych wywołań systemowych czy samoodtwarzanie maszyny wirtualnej (migawki) [23].

Were, Singha i Odawę w *Security Evaluation Framework for Virtualised* proponują wielowarstwowe ramy oceny i ochrony środowisk wirtualnych dla kluczowych wektorów ryzyka (podatności hipernadzorcy, ucieczki z maszyn wirtualnych, ataki między maszynami wirtualnymi) w celu zwiększenia wykrywalności, skrócenia czasu reakcji i ograniczenia narzutu na wydajność. Ramy obejmują warstwy hipernadzorcy, maszyn wirtualnych, sieci, zarządzania oraz monitorowania. Akcentują „mikrohipernadzorcę”, wzmocnioną izolację, segmentację sieci, wieloskładnikowe uwierzytelnianie i ciągłe monitorowanie [24].

## Wskazanie wybranych obszarów bezpieczeństwa

Analizy i eksperymenty koncentrują się na trzech głównych obszarach, które obejmują ryzyka wspólne dla fizycznych maszyn oraz maszyn wirtualnych, ale także wektory specyficzne dla wirtualizacji. Do obszarów tych należą m.in. **warstwa sieciowa**, a w niej ataki takie jak podsłuch (ang, *sniffing)*, podszywanie się pod ARP (ang. *ARP spoofing*), *brute-force*, a także **eskalacja uprawnień**, a w niej eskalacja przez plik z bitem SUID, eskalacja przez błędną konfigurację sudo, eskalacja poprzez dostęp do ścieżki */dev/mem,* atak typu *Rowhammer*, a także dostęp do urządzeń peryferyjnych. Ostatnim obszarem są **kanały hosta oraz gościa w VMware**, czyli wspólny dostęp do schowka i foldery współdzielone, co jest podatnością typową dla maszyn wirtualnych, której nie da się bezpośrednio porównać z systemami zainstalowanymi na fizycznym komputerze.

## Sposób przeprowadzenia testów

Badania realizowane są w kontrolowanym środowisku składającym się z hosta, dwóch wariantów uruchomieniowych systemów badanych (na maszynie fizycznej oraz w maszynie wirtualnej) i odrębnego węzła generującego ruch lub bezpośrednio atak. Tam, gdzie było to możliwe, realizowano scenariusz dla pary składającej się z systemu zainstalowanego na fizycznym sprzęcie oraz systemu uruchomionego przy pomocy maszyny wirtualnej, a także na systemie Windows 10 oraz, dla porówniania, Ubuntu 22.04. Zachowane zostały identyczne lub zbliżone ustawienia oprogramowania.

## 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, wykonująca program tak, jakby była fizycznym urządzeniem, więc można byłoby powiedzieć, że jest to „komputer”, działający wewnątrz fizycznego komputera. Aplikacja maszyny wirtualnej (nazywana „gościem”) uruchamia swój własny system operacyjny na rzeczywistej maszynie (zwanej „gospodarzem”). Wirtualny system operacyjny może być dowolny, np. Windows, MacOS lub Linux i nie jest ograniczony do jednego systemu operacyjnego na maszynie gospodarza [3]. Każda maszyna wirtualna działa niezależnie i nie ma wpływu na działanie innych maszyn wirtualnych.

## Charakterystyka wirtualizacji

Wirtualizacja to technologia umożliwiająca tworzenie wielu odizolowanych środowisk komputerowych, zwanych maszynami wirtualnymi, na jednym fizycznym urządzeniu. Dzięki warstwie pośredniczącej, zwanej hipernadzorcą (ang. ***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 to oprogramowanie, które umożliwia tworzenie i zarządzanie maszynami wirtualnymi poprzez oddzielenie systemów operacyjnych gości od fizycznej infrastruktury sprzętowej. W zależności od sposobu działania, wyróżniamy jego dwa główne typy. Pierwszym z nich jest **hipernadzorca natywny** (ang. *bare-metal*), który działa bezpośrednio na sprzęcie, bez potrzeby instalowania systemu operacyjnego gospodarza. Przykładem jest Microsoft Hyper-V. Drugim typem jest **hipernadzorca hostowany**, który funkcjonuje jako aplikacja zainstalowana w ramach istniejącego systemu operacyjnego, chociażby popularne VirtualBox, czy VMware Workstation.

Dodatkowo, hipernadzorcę można klasyfikować ze względu na sposób wirtualizacji. Wyróżnia się:

* **pełną wirtualizację**, gdzie system gościa działa bez konieczności modyfikacji, nieświadomy, że funkcjonuje w środowisku wirtualnym, jest w pełni niezależny;
* **wirtualizację wspieraną sprzętowo**, gdzie wykorzystuje specjalne funkcje procesora, takie jak Intel VT-x czy AMD-V, w celu optymalizacji pracy maszyn wirtualnych;
* **parawirtualizację**, która wymaga modyfikacji systemu gościa, który jest „świadomy”, że działa w środowisku wirtualnym i potrafi efektywnie współpracować z hipernadzorcą [5].

## Przewidywane efekty

Przewidywanym wynikiem analiz jest wniosek, że przeniesienie systemów do środowisk wirtualnych co do zasady nie zmienia podatności usług i aplikacji, lecz modyfikuje profil ryzyka, dodając wektory specyficzne (warstwa zarządzania, integracje między gospodarzem a gościem, sieci) przy jednoczesnych korzyściach izolacyjnych. Poziom bezpieczeństwa maszyn fizycznych i wirtualnych powinien być porównywalny, a różnice wynikać głównie z konfiguracji oprogramowania i specyfiki maszyn wirtualnych.

# Środowisko testowe

Rozdział przedstawia wybrane do testów narzędzia, ich konfigurację, a także parametry techniczne sprzętu, na którym działają omawiane oprogramowania. Wskazane zostały także motywacje użycia takich, a nie innych rozwiązań.

## 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 odpowiednim narzędziem do prowadzenia badań nad bezpieczeństwem w środowiskach wirtualnych.

A screenshot of a computer

AI-generated content may be incorrect.

Rys. 3.1. Widok podstawowy Vmware Workstation w wersji 17.6.3.

Środowisko Vmware Workstation oferuje szereg funkcji ułatwiających pracę z maszynami wirtualnymi. Umożliwia tworzenie maszyn z wybraną konfiguracją systemu operacyjnego, pamięci RAM oraz przestrzeni dyskowej. Pozwala na wykonywanie migawek (ang. *snapshot*), czyli zapisywanie stanu maszyny w wybranym momencie i przywracanie go w razie potrzeby. Dodatkowo oferuje klonowanie maszyn, uruchamianie wielu maszyn jednocześnie przy zachowaniu izolacji między nimi, a także zaawansowaną konfigurację sieci wirtualnych (np. NAT, bridge, host-only). Program umożliwia również łączenie się z zdalnymi serwerami, takimi jak Vmware ESX.

## 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), natomiast drugą z systemem Ubuntu 24.04.2 LTS. Obie maszyny skonfigurowano w taki sposób, że zostały przypisane po cztery rdzenie procesora, aby zapewnić stabilność i wydajność przy jednoczesnych zachowaniu zasobów gospodarza. Przydzielono im po 8 GB pamięci RAM. Połączenie sieciowe ustawiono w trybie mostkowanym (*bridged*), umożliwiając pełną komunikację z innymi urządzeniami w sieci lokalnej, co było niezbędne do testów typu *Man-in-the-Middle* i *sniffing*. Dla systemu Windows wybrano dysk NVMe, natomiast dla Ubuntu – SCSI, zgodnie z zaleceniami instalatorów i wymaganiami kompatybilności. Oprogramowanie układowe ustawiono jako UEFI, co zapewnia nowoczesne mechanizmy bezpieczeństwa i zgodność z funkcjami systemowymi, takimi jak Microsoft Defender Credential Guard. Na obu maszynach zainstalowano VMware Tools, poprawiające wydajność i integrację ze środowiskiem wirtualnym. Konfiguracja maszyn pozwalała na swobodne przeprowadzanie zaplanowanych scenariuszy testowych i odzwierciedlała typowe ustawienia spotykane w środowiskach rzeczywistych.

## 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.

# Testy bezpieczeństwa

Rozdział stanowi część badawczą pracy, z jednoczesnym uwzględnieniem teoretycznych wyjaśnień pojęć związanych z przeprowadzanymi badaniami. Analiza obejmuje identyfikację potencjalnych zagrożeń, symulację wybranych ataków oraz ocenę skuteczności mechanizmów ochronnych w kontrolowanych warunkach laboratoryjnych. Przeprowadzane testy pozwalają na porównanie podatności systemów w różnych konfiguracjach oraz określenie wpływu wirtualizacji na bezpieczeństwo operacyjne.

## Podsłuchiwanie pakietów (ang. *sniffing*)

Podsłuchiwanie pakietów (ang. *sniffing*) to pasywne przechwytywanie i analiza ruchu sieciowego przez program działający na interfejsie sieciowym, co umożliwia odczyt treści niezabezpieczonych pakietów (np. HTTP, Telnet) i pozyskanie danych uwierzytelniających. Jest stosowany do diagnostyki i testów, natomiast z perspektywy bezpieczeństwa jest formą nieuprawnionej ingerencji. Zbliżony jest do „podsłuchu przewodu”, a w sieciach przełączanych jego skuteczność wobec obcego ruchu zwykle wymaga technik pośrednich (np. MITM/ARP) zamiast czystej obserwacji łącza [22].

Celem testów było sprawdzenie czy możliwe jest pasywne podsłuchiwanie ruchu sieciowego 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 na maszynie wirtualnej). We wszystkich przypadkach wyłączono domyślne zabezpieczenia systemowe, takie jak zapora Windows Firewall czy ufw (w systemie Ubuntu), 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 | Fizyczne | brak | nie |
| UV | Ubuntu 22.04 | VMware | brak | nie |
| UB | Ubunu 22.04 | Fizyczne | brak | nie |

Tab. 4.1. 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. 4.1. 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. 4.2. 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. 4.3. 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. 4.4. Podgląd danych logowania uzyskanych przez narzędzie „*Follow HTTP Stream*” w Wireshark na komputerze atrakującym.

**A screenshot of a computer

AI-generated content may be incorrect.**

Rys. 4.1. Schemat topologii środowiska testowego w ataku typu *sniffing*.

Przygotowany został schemat przedstawiający topologię środowiska testowego. W jego skład wchodzą: maszyny testowe (fizyczne oraz maszyny wirtualne), 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 podsłuchiwaniem. 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 podsłuchiwania, 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*.

## *ARP spoofing*

### Teoretyczne podstawy ataku *ARP spoofing*

*ARP spoofing* to technika ataku w sieciach lokalnych, polegająca na wysyłaniu fałszywych komunikatów ARP (ang. *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. 4.2. Schemat ataku *ARP spoofing*.

Atak przebiega zazwyczaj w dwóch etapach. W pierwszym atakujący wysyła fałszywe odpowiedzi ARP do ofiary, informując, że adres IP routera (bramy) należy do jego (atakującego) adresu MAC. Następnie 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 ze względu na swoją skuteczność, jak 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 m.in. do kradzieży tożsamości użytkownika (np. przejęcie sesji logowania), instalacji złośliwego oprogramowania (poprzez modyfikację ruchu) lub też 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].

### 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.

Fot. 4.5. 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).

Fot. 4.6. 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, czyli systemu, środowiska uruchomieniowego (komputer fizyczny czy maszyna wirtualna) oraz aktywnych zabezpieczeń.

Scenariusze testowe obejmowały osiem konfiguracji: Windows 10 oraz Ubuntu 22.04, każdorazowo uruchomione jako maszyna wirtualna oraz jako system fizyczny, 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. 4.2. 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 4.1. 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.

Wykres. 4.1. Skuteczność ataków *ARP spoofing (MITM)* w zależności od systemu operacyjnego i środowiska uruchomieniowego.

Poniższe zestawienie (Wykres 4.2.) pokazuje uśrednioną skuteczność przechwycenia danych logowania dla wszystkich testów przeprowadzonych na systemach uruchomionych bezpośrednio na sprzęcie oraz w środowisku wirtualnym.

Wykres 4.2. Średnia skuteczność ataków *ARP spoofing* w zależności od środowiska uruchomieniowego (maszyna fizyczna i 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), ale także różnic w domyślnych konfiguracjach systemowych wpływających na obsługę ARP (*cache timeout*, filtrowanie pakietów ARP) [12]. Może to być też kwestia 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 według wyników ma jednak system operacyjny, natomiast należy mieć na uwadze, że na wyniki mogły wpływać również czynniki środowiskowe, takie jak m.in. zachowanie domowego *AP/switcha* (*ARP cache*, *proxy* *ARP*), charakterystyka Wi‑Fi, obciążenie sieci/CPU, tempo lub wzorzec zatruwania generowany przez narzędzie.

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], czy segmentacja sieci (VLAN) [15].

## Atak siłowy (ang. *brute-force*)

### Charakterystyka ataku siłowego

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 (ang. *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.

A purple oval with white text

AI-generated content may be incorrect.Rys. 4.3. Schemat ataku brute force z wykorzystaniem Hydry – wysyłanie żądań HTTP POST z próbami logowania do aplikacji webowej w systemie docelowym.

### Analiza i testy bezpieczeństwa systemów na atak *brute-force*

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), a także czy wbudowane mechanizmy bezpieczeństwa (*firewall/ufw*) mają wpływ na skuteczność ataku. Sprawdzone zostało także jedno z podstawowych założeń pracy, mianowicie 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.

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 A. Poniżej znajduje się przykładowy wygląd formularza logowania.

A screenshot of a computer

AI-generated content may be incorrect.

Fot. 4.7. 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* oraz *-P* oznaczają pliki z loginami i hasłami, -*s* to port serwera HTTP (tu: 8080), *http-post-form* oznacza moduł ataku na formularze POST. "*/login.php:...:Nieprawidłowe dane*" to format ścieżki do formularza *(/login.php*), gdzie znajduje się parametr POST z loginem i hasłem *(^USER^, ^PASS^)* oraztekst informujący o błędzie logowania (*Nieprawidłowe dane*).

Na fotografii 4.8. 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.



Fot. 4.8. 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 | Szybkość (próby/min) |
| Windows 10 | fizyczny komputer | włączone | powodzenie | 3795 |
| Windows 10 | fizyczny komputer | wyłączone | powodzenie | 3194 |
| Windows 10 | maszyna wirtualna | włączone | powodzenie | 3680 |
| Windows 10 | maszyna wirtualna | wyłączone | powodzenie | 3795 |
| Ubuntu 22.04 | fizyczny komputer | włączone | powodzenie | 3770 |
| Ubuntu 22.04 | fizyczny komputer | wyłączone | powodzenie | 3195 |
| Ubuntu 22.04 | maszyna wirtualna | włączone | powodzenie | 3750 |
| Ubuntu 22.04 | maszyna wirtualna | wyłączone | powodzenie | 3683 |

Tab. 4.3. 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 (próby*/min*) pozostają w większości przypadków bardzo zbliżone i wynosiły po około 2 minuty. Różnice rzędu kilkudziesięciu prób na minutę 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. 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 siłowym. Brak reakcji systemu na liczne nieudane próby logowania oznacza, że zabezpieczenia tego typu muszą być wdrażane świadomie przez administratora systemu.

Aby zaradzić takiemu stanowi rzeczy można byłoby np. wprowadzić konfigurację limitów połączeń i filtrowanie ruchu. Umożliwiłoby to blokowanie adresów IP generujących dużą liczbę prób logowania. Innym sposobem może być po prostu także stosowanie uwierzytelniania dwuskładnikowego na poziomie systemu operacyjnego lub aplikacji, co istotnie podnosi poziom bezpieczeństwa, nawet przy skutecznym odgadnięciu loginu i hasła, jest to proste i jednocześnie trudne do złamania zabezpieczenie, które jednak negatywnie wpływa na wygodę użytkowania systemu lub aplikacji.

Mimo, że 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.

## Eskalacja uprawnień

### Eskalacja uprawnień – wprowadzenie teoretyczne

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. Przykładowo, na 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. Pierwszy typ to eskalacja pionowa (ang,. *vertical privilege escalation*), która polega na uzyskaniu przez użytkownika wyższych uprawnień, niż zostały mu nadane, np. przejęcie roli administratora przez użytkownika o uprawnieniach ograniczonych. Drugim typem jest eskalacja pozioma (ang. *horizontal privilege escalation*), polegająca 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 m.in. błędne konfiguracje uprawnień *sudo* [18], wykorzystanie plików wykonywalnych z ustawionym bitem *SUID*, czy też wykorzystanie znanych podatności jądra systemu operacyjnego (np. *Dirty COW*, *CVE-2016-5195*).

Eskalacja może też być przeprowadzona przez niezabezpieczone skrypty startowe oraz pliki z nadanymi niepoprawnymi uprawnieniami [18].

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

Aby ograniczyć ryzyko eskalacji uprawnień można chociażby stosować mechanizmy takie jak *Mandatory Access Control* (MAC), stosować zasady minimalnych uprawnień oraz nie zapominać o regularnych aktualizacjach systemów operacyjnych i bibliotek. Skuteczne może też być opisane przez Hiroki Kuzono oraz Toshihiro Yamauchi zastosowanie nowoczesnych technik obronnych, takich jak np. *Kernel Data Relocation Mechanism (KDRM),* które ograniczają możliwość przeprowadzania ataków na jądro systemu [19].

### Eskalacja uprawnień przez plik z ustawionym bitem SUID[[4]](#footnote-4)

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. Test nie może zostać przeprowadzony na systemie Windows, a więc podatność ta została przetestowana jedynie na Ubuntu 22.04.

Przez to, że wirtualizacja wprowadza dodatkową warstwę pośredniczącą między systemem operacyjnym a sprzętem, warto sprawdzić czy hipernadzorca nie modyfikuje działania takich mechanizmów i nie wprowadza różnic w skuteczności ataku.

Utworzono użytkownika testowego o nazwie testuser1, który posiada jedynie podstawowe uprawnienia. SUID zostało ustawione ręcznie poprzez komendę sudo chmod u+s /bin/bash. Sam atak polega na uruchomieniu /bin/bash -p przez nieuprzywilejowanego użytkownika, gdzie -p oznacza *privileged mode*, czyli zachowanie uprawnień administratora.

Test przebiegał w taki sposób, że początkowo zalogowano się na konto root, nadano plikowi /bin/bash bit SUID:

sudo chmod u+s /bin/bash

Następnie alogowano się na konto testuser1, uruchomiono nową powłokę z opcją -p: /bin/bash -p

Na końcu sprawdzono ID użytkownika komendą id. Wynik w konsoli był następujący:

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 – maszyna wirtualna** | **Ubuntu – fizyczny system** |
| 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 | Powodzenie | Powodzenie |

Tab. 4.4. – Wyniki testu 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. 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.

Aby uniknąć takich podatności niedopuszczalne jest ręczne nadawanie bitu SUID plikom takim jak /bin/bash, a także w środowiskach produkcyjnych powinno się monitorować zmiany uprawnień systemowych plików binarnych.

### Eskalacja uprawnień przez błędną konfigurację *sudo[[5]](#footnote-5)*

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[[6]](#footnote-6)* jako administrator, bez konieczności podawania hasła.

Test został przeprowadzony na dwóch środowiskach z systemem Ubuntu, czyli maszynie wirtualnej oraz systemie zainstalowanym bezpośrednio na fizycznym komputerze. Wykorzystano użytkownika *testuser1* z ograniczonymi uprawnieniami. W pliku konfiguracyjnym /etc/sudoers dodano następujący wpis:

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

Pomimo, że wpis ten został 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* czy *vim*, umożliwiają wywołanie powłoki systemowej. Taki dostęp może prowadzić do pełnej eskalacji uprawnień. Test stanowi zatem symulację błędnej, ale spotykanej konfiguracji bezpieczeństwa.

Test przebiegał w taki sposób, że użytkownik z ograniczonymi uprawnieniami (*testuser1*) wywołał polecenie:

sudo /usr/bin/nano /etc/shadow

Dzięki wpisowi NOPASSWD, polecenie zostało wykonane bez żądania hasła. Edytor *nano* został otwarty z uprawnieniami administratora, umożliwiając podgląd oraz edycję wrażliwego pliku systemowego. Następnie za pomocą skrótu *Ctrl+R,* a następnie *Ctrl+X*, uruchomiono tryb poleceń, w którym wpisano:

reset; bash

W efekcie użytkownik otrzymał pełną powłokę systemową z uprawnieniami administratora.

|  |  |  |
| --- | --- | --- |
| **Parametr** | **Maszyna wirtualna** | **Fizyczny system** |
| Możliwość eskalacji uprawnień | Tak | Tak |
| Konieczność podania hasła | Nie | Nie |
| Dostęp do plików systemowych | Tak (np. /etc/shadow) | Tak (np. /etc/shadow) |

Tab. 4.5. Wyniki testu eskalacji uprawnień przez błędną konfigurację *sudo.*

W obu środowiskach testowych 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.

Należy zatem unikać wpisów NOPASSWD dla programów umożliwiających interakcję z systemem, takich jak *nano, vim* czy *bash*. W razie potrzeby umożliwienia dostępu do pojedynczych poleceń można korzystać z *wrapperów[[7]](#footnote-7)* (np. script.sh z ograniczoną funkcjonalnością) oraz dodatkowych mechanizmów kontroli (*SELinux, AppArmor*).

### 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.

Test przeprowadzono na dwóch platformach, czyli systemie Ubuntu 22.04 działającym jako maszyna wirtualna oraz na identycznym systemie zainstalowanym na sprzęcie fizycznym. Użytkownik testowy (*testuser1*) 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.

Test przebiegał w taki sposób, że użytkownik sprawdził uprawnienia do /dev/mem, poprzez komendę:

ls -l /dev/mem

Następnie podjął próbę odczytu zawartości pamięci:

hexdump -C /dev/mem | head

Spróbował również zapisać dane do /dev/mem:

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

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

|  |  |  |
| --- | --- | --- |
| **Parametr** | **Maszyna wirtualna** | **Fizyczny system** |
| 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 |

Tab. 4.6. Wyniki testu eskalacji uprawnień przez dostęp do /dev/mem.

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.

Potencjalne różnice między środowiskiem wirtualnym a fizycznym mogłyby się pojawić w przypadku odmiennych implementacji urządzeń /dev/mem przez hipernadzorcę. W systemie zainstalowanym na maszynie fizycznej /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. Pomimo braku różnic w wynikach testu, architekturalnie możliwe są rozbieżności w zachowaniu przy innym jądrze lub wersji hipernadzorcy.

### Eskalacja uprawnień poprzez podatność sprzętową *Rowhammer*

Celem testu było sprawdzenie, czy technika ataku *Rowhammer*, polegająca na fizycznym zakłóceniu działania komórek pamięci RAM, może zostać skutecznie przeprowadzona w środowisku maszyny wirtualnej oraz na systemie uruchomionym bezpośrednio na sprzęcie. Porównanie miało na celu ocenę potencjalnych różnic w podatności między tymi środowiskami.

Test został wykonany z wykorzystaniem narzędzia *Rowhammerjs* udostępnionego przez *IAIK* (Graz University of Technology). Test został przeprowadzony jedynie na systemie Linux. Użytkownik testowy miał dostęp do kompilacji i uruchomienia aplikacji C++ z poziomu administratora.

Badanie przeprowadzono w taki sposób, że sklonowano repozytorium i przeprowadzono kompilację:

git clone https://github.com/IAIK/rowhammerjs.git

cd rowhammerjs/native

make

Następnie uruchomiono test z uprawnieniami administratora:

sudo ./rowhammer

Test automatycznie rozpoczął „uderzanie” w komórki pamięci RAM. Po ok. 10 minutach test został ręcznie przerwany przez użytkownika.

|  |  |  |
| --- | --- | --- |
| **Parametr** | **Maszyna wirtualna** | **Fizyczny system** |
| Czy test się uruchomił | Tak | Tak |
| Czy trwał nieprzerwanie | Tak (do ręcznego zatrzymania) | Tak (do ręcznego zatrzymania) |
| Czy wykryto problemy (np. błędy bitów) | Nie stwierdzono (brak alertów) | Nie stwierdzono (brak alertów) |
| Czy pojawiły się komunikaty o trudnościach z dostępem do stron pamięci | Tak (częściowe ostrzeżenia) | Tak (częściowe ostrzeżenia) |

Tab. 4.7. Wyniki testu eskalacji uprawnień poprzez podatność Rowhammer.

Test wykazał, że zarówno środowisko wirtualne, jak i fizyczne umożliwiają uruchomienie narzędzia *Rowhammer*, jednak nie odnotowano faktycznych błędów bitowych podczas trwania testu. Jest to zgodne z oczekiwaniami, ponieważ większość nowoczesnych systemów wykorzystuje mechanizmy ochrony (np. *ECC*, zwiększoną izolację pamięci), które zmniejszają podatność na ataki typu *Rowhammer*.

Różnice między środowiskami mogą jednak występować w bardziej ekstremalnych warunkach, np. przy wykorzystaniu starszego sprzętu lub pamięci bez *ECC*. W środowiskach wirtualnych dodatkową warstwę zabezpieczenia stanowi hipernadzorca, który pośredniczy w dostępie do fizycznej pamięci RAM. Oznacza to, że środowisko natywne jest potencjalnie bardziej podatne na atak *Rowhammer* niż środowisko wirtualne.

Fakt, że test nie zakończył się samoczynnie, wynika z jego charakteru, ponieważ narzędzie celowo próbuje wywołać niestabilność pamięci przez długotrwałe operacje. W przypadku braku wykrycia podatności, test może trwać nawet godzinami.

### Dostęp do urządzeń *PCI/USB*

Test miał na celu porównanie możliwości dostępu do urządzeń *PCI* oraz *USB* przez użytkownika bez uprawnień administracyjnych w systemach Ubuntu oraz Windows, uruchomionych na systemie natywnym oraz w środowisku wirtualnym. Założeniem było sprawdzenie, czy środowisko wirtualne wprowadza istotne ograniczenia w zakresie widoczności i interakcji ze sprzętem, które mogłyby zmniejszyć ryzyko eskalacji uprawnień poprzez analizę lub manipulację urządzeniami fizycznymi.

Testy przeprowadzono na kontach użytkowników nieposiadających uprawnień administratora. W środowisku Ubuntu wykorzystano narzędzia *lsusb*, *lspci*, odczyty z /dev/bus/usb oraz /sys/bus/pci w celu sprawdzenia możliwości dostępu do danych sprzętowych. W systemie Windows zastosowano polecenia *PowerShell* (Get-PnpDevice, Get-WmiObject) pozwalające na odczyt danych o urządzeniach *PCI* i *USB*.

A screen shot of a computer

AI-generated content may be incorrect.  
Fot. 4.9. Wyniki polecenia *lsusb* w systemie Ubuntu w środowisku wirtualnym VMware.

A computer screen shot of a computer

AI-generated content may be incorrect.  
Fot. 4.10. Wyniki polecenia *lsusb* w systemie Ubuntu, uruchomionym na maszynie fizycznej.

Wyniki pokazały, że w systemie Ubuntu uruchomionym na fizycznym sprzęcie użytkownik nieuprzywilejowany ma pełen dostęp do listy urządzeń *PCI* i *USB* (fot. 4.10) oraz może bezpośrednio odczytywać ich konfigurację i dane za pomocą przestrzeni /sys oraz /dev. W środowisku wirtualnym Ubuntu (fot. 4.9) dostęp ten jest znacząco ograniczony. Użytkownik widzi tylko urządzenia przypisane przez hipernadzorcę, a odczyt danych z urządzeń fizycznych jest często niemożliwy.

W systemie Windows, działającym natywnie, również uzyskano pełną widoczność fizycznych urządzeń i możliwość odczytu szczegółowych informacji, takich jak identyfikatory producenta (*VID*, *PID*), typy urządzeń czy przypisane porty. W maszynie wirtualnej Windows zakres ten ogranicza się jedynie do urządzeń emulowanych przez VMware. Urządzenia fizyczne nieprzekazane hipernadzorcę są niewidoczne.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **System** | **Maszyna** | **PCI - widoczne urządzenia** | **USB - odczyt danych** | **Dostęp do /sys lub /dev** | **Uwagi** |
| Ubuntu | Natywna | wszystkie fizyczne | tak | tak | Pełny dostęp |
| Ubuntu | Wirtualna | tylko przypisane przez VM | zależne od *passthrough* | ograniczony | Widoczność zależna od hipernadzorcy |
| Windows | Natywna | wszystkie fizyczne | tak | brak /sys | Dostęp przez *PowerShell* |
| Windows | Wirtualna | tylko emulowane | tylko wirtualne | brak /sys | Brak dostępu do fizycznych urządzeń |

Tab. 4.8. Wyniki testu eskalacji uprawnień poprzez dostęp do urządzeń PCI i USB

Wyniki wskazują, że środowisko natywne zapewnia większy poziom dostępu do sprzętu z poziomu użytkownika bez uprawnień administratora. W systemie Ubuntu możliwy jest dostęp do danych binarnych i rejestrów urządzeń, co w określonych scenariuszach mogłoby posłużyć do rekonstrukcji działania sprzętu, pasywnego podsłuchiwania czy nawet prób manipulacji. System Windows również udostępnia istotne informacje o sprzęcie, jednak głównie w formie metadanych możliwych do odczytania przez *PowerShell*.

Środowiska wirtualne skutecznie ograniczają dostęp do warstwy sprzętowej. Hipernadzorca pełni funkcję filtra, dzięki któremu tylko jawnie przekazane urządzenia mogą być widoczne i obsługiwane przez system gościa. W praktyce oznacza to, że użytkownik działający w maszynie wirtualnej, nawet jeśli posiada wiedzę techniczną i narzędzia do analizy sprzętu, nie będzie w stanie dotrzeć do urządzeń, których nie udostępnił mu administrator hipernadzorcy. To odseparowanie znacząco utrudnia eskalację uprawnień opartą na interakcji z urządzeniami fizycznymi.

### Podsumowanie rozdziału i wnioski dotyczące eskalacji uprawnień

W rozdziale poświęconym testom eskalacji uprawnień przeprowadzono szereg szczegółowych eksperymentów mających na celu ocenę możliwości uzyskania wyższych uprawnień przez użytkowników nieuprzywilejowanych, zarówno w środowiskach fizycznych, jak i wirtualnych. Analiza objęła systemy Ubuntu i Windows uruchomione natywnie oraz w maszynach wirtualnych VMware. Choć testy dotyczyły obu platform, znaczna część scenariuszy została zrealizowana właśnie w systemie Ubuntu. Wynika to z faktu, że Linux, jako system otwartoźródłowy (ang. *open-source*), udostępnia użytkownikom znacznie więcej informacji o systemie, jego strukturze i sprzęcie, umożliwiając dogłębną analizę mechanizmów działania oraz potencjalnych wektorów ataku. Dzięki temu możliwe było zbadanie takich aspektów jak dostęp do pamięci jądra poprzez /dev/mem, analiza plików *SUID* czy bezpośrednia interakcja z urządzeniami USB i PCI poprzez przestrzenie /dev oraz /sys. W systemie Windows wiele z tych operacji jest domyślnie zablokowanych lub realizowanych poprzez warstwy pośrednie, takie jak rejestr, *PowerShell* lub usługi systemowe, co utrudnia przeprowadzenie analogicznych testów w sposób technicznie równoważny.

Środowiska wirtualne skutecznie ograniczają dostęp do warstwy sprzętowej, zwłaszcza w kontekście urządzeń PCI i USB, co znacząco utrudnia eskalację uprawnień bazującą na analizie lub manipulacji sprzętem. W przypadku klasycznych technik, takich jak wykorzystanie plików *SUID* czy próby dostępu do /dev/mem, różnice między środowiskiem natywnym a wirtualnym są pomijalne. Skuteczność ataku zależy głównie od konfiguracji systemu, a nie od jego osadzenia w maszynie fizycznej czy wirtualnej.

Choć wirtualizacja nie eliminuje wszystkich wektorów ataku, znacząco ogranicza te związane z fizycznym dostępem do sprzętu. Tym samym maszyny wirtualne stanowią efektywną warstwę ochronną w scenariuszach, gdzie pełna izolacja sprzętowa użytkowników jest istotna z punktu widzenia bezpieczeństwa.

## Kanały integracyjne między systemem gospodarza a gościa w VMware (schowek, foldery współdzielone) – ryzyko nieautoryzowanego wycieku danych

Wirtualizacja dostarcza mechanizmów integracyjnych pomiędzy systemem gospodarza a maszynami wirtualnymi, które podnoszą użyteczność środowiska, lecz jednocześnie tworzą kanały przepływu danych. W kontekście VMware Workstation są to przede wszystkim funkcje realizowane przez VMware Tools, m.in. współdzielony schowek (ang. *Clipboard*) oraz współdzielone foldery (ang. *Shared Folders*). W odróżnieniu od klasycznych scenariuszy komunikacji sieciowej (NAT/bridged/host-only), działanie tych kanałów nie zależy od topologii sieci wirtualnej, lecz od konfiguracji integracji systemu gospodarza i systemu gościa oraz aktywności komponentów *Tools*.

A pink oval with black text

AI-generated content may be incorrect.

Rys. 5.1. Stanowisko: host Windows i gość (Windows/Ubuntu); kanały VMware Tools — schowek i współdzielone foldery.

Celem rozdziału jest weryfikacja, w jakim stopniu konfiguracja kanałów między gościem a gospodarzem wpływa na możliwość i kierunek transferu danych oraz jakie ryzyka z tego wynikają. Analizie poddano dwa mechanizmy: Schowek, włączony w obu kierunkach oraz Foldery współdzielone, w trybach *Disabled*, *Read-only* i *Read-write*, z oceną odczytu i zapisu po obu stronach.

### Schowek – przebieg i wyniki

Dla Schowka wykonano pięć prób na kierunek (host do gość oraz gość do host) z pomiarem czasu w narzędziach systemowych. W Ubuntu użyto *wl-clipboard* (*Wayland*). Integralność sprawdzano przez SHA-256[[8]](#footnote-8).

We wszystkich próbach uzyskano pełną skuteczność transferu tekstu w obu kierunkach (z hosta do gościa oraz z gościa do hosta). W systemie Windows jako gościu czasy pojedynczych operacji mieściły się w zakresie około 0,001–0,004 s, a w systemie Ubuntu było to od około 0,03 do około 0,06 s. We wszystkich przypadkach potwierdzono integralność przekazywanej treści (zgodność sum kontrolnych SHA-256 pomiędzy źródłem a plikiem wynikowym). Pomiary prowadzono w stabilnym kontekście sesji użytkownika. Obserwacje pokazały również, że przełączanie kont w trakcie testów może prowadzić do braku synchronizacji schowka pomiędzy systemami, co nie jest równoznaczne z wyłączeniem funkcji.

|  |  |  |
| --- | --- | --- |
| Próba | Kierunek | Czas [s] |
| 1 | z hosta do gościa | 0,00398 |
| 2 | z hosta do gościa | 0,00198 |
| 3 | z hosta do gościa | 0,00247 |
| 4 | z hosta do gościa | 0,00308 |
| 5 | z hosta do gościa | 0,00238 |
| 6 | z gościa do hosta | 0,00155 |
| 7 | z gościa do hosta | 0,00123 |
| 8 | z gościa do hosta | 0,00086 |
| 9 | z gościa do hosta | 0,00167 |
| 10 | z gościa do hosta | 0,00092 |

Tab. 4.9. Schowek — *Windows jako gość*: czasy operacji „z hosta do gościa” i „z gościa do hosta”.

|  |  |  |
| --- | --- | --- |
| Próba | Kierunek | Czas [s] |
| 1 | z hosta do gościa | 0,05000 |
| 2 | z hosta do gościa | 0,03000 |
| 3 | z hosta do gościa | 0,04000 |
| 4 | z hosta do gościa | 0,05000 |
| 5 | z hosta do gościa | 0,06000 |
| 6 | z gościa do hosta | 0,00384 |
| 7 | z gościa do hosta | 0,00174 |
| 8 | z gościa do hosta | 0,00180 |
| 9 | z gościa do hosta | 0,00210 |
| 10 | z gościa do hosta | 0,00209 |

Tab. 4.10. Schowek — *Ubuntu jako gość*: czasy operacji „z hosta do gościa” i „z gościa do hosta”

W badaniach nie wykorzystywano hosta z systemem Ubuntu. System ten występował wyłącznie jako maszyna wirtualna, a odczyty „z gościa do hosta” dotyczą hosta Windows. Różnica czasów pomiędzy gościem Windows a gościem Ubuntu ma charakter operacyjny (narzędzia użytkownika i środowisko graficzne). W badanej konfiguracji schowek stanowi szybki i powtarzalny kanał wymiany danych pomiędzy hostem a maszynami wirtualnymi. Sama obecność mechanizmu, przy aktywnych komponentach narzędziowych w bieżącej sesji użytkownika, umożliwia niemal natychmiastowy transfer tekstu poza płaszczyzną sieci.

### Współdzielone foldery – przebieg i wyniki

Dla oceny mechanizmu utworzono dwa udziały po stronie hosta. Pierwszy w trybie tylko do odczytu (*Read-only*), natomiast drugi z prawem zapisu (*Read-write*). Tryb *Disabled,* zgodnie z oczekiwaniami, skutkował brakiem widoczności udziałów i uniemożliwiał wykonanie jakichkolwiek operacji wejścia–wyjścia.

A screenshot of a computer error

AI-generated content may be incorrect.  
Fot. 4.11. Badanie trybu Read-only – nieudana próba utworzenia nowego pliku w folderze współdzielonym z poziomu maszyny wirtualnej.

W trybie *Read-only* odczyt z hosta do gościa był w pełni skuteczny, natomiast zapis z gościa do hosta był blokowany (fot. 4.11). W trybie *Read-write* odczyt i zapis działały w obu kierunkach. We wszystkich przypadkach potwierdzono integralność treści zgodnością sum kontrolnych. Czas przenoszenia niewielkich plików był pomijalny, a dla większych zależał głównie od podsystemu dyskowego hosta.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tryb** | **Widoczność udziału w gościu** | **Odczyt z hosta do gościa** | **Zapis z gościa do hosta** | **Uwagi** |
| Disabled | Nie | Nie | Nie | Kanał wyłączony. |
| Read-only | Tak | Tak | Nie | Jednokierunkowy przepływ informacji z hosta. |
| Read-write | Tak | Tak | Tak | Dwukierunkowa wymiana plików; integralność potwierdzona. |

Tab. 4.11. Współdzielone foldery — wyniki.

Współdzielone foldery stanowią kanał przepływu danych niezależny od konfiguracji sieci. Tryb *Read-only* pozwala na kontrolowany, jednokierunkowy wgląd gościa w wybrane zasoby hosta, natomiast *Read-write* otwiera pełną wymianę plików, co istotnie zwiększa powierzchnię ryzyka.

Wyniki były zgodne dla systemu Windows oraz Ubuntu i potwierdzają, że integracje host-gość istotnie modyfikują powierzchnię ryzyka w środowiskach wirtualnych. Decydująca jest konfiguracja mechanizmów *Tools*. Ich wyłączenie lub restrykcyjne ustawienie ogranicza możliwość wycieku danych poza płaszczyzną sieci, a także ogranicza zagrożenia wynikające np. z możliwości przeniesienia plików wykonywalnych, zawierających złośliwy kod.

# Zakończenie i wnioski

Przeprowadzone badania potwierdzają tezę, że bezpieczeństwo systemów operacyjnych w maszynach wirtualnych jest zasadniczo porównywalne z systemami natywnymi, a różni się profil ryzyka. Dobrze widać to na przykładzie rozdziałów 4.4.6 oraz 4.5. Rozdział 4.4.6 pokazuje, że wirtualizacja skutecznie ograniczyła potencjalne ryzyka ataków, poprzez ograniczenie dostępu do urządzeń podłączonych poprzez porty USB i PCI, natomiast rozdział 4.5 przedstawia, że warstwa wirtualizacji (szczególnie przy nieprzemyślanej konfiguracji) stwarza nowe zagrożenia, które narażają system gospodarza na np. przedostanie się złośliwego kodu.

Testy sieciowe, *sniffing* oraz *brute-force,* nie wykazały żadnych różnic w zachowaniu między systemem natywnym, a wirtualnym. Warto zaznaczyć, że tryb wykorzystany tryb sieciowy *bridged* w maszynie wirtualnej odwzorowuje warstwy L2 (łącza danych) i L3 (sieci) jak na fizycznym hoście. Maszyna wirtualna dostaje własny adres MAC/IP w tej samej podsieci, a hipernadzorca przekazuje ramki bez ingerencji, więc zachowanie sieciowe nie różni się od tego w systemie natywnym.

Pasywny *sniffing* w sieci (domowy router/switch, Wi‑Fi) i tak nie zobaczy obcego ruchu. Ograniczenie wynika z topologii/przełączania, nie z tego, czy ofiara to maszyna wirtualna czy fizyczny komputer, więc wynik w obu przypadkach jest taki sam, czyli brak przechwyconych pakietów innych hostów.

W ataku *brute*‑*force* wąskim gardłem była aplikacja webowa bez mechanizmów obronnych, nie rodzaj środowiska. Ta sama aplikacja i ten sam profil ruchu dają zbliżoną przepustowość prób w maszynach wirtualnych i natywnych oraz w porównywanych systemach.

Nieco odmienne wyniki dały testy *ARP spoofing*, ponieważ odnotowane zostały różnice zarówno między systemami, jak i środowiskiem uruchomieniowym. W ataku tym duże znaczenie odgrywa warstwa łącza danych (L2) i dynamika tablic ARP po obu stronach. Jest to atak aktywny, co za tym idzie, liczy się czas na podtrzymanie fałszywych wpisów, odpowiedzi bramy czy też na odświeżanie pamięci podręcznej [25]. Obecność lub brak zapory na hoście nie różnicowała istotnie wyników, bo firewall na hoście nie filtruje ARP. Czasami w danych wyglądało to paradoksalnie (nieco lepiej „z włączoną zaporą”), ale to efekt uboczny zmienności czasowej i ruchowej, ponieważ sukces zależał od momentu „zatrucia” pamięci podręcznej względem żądań HTTP, częstotliwości „*re‑poisoningu*” i naturalnego ruchu sieciowego. Wynika z tego, że *ARP spoofing* pozostaje realnym ryzykiem niezależnie od tego czy atak przeprowadzany jest na maszynę wirtualną czy system natywny.

Scenariusze oparte o błędy konfiguracyjne (*SUID* na /bin/bash, NOPASSWD dla *nano*) prowadziły do natychmiastowej i pełnej eskalacji w identyczny sposób na maszynie wirtualnej i w systemie natywnym, co podkreśla, że jakość administracji i kontrola zmian jest decydująca, a wirtualizacja nie stanowi tu bariery. Z drugiej strony, mechanizmy jądra skutecznie blokowały potencjalne nadużycia /dev/mem w obu środowiskach, co pokazuje wagę twardych polityk jądra i aktualnego stosu bezpieczeństwa. Próba *Rowhammer* nie wykazała błędów bitowych w badanych warunkach. Obserwacje są zatem spójne z obecnością współczesnych zabezpieczeń (np. *ECC*, izolacja pamięci). Choć teoretycznie środowisko natywne może być bardziej narażone na efekty fizyczne, testy nie dostarczyły empirycznego potwierdzenia różnic.

Przeprowadzenie testów przy wykorzystaniu dwóch odmiennych systemów, czyli Windows i Linux, przede wszystkim ograniczyło wpływ specyfiki systemu operacyjnego na wyniki, dzięki czemu łatwiej można było wyciągnąć wnioski dotyczące zachowania maszyn wirtualnych. Dodatkowo, można było porównać omawiane systemy operacyjne pod kątem bezpieczeństwa. Nie można wskazać jednoznacznie, który system jest bardziej odporny na ataki, ponieważ różnią się one przede wszystkim modelem zarządzania uprawnieniami, domyślną konfiguracją mechanizmów ochronnych oraz zakresem dostępu użytkownika do zasobów systemowych, co prowadzi do odmiennego profilu ryzyka w każdym z przypadków.

Wirtualizacja sama z siebie nie podnosi ani nie obniża bezpieczeństwa. Z jednej strony odwzorowuje zachowanie sieci, więc nie zmienia podatności na ataki sieciowe, z drugiej ogranicza dostęp do fizycznych urządzeń, co redukuje wektory sprzętowe. Jednocześnie dodaje nowe kanały i wymaga zarządzania warstwą hipernadzorcy. Ostateczny poziom bezpieczeństwa zależy więc głównie od konfiguracji systemu lub aplikacji, polityk hipernadzorcy i infrastruktury sieciowej.

# Literatura

[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.

[20] Scarfone K., Souppaya M., Hoffman P., *Guide to Security for Full Virtualization Technologies (NIST SP 800-125)*

[21] Pék, Buttyán, Bencsáth (2013): *A Survey of Security Issues in Hardware Virtualization*, ACM Computing Surveys 45(3)

[22] Glăvan D., Răcuciu C., Moinescu R., Eftimie S., *Sniffing attacks on computer networks*, *Scientific Bulletin of Naval Academy*, t. XXIII, nr 1, 2020.

[23] Mahipal S., Ceronmani Sharmila V., *Virtual Machine Security Problems and Countermeasures for Improving QoS in Cloud Computing*.

[24] Were M., Singh R. S., Odawa J., *Security Evaluation Framework for Virtualised Environments*.

[25]  Alina & Saraswat (RDCAPE 2021)

# 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

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 przesłanych przez protokół HTTP. [↑](#footnote-ref-3)
4. Bit SUID - atrybut plików wykonywalnych w systemach uniksowych, który powoduje, że proces uruchomiony z takiego pliku działa z uprawnieniami właściciela. [↑](#footnote-ref-4)
5. S*udo* – narzędzie systemów Unix/Linux umożliwiające wykonanie polecenia z uprawnieniami innego użytkownika, najczęściej administratora. Stosowane jest w celu tymczasowego podniesienia uprawnień bez konieczności logowania się na konto administratora. [↑](#footnote-ref-5)
6. Nano - prosty edytor tekstu w terminalu Linux/Unix, służący do edycji plików systemowych i skryptów. [↑](#footnote-ref-6)
7. *Wrapper* - prosty skrypt lub program pośredniczący, który ogranicza funkcjonalność wywoływanego polecenia, np. pozwala na wykonanie tylko jednej, ściśle zdefiniowanej operacji zamiast pełnego dostępu do programu. [↑](#footnote-ref-7)
8. SHA-256 to funkcja kryptograficzna, przekształcająca dowolne dane wejściowe w 256-bitowy unikalny ciąg znaków. Wykorzystuje się ją m.in. do weryfikacji autentyczności i integralności danych. [↑](#footnote-ref-8)