W głowie hakera – jak chronić własność intelektualną w erze AI?

W dobie rosnącej roli sztucznej inteligencji i uczenia maszynowego zabezpieczenie własności intelektualnej zawartej w modelach i aplikacjach staje się kluczowe. Nowoczesne algorytmy, trenowane na poufnych danych, są dziś jednym z najcenniejszych aktywów firm technologicznych. Niestety, stanowią również atrakcyjny cel dla cyberprzestępców. Aby lepiej zrozumieć zagrożenia, przyjmijmy na moment perspektywę hakera – jego metody działania oraz możliwe strategie obronne.

Rosnące znaczenie ochrony IP

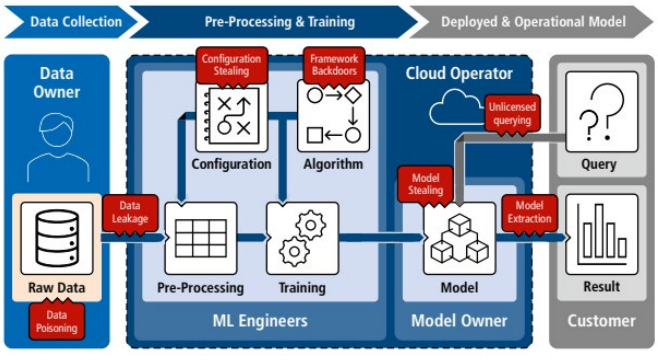
Zgodnie z raportem VDMA (2024) naruszenia własności intelektualnej są nadal powszechne. Plagiaty i nielegalna reprodukcja są często inicjowane przez konkurencję, partnerów lub zorganizowaną przestępczość.

Branże najbardziej narażone to m.in. sektor medyczny, energetyka, motoryzacja i obszary z infrastrukturą krytyczną. W tych przypadkach skuteczny atak może mieć poważne konsekwencje – od utraty danych po zagrożenie życia.

AI i ML – nowe pole bitwy

Cykl życia modeli ML (MLL – Machine Learning Lifecycle) obejmuje m.in. zbieranie danych, przetwarzanie, trening, walidację i wdrożenie. Każdy z tych etapów stanowi potencjalny wektor ataku. Z perspektywy hakera:

- dane treningowe mogą zawierać informacje wrażliwe (np. medyczne),
- algorytmy preprocessingowe (np. skalowanie obrazów) można zmanipulować,
- parametry i architektura modelu mogą zostać skradzione lub odtworzone,
- sama aplikacja może zostać zreverse-engineerowana.



Rysunek 1. Powierzchnie ataku w cyklu życia uczenia maszynowego

Przykład ataku: data poisoning

Haker może wprowadzić subtelne manipulacje do danych wejściowych (np. modyfikując obrazy na poziomie subpikseli), które po przetwarzaniu prowadzą do nieprawidłowej klasyfikacji. Tego typu ataki są trudne do wykrycia, a ich skutki mogą być katastrofalne – szczególnie w kontekście AI w medycynie.

Kradzież modelu (model stealing)

Istnieją dwa podstawowe scenariusze:

1. Kopia modelu – atakujący uzyskuje dostęp do wytrenowanej sieci neuronowej (np. poprzez skompilowaną aplikację) i kopiuje jej architekturę oraz parametry.
2. Trenowanie modelu zastępczego – na podstawie zapytań do API modelu źródłowego tworzony jest model o zbliżonej funkcjonalności.

Ochrona samego modelu nie wystarczy – równie ważne jest zabezpieczenie **kodu aplikacji**, który często zawiera krytyczne fragmenty logiki biznesowej oraz „fine-tuningi".

Ataki na poziomie aplikacji

Typowe techniki stosowane przez hakerów to:

- Patching – modyfikacja bajtów kodu w pamięci w celu usunięcia np. sprawdzania licencji.
- Debugging – analiza programu krok po kroku.
- Memory Dumping – przechwycenie obrazu pamięci z działającej aplikacji.
- Hooking – wstrzykiwanie bibliotek (DLL) i modyfikacja działania w czasie rzeczywistym.
- Program simplification – uproszczenie kodu przez dekompilatory lub narzędzia optymalizujące.

Strategie obronne

Skuteczna ochrona przed powyższymi technikami wymaga kombinacji kilku podejść:

1. **Szyfrowanie** – chroni dane i kod w stanie spoczynku. Wymaga odszyfrowania przed wykonaniem – dlatego w czasie działania potrzebne są dodatkowe środki ochrony.
2. **Kontrola integralności.** Weryfikacja spójności kodu i danych. Każda manipulacja (np. zamiana instrukcji na breakpoint) może zostać wykryta. Silne podpisy kryptograficzne gwarantują integralność i autentyczność danych treningowych. W przypadku danych treningowych zaufana strona może podpisywać źródło. Wszelkie manipulacje złamią podpis, co uniemożliwi użycie danych do treningu.
3. Antydebugging. Techniki mające na celu wykrycie dynamicznej analizy. Stosowanie funkcji takich jak IsDebuggerPresent() czy sprawdzanie flag w strukturze PEB. Choć techniki te są dobrze znane, stanowią pierwszą linię obrony.

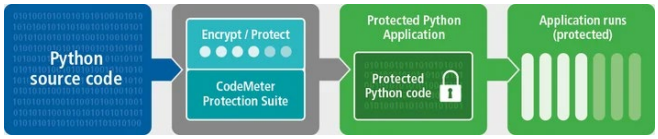
4. **Zaawansowana obfuskacja.** Zwiększa złożoność programu, aby maksymalnie utrudnić analizę nawet dla zrzutu pamięci. W przeciwieństwie do prostego zaciemniania, które jest podatne na optymalizacje kompilatora, zaawansowane techniki stosują bardziej zintegrowane i sprzężone transformacje, wpływające na przepływ programu przy zachowaniu oryginalnej semantyki. Przykładowe techniki to:
   - **Zależności stanowe** – sprawiają, że instrukcje działają inaczej, jeśli zmieniony zostanie porządek wykonania funkcji.
   - **Spłaszczanie przepływu sterowania** – transformuje kod w mikro-maszyny wirtualne.
   - **Przekierowanie przepływu sterowania** – ukrywa połączenia między blokami kodu.
   - **Przeplatanie funkcji** – łączy wiele funkcji w jedną gigantyczną, co utrudnia analizę. Celem jest maksymalne utrudnienie analizy statycznej, jednocześnie zachowując jak najlepszą wydajność.
5. Wykonywanie kodu w środowiskach zaufanych
6. Wyodrębnienie newralgicznych fragmentów kodu, zaszyfrowanie ich i wykonywanie w bezpiecznym środowisku, np. donglu sprzętowym lub kontenerze w chmurze. Najwyższy poziom bezpieczeństwa osiąga się poprzez wykonywanie kodu nie na sprzęcie atakującego, ale w zdalnym, bezpiecznym środowisku (np. dongle, chmura). Małe, krytyczne części kodu są ekstrahowane i szyfrowane, a następnie przesyłane do zaufanego środowiska wraz z danymi wejściowymi, gdzie są wykonywane. Wynik jest przesyłany z powrotem do programu.
7. Licencjonowanie modeli i ograniczenia użytkowania. Własność jest zawarta zarówno w modelu, jak i w kodzie aplikacji. Licencjonowanie modelu i aplikacji uniemożliwia uruchomienie jej bez ważnej licencji, co pozwala na ograniczenie użycia (np. model pay-per-use). Zapobiega to także atakom typu „query/output" mającym na celu ponowne wytrenowanie innych modeli, ponieważ eksploracja zachowania modelu jest ograniczona.

Skuteczne narzędzia do ochrony MLL

WIBU-SYSTEMS, autor pakietu rozwiązań chroniących zasoby cyfrowe, oferuje kompleksowe podejście do ochrony IP w AI. Pakiet **CodeMeter Protection Suite** pozwala na:

- szyfrowanie i obfuskację kodu,
- detekcję manipulacji (tamper detection),
- izolację fragmentów aplikacji (Blurry Box),
- licencjonowanie i kontrolę użycia (np. pay-per-use),

Z uwagi na dostępność frameworków open source i popularność Pythona aplikacje AI często są w nim rozwijane. **AxProtector Python** chroni zarówno kod frameworka używany do trenowania modelu, jak i dane wykorzystywane w cyklu uczenia maszynowego przed manipulacją, kradzieżą własności intelektualnej i nieautoryzowanym użyciem.



Rysunek 2. Integracja AxProtector Python z procesem CI/CD

AxProtector Python szyfruje aplikację Python metoda po metodzie. Poszczególne metody są odszyfrowywane tylko w czasie wykonywania, gdy są potrzebne. Odszyfrowywanie i przetwarzanie odbywa się w chronionym środowisku. Oryginalny kod Python jest przechowywany w pamięci tylko w czasie jego przetwarzania.

Podsumowanie

Wraz z rozwojem AI rośnie również powierzchnia ataku na aplikacje zawierające modele ML. Własność intelektualna jest dziś rozproszona – znajduje się zarówno w modelu, jak i kodzie aplikacji. Dlatego potrzebna jest wielowarstwowa ochrona, obejmująca zarówno zabezpieczenia statyczne (szyfrowanie, obfuskacja), jak i dynamiczne (integrity checks, środowiska zaufane).

Rozwiązania takie jak CodeMeter Protection Suite oferują możliwości zapewniające ochronę na wielu poziomach. Świadomość zagrożeń i wdrożenie odpowiednich środków bezpieczeństwa jest kluczowe dla ochrony IP w procesie wytwarzania oprogramowania.

JANUSZ HRYSZKIEWICZ

janusz.hryszkiewicz@wibu.com

Związany z IT od 25 lat, głównie w zakresie rozwiązań dedykowanych (Microsoft, Java, Python, SAP). Autor kilku rozwiązań dla biznesu, które stały się produktami. Obecnie zaangażowany w zwiększenie cyberodporności oprogramowania, AI oraz systemów wbudowanych.