

Własne środowisko do ćwiczeń red team

Członkowie ofensywnych zespołów bezpieczeństwa muszą opanować wiele aspektów i niuansów dotyczących środowisk, w których prowadzą swoje operacje. Większość potrzebnych umiejętności wynika nie z przyswojenia suchych faktów, lecz zebrania odpowiedniego doświadczenia wynikającego z praktyki. Niestety systemy produkcyjne, z oczywistych względów, nie nadają się do ćwiczeń bardziej skomplikowanych mechanizmów ataków. W sukurs przyjść mogą różnorakie platformy (najczęściej wymagające opłaty) lub ręczne przygotowanie środowiska testowego. Ten ostatni krok, często wybierany przez zawodowców, ma dwie stosunkowo duże wady. Po pierwsze, jest to proces żmudny i czasochłonny, a po drugie mało elastyczny. Częsta rekonfiguracja środowiska testowego zaczyna pochłaniać znaczną część czasu zespołu. Aby temu zapobiec, powstały różne mechanizmy automatyzacji przygotowania takiego środowiska.

Iw niniejszym artykule zajmiemy się jedną z nich – Ludusem. Należy wspomnieć, że stworzenie pełnoprawnego środowiska do działań ofensywnych jest też szansą dla zespołu niebieskiego – defensywnego do przetestowania swoich możliwości detekcji oraz wdrożonych mitygacji.

I CZYM JEST LUDUS

Ludus to system automatyzacji budowy środowisk (ukierunkowany na przygotowanie pod kątem laboratoriów do przeprowadzania testów bezpieczeństwa), który wykorzystuje u swojej podstawy Proxmoxa. Ten ostatni to otwartoźródłowy hipernadzorca typu pierwszego (bare metal), który instaluje się bezpośrednio jako warstwę „systemu operacyjnego”. W tym konkretnym przypadku implementacja opiera się na systemie Debian. Proxmox wykorzystuje technologie takie jak LXC oraz KVM do dostarczenia zwirtualizowanych systemów. Oprócz tego dysponuje wygodnym API oraz interfejsem webowym, które ułatwiają korzystanie z niego.

API to kluczowy element, który pozwala zintegrować automatyzację z Ludusem. Z punktu widzenia architektury Ludusa to właśnie postawienie na automatyzację jest największym argumentem za tym rozwiązaniem. Dzięki temu można przygotować określone scenariusze ćwiczeń, obejmujące przemyślane wektory ataku, zaprogramować je zgodnie z doktryną Infrastructure as a Code (IaC), udostępnić określonym osobom i odtwarzać tak długo, jak to tylko potrzebne. Bez konieczności żmudnego przywracania, ponieważ warstwa orkiestracji pozwala na zdefiniowanie zależności – np. fakt, że kontroler domeny (w scenariuszu dotyczącym atakowania całej usługi katalogowej) powinien „wstać” przed resztą systemów.

Możliwość przywracania środowiska w sposób zautomatyzowany ułatwia iteracyjne podejście do ćwiczeń blue team/purple team. Ilekroć środowisko zostanie nieodwracalnie uszkodzone w wyniku ataku (np. poprzez podmienienie kluczowych binarek w systemie plików), może zostać w prosty sposób odtworzone, bez konieczności manualnego wykorzystywania przygotowanych migawek maszyn.

DLACZEGO WARTO ROZBUDOWAĆ WŁASNE ŚRODOWISKO DO RED TEAMINGU

W obecnych czasach truizmem jest stwierdzenie, że obraz odporności organizacji zmienia się w sposób bardzo dynamiczny. To samo można stwierdzić o prawie każdym innym aspekcie, który dotyczy nowoczesnych technologii. Niestety utrzymywanie zespołu ofensywnego bezpieczeństwa to proces żmudny. Umiejętności nie można na być i utrzymywać, scrollując wpisy na X (dawniej Twitter) czy posty na blogach tematycznych. Wymagane jest do tego przeprowadzanie prawdziwych lub symulowanych operacji ofensywnych. Niestety nie zawsze organizacja może zapewnić sobie komfort ciągłych testów ofensywnych przeprowadzanych przez zespół czerwony. Dlatego dobrym pomysłem jest wirtualizacja środowiska (może z wykorzystaniem elementów charakterystycznych dla danej organizacji jak np. narzędzia zbierające telemetrię lub zastosowane oprogramowanie typu EDR/XDR) i wielotorowe rozwijanie nie tylko zdolności ofensywnych, ale także znajomości narzędzi.

W czasie niebywałego boomu na generatywne AI (w postaci dużych modeli językowych) zespoły ofensywne również chcą przetestować możliwości agentowych systemów opartych na LLM-ach w zderzeniu z realnymi podatnościami. O ile bardzo nieodpowiedzialnym byłoby uruchomienie takich systemów na produkcyjnej infrastrukturze organizacji (z wielu powodów), o tyle zwirtualizowane środowisko, gotowe do szybkiego ponownego uruchomienia w innej konfiguracji lub przywrócenia do stanu początkowego, wydaje się być doskonałym benchmarkiem możliwości takich narzędzi.

Z doświadczenia autora, który bierze regularnie udział w ćwiczeniach red team, ważne do ewaluacji własnych umiejętności, ale także zrozumienia taktyk zespołów broniących i poznania ich narzędzi, jest możliwość spojrzenia i wcielenia się w rolę obrońcy. Przygotowane środowisko może integrować te same narzędzia, z których korzystają zespoły defensywne. Dzięki takiej organizacji środowiska pracy możliwe jest nie tylko testowanie technik, ale także podglądanie

i śledzenie, jakie artefakty, zdarzenia i alarmy mogą one generować oraz jakimi narzędziami będzie się posługiwać zespół broniący. Posiadając tę wiedzę, można stworzyć własne modyfikacje technik, które będą dużo skuteczniejsze w konkretnych kontekstach scenariuszy oraz mniej widoczne (trudniejsze do wykrycia) przez zespoły broniące. Wysoka znajomość kontekstu znacznie podwyższa skuteczność ataku, a to pozwala z kolei przygotować organizację na bardziej wysublimowane ataki przeprowadzane przez lepiej przygotowanych atakujących.

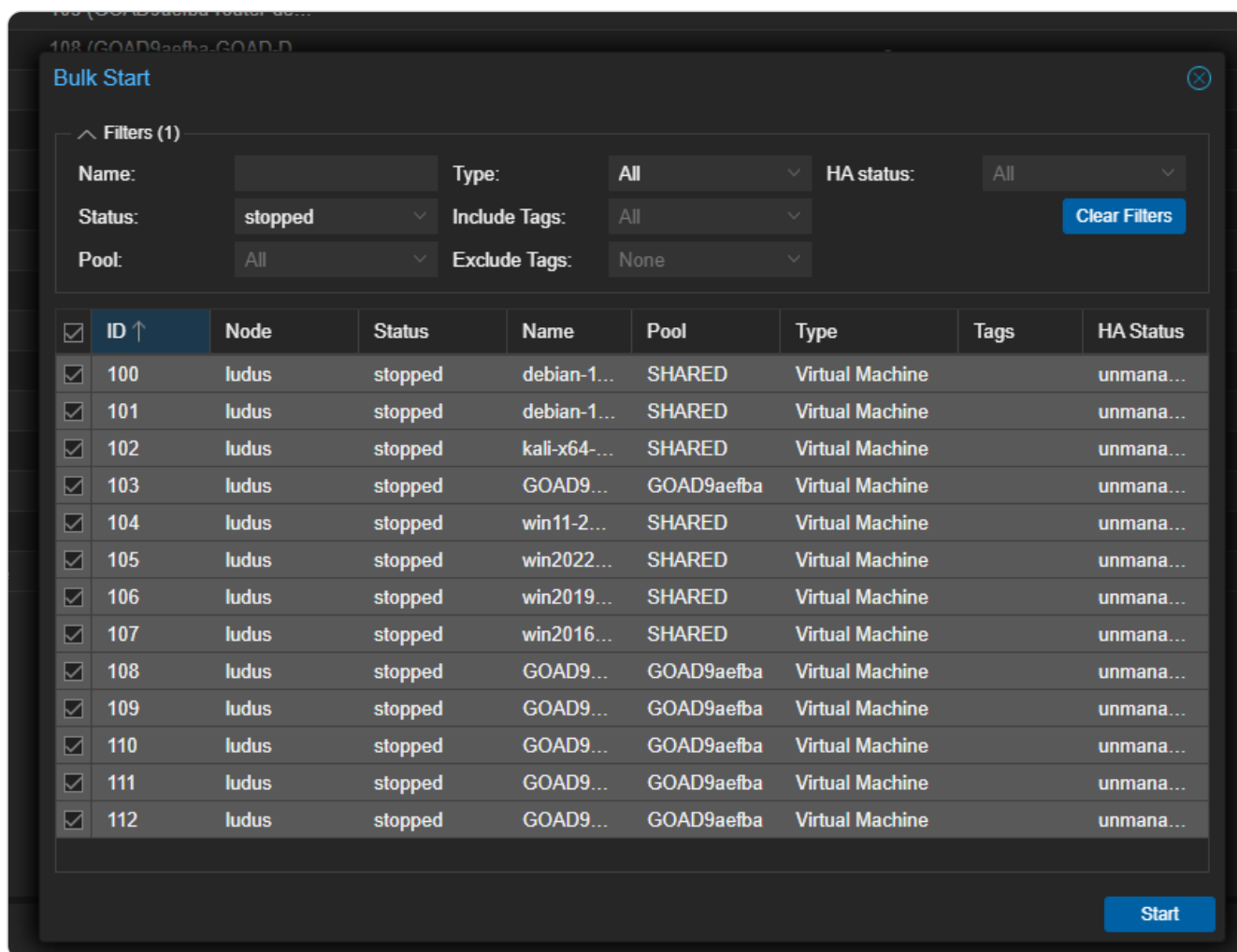
Takie podejście do zagadnienia testowych „labów” pozwala, po inwestycji związanej z infrastrukturą, opierać się na własnych zasobach bez konieczności regularnego opłacania abonamentów za testowe środowiska, oferowane przez kilka firm związanych z cyberbezpieczeństwem. Dodatkową zaletą jest fakt, że operatorzy red team będą musieli poświęcić chwilę na przygotowanie scenariuszy. Z jednej strony jest to czas, w którym nie szkolą się w atakowaniu, z drugiej strony jest to możliwość poznania architektury atakowanych systemów. Prawdziwy haker (definicja autora jest dalece inna od potocznej, błędnej prezentowanej w mediach, która wskazuje, że jest to osoba dokonująca nielegalnych czynów przy pomocy komputera) to specjalista, który wie, jak może uszkodzić dany system, wykorzystać pewne mechanizmy na swoją korzyść właśnie dlatego, że doskonale rozumie

zasadę działania i umie odtworzyć dane rozwiązanie, a nie dlatego, że poznał kilka magicznych komend i nauczył się przekazywać wyjście jednego programu na wejście kolejnego. Dlatego z tego miejsca serdecznie zachęcam do zabawy i stworzenia własnego lab'a, nawet jeśli nie w organizacji, to w bycie określanym mianem homelab. Rozsądnie przeprowadzona konfiguracja nie zmniejsza bezpieczeństwa sieci, za to dostarcza dodatkowe narzędzia, kolejnego miejsca, które pozwala na bezpieczne psucie i rozkładanie na czynniki pierwsze.

Przed przystąpieniem do budowy specjalnego, podatnego środowiska należy jednak skonsultować swój przypadek użycia oraz posiadane licencje oprogramowania z EULA producenta oprogramowania, aby nie naruszać warunków licencyjnych korzystania z systemów operacyjnych. W niniejszym artykule wykorzystane zostaną „trochę” zgodne z ideą Ludusa, ewaluacyjne licencje Microsoftu, ważne przez 180 dni. Należy jednak pamiętać, że np. budowanie trwałych labów może wymagać zakupu odpowiednich licencji.

I ARCHITEKTURA ROZWIĄZANIA

To, co wnikliwy czytelnik już wywnioskował z jednego z poprzednich akapitów, to fakt, że Ludus nie jest hipernadzorcą – to element orkiestracji – zautomatyzowanego zarządzania środowiskiem. Z tego



Rysunek 1. Zestawienie maszyn w webowej konsoli Proxmoxa

względę wprowadza i rozszerza pewną warstwę abstrakcji nad wirtualizacją zapewnioną przez Proxmoxa (Rysunek 1). Aby sprawnie poruszać się po tym środowisku, należy poznać kilka terminów, które ułatwią zrozumienie koncepcji Ludusa.

Bazowym kompletnym obiektem, z którego buduje się większe sieci, jest oczywiście pojedyncza wirtualna maszyna. Ale żeby móc ją powtarzalnie budować, potrzebny jest przepis oraz orkiestrator, który przygotuje ten bazowy element. W przypadku omawianego rozwiązania jest to Packer firmy Hashicorp, a sam „przepis” to nic innego jak definicja obrazu bazowego w postaci pliku konfiguracyjnego packera (pkr.hcl) oraz dodatkowych pomocniczych skryptów. Filozofia budowy wirtualnej maszyny w środowisku Ludus wskazuje potrzebę wykorzystania niezmodyfikowanych obrazów .ISO. Ma to na celu zapewnienie powtarzalności procesu budowania. Wymusza to też zawieranie wszystkich modyfikacji w konfiguracji packera, co znacznie obniża skomplikowanie projektu, usuwając konieczność przechowywania biblioteki różnych zmodyfikowanych obrazów (a przy okazji ułatwia porównywanie maszyn, skoro wszystkie zmiany są *explicite* wprowadzane przez konfigurator). Wynika to z podejścia IaC, chociaż bardziej wymagający użytkownicy mogą nagiąć trochę zasady i przy odrobinie wysiłku można wykorzystać customizowane obrazy początkowe.

Na start użytkownicy otrzymują następujące szablony:

- » debian11
- » debian12
- » kali
- » win11-22h2-x64-enterprise
- » win2022-server-x64

Bibliotekę bazowych obrazów można rozszerzyć, w Internecie [1] pełno jest dostępnych obrazów zawierających między innymi konfigurację z gotowych szablonów.

W przypadku większości bazowych systemów operacyjnych nie powinno być problemu z automatyzacją budowania szablonów, dokumentacja Ludusa wskazuje jednak problematyczną (acz możliwą) obsługę systemów z rodziny macOS, które pozbawione są wsparcia agenta QEMU.

Maszyny wirtualne można następnie logicznie uporządkować w tzw. range. Range (strzelnica?) to zbiór skonfigurowanych maszyn z dodatkowymi użytkownikami, wpięte do wydzielonych logicznie wirtualnych sieci. Proces budowy i uruchomienia gotowego range jest nazywany *deployem*, a przywrócenie do pierwotnego stanu (przez ponowny *deploy*, a nie sprzątanie artefaktów działań) *redeployem*/resetem środowiska.

Niestety duże możliwości oraz stosunkowo duża elastyczność Ludusa powoduje, że wymagania sprzętowe są całkiem znaczne. Z własnego doświadczenia autor może potwierdzić, że o ile używalne laboratorium można zbudować przy pomocy starego komputera klasy miniPC, to jednak dobrze działające środowisko wymaga większych zasobów. Do niewielkich (do max. 8-10 maszyn) wystarczy środowisko z 64GB RAM i 8-12 rdzeni procesora. Oczywiście CPU musi wspierać wirtualizację. W przypadku większych symulacji (~15 maszyn) potrzebne będzie 128GB RAM, powyżej 16 rdzeni. Niezależnie od przypadku użycia warto zaopatrzyć się w dyski NVME przynajmniej 2TB. Empirycznie przetestowane dane zostały zaprezentowane w Tabeli 1.

Profil symulowanego środowiska	Liczba VM	RAM (host)	Uwagi
Mały	8-10	>= 64 GB	AD + 1-2 serwery + 2-3 stacje + maszyna atakującego (można wykorzystać VPN i własny komputer np. ze środowiskiem exegol)
Średni	12-18	>= 128 GB	Dodatkowe serwery, więcej stacji, elementy telemetrii
Duży	20-40	>= 256 GB	Wiele segmentów, SIEM/EDR, kilka teamów równolegle

Tabela 1. Minimalne wymagania sprzętowe w zależności od przypadku użycia wirtualnego środowiska

Kluczowe jest, aby host, wykorzystany do budowy podatnego labu, był wyposażony w porty ethernet, ponieważ występują problemy z łącznością przy pomocy bezprzewodowych kart sieciowych. Zaleca się również umieszczenie ludusowego hosta w testowej podsieci, z bardzo okrojonym dostępem sieciowym (dostęp do samego środowiska będzie realizowany przez VPN), ponieważ (oprócz oczywiście podatnych szablonów) środowisko to wyłącza wszelkie mitygacje na znane ataki celujące w spekulacyjne wykonywanie operacji przez procesor.

Niestety w momencie, kiedy powstaje ten artykuł (luty 2026), ceny podzespołów komputerowych, nakręcane spekulacją na bańce AI, wystrzeliły w górę, więc najlepiej będzie (niczym w programach pana Słodowego) wyjąć potrzebne elementy z szuflady biurka. Wciąż jednak, przy prężnie rozwijającym się zespole, inwestycja we własne środowisko może być dużo bardziej opłacalna niż opłacanie miesięcznych licencji korporacyjnych w firmach, które udostępniają podobne środowiska.

PRZYGOTOWANIE ŚRODOWISKA URUCHOMIENIOWEGO

Po zdobyciu i skonfigurowaniu odpowiedniego hardware należy skupić się na warstwie oprogramowania. Jak zostało wspomniane na samym początku artykułu, Ludus potrafi zarządzać środowiskami opartymi na Proxmoxie. Nie jest to oczywiście jedyna możliwość, bo wspierane są także scenariusze wykorzystujące:

- » Google Cloud Platform
- » Hyper-V
- » VMware Fusion
- » Azure

Jednak z punktu widzenia autora najbardziej kusząca wizja to instalacja „bare metal”, na sprzęcie posiadanym fizycznie w laboratorium. Dokumentacja sugeruje użycie internetowej (netinstall) instalacji Debiana 13, który jest w chwili pisania niniejszego artykułu stabilną dystrybucją. A ponieważ sama konfiguracja bazowego systemu operacyjnego jest zadaniem prostym (a przynajmniej powinna być), to zostanie ona pominięta w tym artykule. Warto pamiętać o konieczności uruchomienia serwera SSH, bo to z poziomu zdalnej powłoki będą się odbywały następne kroki instalacji.

programista

1/2026 (122)

Cena 32.90 zł (w tym VAT 8%)

WŁASNE ŚRODOWISKO DO ĆWICZEŃ RED TEAM



ISSN 2084-9400



9 772084 940602

Conan – menadżer pakietów,
którego potrzebuje projekt C++

Alternatywne sposoby
zmiennorodzajowych

MicroPython – lekki Python dla
mikrokontrolerów i nie tylko

Procedury generacji muzyki
(na serio)

Nowy numer już w empikach