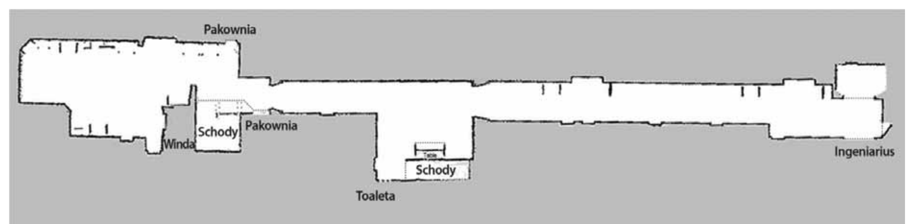
Bezpieczeństwo aplikacji robotów z wykorzystaniem ROS. Część 2

David Portugal, Miguel A. Santos, Samuel Pereira, Micael S. Couceiro

Wyniki i dyskusja

W tej części przedstawiono wyniki badań różnych propozycji zwiększenia bezpieczeństwa, które zostały opisane w poprzedniej części. Platforma testowa składała się z: procesora Intel i5-4590 (3,30 GHz), 8 GB pamięci RAM oraz 64-bitowej wersji systemu operacyjnego Ubuntu Linux 16.04 z systemem ROS Kinetic Kame. Wyniki w tej części koncentrują się na wydajności komunikacji każdego rozwiązania podczas przesyłania danych pomiędzy publikującym a *węzłem* subskrybenta działającym na tym samym komputerze. Pozwoliło to porównać opóźnienie w komunikacji, liczbę utraconych wiadomości, zdolność do nadążania za zamierzonymi prędkościami publikacji, poziomy dostępu z nieautoryzowanych węzłów w sieci ROS oraz ogólnie ocenić kompromis pomiędzy bezpieczeństwem a płynnością działania każdego podejścia.

W każdej z prób zdefiniowano dwa typy wiadomości, które mają być publikowane i subskrybowane przez dwa osobne węzły ROS: (I) „Hello World!" ciąg znaków z nagłówkiem zawierającym *znacznik czasu publikacji*, numer w sekwencji komunikatów i opcjonalny ciąg *frame_id* (który ustawiono na „0"), (II) mapa siatki *nav_msgs/Occupancy* zilustrowana na rysunku 1, która również zawierała przydatne informacje nagłówka. W tabeli 1 przedstawiono schemat przeprowadzonych eksperymentów. Dla każdego testowanego rozwiązania opublikowano 600 000 razy w trzech różnych zamierzonych szybkościach publikacji (1, 10 i 30 kHz) 27-bajtowy ciąg znaków z komunikatem nagłówka. Mapa rozkładu pomieszczeń, składająca się z 343 kB danych, została opublikowana 150 000 razy, również w trzech różnych planowanych



Rys. 1. Mapa rozkładu pomieszczeń transmitowana podczas eksperymentów (1187 × 296, z rozdzielczością 0,05 m/komórkę)

Tabela 1. Schemat badań przeprowadzonych dla każdej propozycji zabezpieczeń

Typ wiadomości	Łączny # wiadomości @ częstotliwość publikowania		
Ciąg znaków „Hello World!" z nagłówkiem (27 B)	600 tys. @ 1 kHz	600 tys. @ 10 kHz	600 tys. @ 30 kHz
Mapa rozkładu pomieszczeń (343 kB)	150 tys. @ 250 Hz	150 tys. @ 2,5 kHz	150 tys. @ 7,5 kHz

(a)
```
header:
  seq: 1
  stamp:
    secs: 1500660269
    nsecs: 676667209
  frame_id: 0
text: Hello World!
```

(b)
```
header:
  seq: 1
  stamp:
    secs: 1500660269
    nsecs: 676667209
  frame_id: map
info:
  map_load_time:
    secs: 1500654640
    nsecs: 559560440
  resolution: 0.0500000007451
  width: 1187
  height: 296
  origin:
    position:
      x: -29.675
      y: -7.4
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
data: [...]     # An int8 array with size: 1187x296=351352
```

Rys. 2. Pola i format wiadomości ROS, które były używane w eksperymentach, (a) komunikat w postaci ciągu znaków, (b) komunikat w postaci mapy

Tabela 2. Wyniki eksperymentów z ciągiem znaków. Dla każdej linii publikowano i subskrybowano 600 tys. znaków o wielkości 27 bajtów (P/S)

	Częstotliwość publikacji (Hz)	Rzeczywista częstotliwość publikacji (Hz)	Utrata pakietu (absolutna/%)	Średnie opóźnienie P/S (ms)	Odchylenie standardowe opóźnienia P/S (ms)	Mediana opóźnienia P/S (ms)
ROS (C++)	1000	999,924	290 (0,048%)	0,141	0,044	0,144
	10 000	9990,432	868 (0,145%)	0,023	0,016	0,020
	30 000	29 951,33	122,659 (20,443%)	0,022	0,007	0,021
SROS (Python)	1000	999,955	85 (0,00014%)	0,131	0,047	0,117
	10 000	9985,279	37,068 (6,178%)	0,061	0,024	0,059
	30 000	20 898,611	358,330 (59,721%)	0,079	0,098	0,083
ROS-AES-Encryption (C++)	1000	999,992	98 (0,016%)	0,150	0,038	0,157
	10 000	9992,973	882 (0,147%)	0,025	0,012	0,023
	30 000	29 962,872	125 337 (20,889%)	0,023	0,006	0,022
Secure ROS (C++)	1000	999,998	54 (0.009%)	0,156	0,047	0,145
	10 000	9996,984	518 (0,086%)	0,023	0,010	0,019
	30 000	29 929,231	156 039 (26,006%)	0,022	0,012	0,021
Secure-ROS-Transport (Python)	1000	999,982	99 (0,016%)	0,167	0,028	0,179
	10 000	9987,499	752 (0,125%)	0,105	1,236	0,054
	30 000	29 683,69	7890 (1,315%)	0,071	0,420	0,066
Rosauth (HTML5/Javascript)	1000	249,929	76 (0,013%)	1,024	2,589	1,006

szybkościach publikacji (250 Hz, 2,5 kHz i 7,5 kHz). W dalszej części wiadomość „Hello World!" oznaczano po prostu jako *ciąg znaków*, a siatkę rozkładu pomieszczeń jako *mapę*. Na rysunku 2 przedstawiono przykłady ciągu znaków i komunikatu mapy opublikowanych podczas eksperymentów.

W przeprowadzonych eksperymentach częstotliwość publikowania została zdefiniowana w taki sposób, aby umożliwić analizę na trzech różnych poziomach: umiarkowanym, szybkim i przytłaczającym, w wyniku czego każda z propozycji została przetestowana do granic możliwości. Ze względu na to, że ROS nie jest systemem czasu rzeczywistego, nie można było zagwarantować docelowej częstotliwości publikowania, a także możliwe było wystąpienie utraty pakietów. We wszystkich eksperymentach zdefiniowano rozmiar kolejki jako 1 dla każdego publikującego i każdego subskrybenta. Przetestowano także oficjalne „niezabezpieczone" wydanie ROS Kinetic Kame, tak aby umożliwić porównanie opóźnień wynikłych z zastosowania różnych propozycji zapewnienia bezpieczeństwa.

W tabeli 2 przedstawiono ogólne wyniki eksperymentów z ciągiem znaków. Jak można zauważyć, większość podejść miała porównywalną wydajność z oficjalną wersją ROS do transmisji małych wiadomości złożonych z ciągu znaków, przy czym wartości opóźnień były bardzo podobne pomimo ogólnie lepszej wydajności systemu ROS bez jakiejkolwiek warstwy bezpieczeństwa. Można jednak zauważyć, że w SROS nie można było utrzymać zamierzonej częstotliwości publikowania wynoszącej 30 kHz, osiągając maksimum ≈ 21 kHz przy bardzo dużej utracie pakietów (59,72%).

Rosauth był oczywiście szczególnym przypadkiem, ponieważ jednostka publikująca nie była rodzimym węzłem ROS, lecz klientem www HTML5/Javascript. Ten klient łączył się z ROS za pośrednictwem *rosbridge* przy użyciu *websockets*, dostarczając komunikat JSON, który był parsowany po stronie ROS i następnie publikowany w sieci ROS. W pętli publikowania Javascript funkcja *setInterval* nakładała dolny limit 4 ms, co skutkowało maksymalną częstotliwością publikacji wynoszącą 250 Hz, jak pokazano w tabeli 2. Z tego powodu testy dla 10 i 30 kHz nie zostały wykonane dla *Rosauth*, ponieważ wyniki byłyby podobne do tych prezentowanych dla częstotliwości 1 kHz. Oczywiście powyższy mechanizm przesyłania wiadomości od nienatywnych klientów do ROS miał wpływ na opóźnienie publikacji/subskrypcji.

Na wykresach pudełkowych z rys. 3 przedstawiono opóźnienie w dostarczeniu wiadomości dla każdej z testowanych propozycji w eksperymencie z ciągiem znaków przy częstotliwości 1 kHz. Średnia wartość dla transmisji 600 tysięcy ciągów znaków została oznaczona czarną gwiazdką. Granice pudełka oraz linia w jego wnętrzu odpowiadają odpowiednio pierwszemu i trzeciemu kwartylowi oraz medianie wartości opóźnienia.

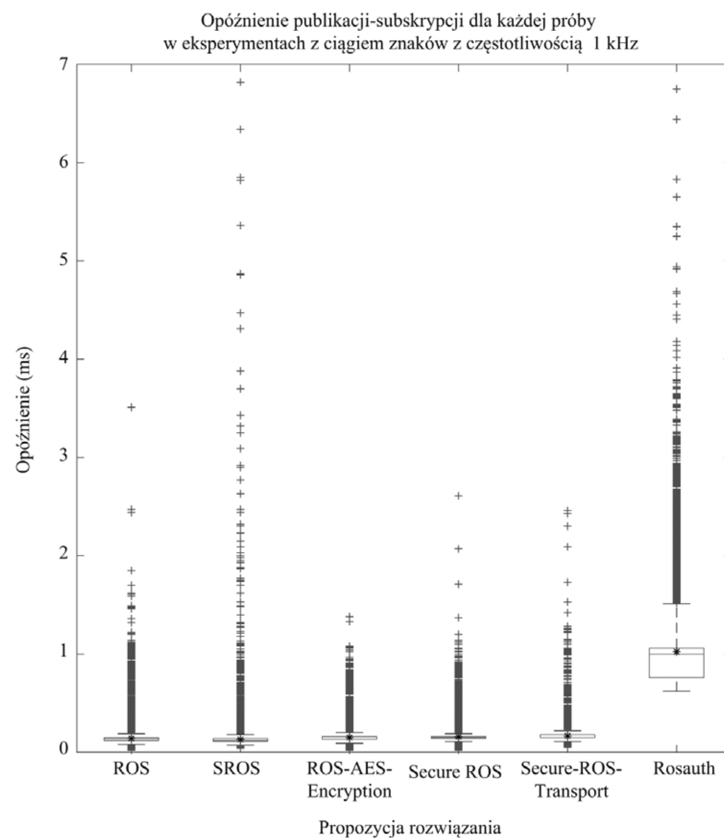
Analiza wykresów pudełkowych potwierdziła większe opóźnienia występujące w przypadku testu *rosauth* i podobne opóźnienia dla wszystkich innych propozycji podczas przesyłania mniejszych wiadomości. Ze względu na duży rozmiar każdego zestawu danych zaobserwowano pewne wartości odstające, szczególnie o wysokich wartościach ekstremalnych, co sugeruje, że czasami opóźnienia były znacznie większe, niż oczekiwano, być może z powodu wystąpienia szczytów obliczeniowych lub opóźnienia sieci.

Z drugiej strony, w tabeli 3 przedstawiono ogólne wyniki eksperymentów wykonanych podczas przesyłania mapy. Ze względu na duży rozmiar wiadomości wynoszący 343 kB wyniki były znacząco różne w porównaniu do eksperymentu z ciągiem znaków. Testy wykonane z ROS i z *Secure ROS* wyróżniały się spośród

pozostałych proponowanych rozwiązań, ponieważ w ich przypadku możliwe było zapewnienie zamierzonych częstotliwości publikowania oraz niskiego procentu utraty pakietów (< 0,2%). Potwierdza to, że *Secure ROS* jest jedną z najbardziej obiecujących propozycji zabezpieczenia ROS niewpływającą na obniżenie wydajności transmisji.

W przypadku wykorzystania algorytmu *szyfrowania ROS-AES* możliwe było publikowanie mapy tylko z częstotliwością wynoszącą około 186 Hz. Oczywiście szyfrowanie dużych bloków danych powodowało opóźnienia w szybkości publikowania. Interesujące było jednak to, że żaden pakiet nie został utracony podczas eksperymentów z *szyfrowaniem ROS-AES*, co oznacza, że krok szyfrowania i publikowania zawsze trwał dłużej niż etap subskrypcji i deszyfrowania. Średnie opóźnienie w dostarczaniu wiadomości do *szyfrowania ROS-AES* mieściło się w przedziale (5,5 ms, 5,8 ms), który był około 15–45 razy większy niż w przypadku zwykłej transmisji ROS.

W przypadku rozwiązań *SROS* i *Secure-ROS-Transport* obserwowano bardzo podobne wyniki. Oba podejścia osiągnęły limit częstotliwości publikacji przy około 80–82 Hz, przy utracie pakietów wynoszącej od około 48% do 53%. Średnie opóźnienie w dostarczaniu wiadomości mieściło się w przedziale (16,0 ms, 16,3 ms), który był od 44 do 129 razy większy niż w przypadku zwykłej transmisji ROS. Fakt, że testy obu rozwiązań zostały przeprowadzone na węzłach ROS napisanych przy użyciu biblioteki Pythona rospy (w przeciwieństwie do C ++ w przypadku ROS, Secure

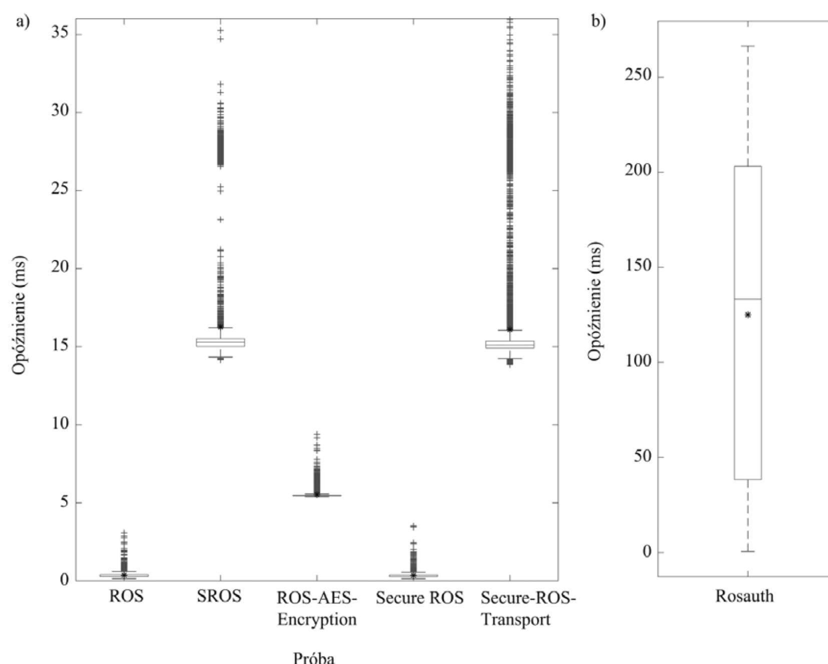


Rys. 3. Przegląd opóźnień P/S dla eksperymentów z ciągiem znaków przy 1 kHz

Tabela 3. Wyniki eksperymentów z mapami. Dla każdej linii publikowano i subskrybowano 150 tysięcy map, każda o wielkości 343 kB

	Częstotliwość publikacji (Hz)	Rzeczywista częstotliwość publikacji (Hz)	Utrata pakietu (absolutna/%)	Średnie opóźnienie P/S (ms)	Odchylenie standardowe opóźnienia P/S (ms)	Mediana opóźnienia P/S (ms)
ROS (C++)	250	250,0	0 (0,0%)	0,366	0,155	0,283
	2500	2499,776	71 (0,047%)	0,171	0,065	0,151
	7500	7496,412	235 (0,157%)	0,126	0,049	0,122
SROS (Python)	250	81,801	75 818 (50,545%)	16,248	3,594	15,286
	2500	81,316	74 376 (49,584%)	16,293	3,593	15,342
	7500	81,859	73 354 (48,903%)	16,173	3,499	15,263
ROS-AES-Encryption (C++)	250	186,923	0 (0,0%)	5,516	0,182	5,451
	2500	186,112	0 (0,0%)	5,544	0,194	5,468
	7500	178,87	0 (0,0%)	5,768	0,288	5,661
Secure ROS (C++)	250	250,0	1 (0,0006%)	0,351	0,143	0,277
	2500	2499,634	72 (0,048%)	0,172	0,065	0,154
	7500	7496,05	184 (0,123%)	0,126	0,041	0,122
Secure-ROS-Transport (Python)	250	80,308	74 855 (49,903%)	16,096	3,678	15,096
	2500	79,938	78 000 (52,0%)	16,286	3,899	15,090
	7500	80,275	79 547 (53,031%)	16,284	3,817	15,162
Rosauth (HTML5/ Javascript)	250	1,613	1 (0,0006%)	125 048,519	88 037,17	133 328,971

ROS i ROS-AES-Encryption) również mogło mieć wpływ na te otrzymane wyniki.



Rys. 4. Przegląd opóźnień P/S dla eksperymentów z mapami przy 250 Hz: a) opóźnienia publikacji/subskrypcji dla każdej próby w eksperymentach z przesyłaniem map z częstotliwością 250 Hz; b) opóźnienie publikacji/subskrypcji dla *Rosauth* w eksperymencie z przesyłaniem map z częstotliwością 250 Hz

Tabela 4. Dane zwracane przez każdą próbę na żądanie złożone w sieci ROS

	ROS	SROS	Szyfrowanie ROS-AES	Secure ROS	Secure-ROS-Transport	Rasauth
rostopic list	+	–	+	–	+	+
rosnode list	+	–	+	+	+	+
rosservice list	+	–	+	+	+	+
rosnode kill	+	–	+	–	+	+
rostopic echo	+	–	–	–	–	+

Podobnie jak poprzednio, w przypadku *Rosauth* obserwowano najniższą częstotliwość publikowania wynoszącą 1,6 Hz i wyjątkowo wysokie średnie opóźnienie wynoszące 125 sekund (od $340 \cdot 10^3$ do $990 \cdot 10^3$ razy większe niż w przypadku zwykłej transmisji ROS), co było wynikiem w przybliżeniu stałego wzrostu opóźnienia dostarczania wiadomości w czasie trwania eksperymentu, który rozpoczął się z opóźnieniami wynoszącymi 0,7 s, podczas gdy na jego zakończeniu opóźnienia wynosiły 266,2 s. Z tego samego powodu co poprzednio testy dla 2,5 kHz i 7,5 kHz zostały pominięte dla *Rosauth*.

Na wykresach pudełkowych przedstawionych na rysunku 4 zilustrowano opóźnienie w dostarczeniu wiadomości dla każdej z testowanych propozycji w eksperymencie z przesyłaniem map przy częstotliwości 250 Hz. Tak jak poprzednio, średnia wartość dla 150 tysięcy transmisji została oznaczona czarną gwiazdką. Opóźnienia dla *Rosauth* przedstawiono na osobnym wykresie ze względu na wyraźną różnicę w rzędzie wielkości.

Dla wszystkich inicjatyw, z wyjątkiem rozwiązania *Rosauth*, średnie opóźnienie było nieco większe niż mediany. Oznacza to, że wartości opóźnień były prawostronnie skośne, to znaczy większość wartości była niższa od średniej i w konsekwencji obserwowano wartości odstające większe od górnego kwartyla dla każdego wykresu pudełkowego. Na podstawie analizy wykresów pudełkowych można było wywnioskować, że najmniejsze opóźnienia przesyłania wiadomości obserwowano w przypadku *Secure ROS*, potem dla *szyfrowania ROS-AES* i następnie w przypadku *SROS* wraz z *Secure-ROS-Transport*.

Opóźnienia transmisji pakietów, utraty i maksymalne osiągalne częstotliwości publikowania nie były jedynymi istotnymi kwestiami w analizie propozycji zapewnienia bezpieczeństwa dla ROS. Przeprowadzono jakościową analizę aspektów bezpieczeństwa każdego rozwiązania, głównie sprawdzając stopień, w jakim dostęp do danych został uniemożliwiony przez nieupoważnione podmioty z dostępem do sieci ROS, w której przesyłane były wiadomości.

ROS udostępnia kilka narzędzi wiersza polecenia, które pozwalają anonimowo pobrać listę używanych tematów (*listarostopic*), listę używanych węzłów (*rosnode list*), listę używanych usług (*listarosservice*), umożliwiają zamykanie działających węzłów (*rosnode kill <node>*), wyświetlanie wiadomości przesyłanych w temacie (*rostopic echo <topic>*) i wiele innych. W związku z tym wykorzystano wyżej wymienione polecenia w sieci ROS i sprawdzono poziomy dostępu przyznawane przez każdą z propozycji zapewnienia bezpieczeństwa.

W tabeli 4 przedstawiono dane zwracane dla każdej z propozycji na żądanie złożone w sieci ROS przez nieautoryzowane węzły. W przypadku *SROS* można stwierdzić, że jest to podejście zapewniające wyższy poziom bezpieczeństwa, gdzie zapytania do nadrzędnego ROS z nieautoryzowanych węzłów pozostawały bez odpowiedzi. *Secure ROS* zapewniał także odpowiedni poziom bezpieczeństwa, nie pozwalając na nieupoważniony dostęp do listy i przeglądania jakichkolwiek wiadomości

w jakimkolwiek temacie ROS. Ponadto zabezpieczenie *Secure ROS* nie pozwalało na zamykanie węzłów. Zaskakujące było to, że z nieznanych przyczyn można było pobrać listę węzłów i usług używanych przez ROS. Należy również zauważyć, że SROS zapewnia konfigurację dostępu na poziomie węzła, natomiast *Secure ROS* zapewnia konfigurację dostępu na poziomie komputera (filtrowanie adresów IP). Z tego powodu każde wywołane zapytanie, na przykład przez SSH z maszyny, na której uruchomiona jest bezpieczna transmisja, zwróci dane w przypadku *Secure ROS*, ale już nie w przypadku użycia *SROS*.

W przeciwieństwie do *SROS* i *Secure ROS* inne rozwiązania nie stanowiły odpowiedniego zabezpieczenia nadrzędnego ROS, zapewniając w ten sposób nieodpowiednie poziomy autoryzacji. Wszystkie pozwalały na wyświetlanie tematów, węzłów i usług z poziomu ROS, a także na zamykanie węzłów bez autoryzacji. Jednakże wywołanie *rostopic echo* w przypadku rozwiązania *Secure-ROS-transport* nie dawało dostępu do wymienianych wiadomości, a w przypadku *szyfrowania ROS-AES* został wyświetlony niezrozumiały zaszyfrowany tekst.

W przypadku rozwiązania *Rosauth* zakłada się, że sieć ROS jest zaufana i zabezpieczane jest tylko połączenie pomiędzy nienatywnym klientem a interfejsem ROS, który następnie przekazuje niezabezpieczone wiadomości przez ROS. Tak więc, z punktu widzenia niniejszej analizy po stronie ROS, *Rosauth* działa tak, jak każda sieć ROS bez zabezpieczeń.

Ogólnie biorąc pod uwagę wszystkie przetestowane rozwiązania, *Secure ROS* i *SROS* są obecnie tymi, które mają największy potencjał do zwiększenia bezpieczeństwa ROS. *Secure ROS* zapewnia imponującą wydajność transmisji przy nieznacznym obciążeniu i w sposób zadowalający uniemożliwia dostęp do danych osobom nieupoważnionym. *SROS* jest bez wątpienia najbezpieczniejszą przetestowaną inicjatywą, zapewniającą zadowalającą wydajność transmisji o dużej przepustowości. Niemniej jednak, biorąc pod uwagę, że wciąż obie są w fazie rozwoju, to ciągle istnieje miejsce na ulepszenia. *Secure ROS* może zapobiegać nieautoryzowanemu dostępowi do listy węzłów i usług używanych przez ROS oraz umożliwiać autoryzację na poziomie węzłów, podczas gdy *SROS* może obsługiwać bibliotekę *roscpp* ROS C ++ oraz zapewniać łatwiejszą instalację i wdrażanie funkcjonalności.

Cyberbezpieczeństwo robotów należy rozwiązać na kilku różnych poziomach. W tej pracy skupiono się na bezpieczeństwie systemu operacyjnego robota (ROS). Oprócz zapewnienia bezpiecznej komunikacji komponentów ROS ważne jest również zabezpieczenie innych komponentów całego systemu robotycznego. Na przykład sieć, w której działają robot(y) powinna być nieprzenikalna, wykorzystując zabezpieczenia WPA2 + AES, ukrywanie SSID, filtrowanie adresów MAC, statyczne adresy IP i wszelkie inne szeroko udokumentowane środki bezpieczeństwa.

Jakikolwiek dostęp nienatywnych klientów ROS powinien korzystać z bezpiecznych połączeń i uwierzytelniania SSL/HTTPS w celu weryfikacji tożsamości klienta. Sieć ROS można wdrożyć w ramach sieci VPN, aby zachować bezpieczeństwo i prywatność w komunikacji sieciowej. Należy zdefiniować reguły zapory zezwalające na ruch tylko na określonych, a nie na domyślnych portach z określonych adresów IP, logowanie robota przez SSH powinno zostać wyłączone, powinno zostać wymuszone używanie silnych haseł uwierzytelniania na poziomie użytkownika, a także zapewnione szyfrowanie przechowywanych danych. Ponadto należy stosować metody starannej ochrony i wymiany kluczy kryptograficznych oraz utrzymania certyfikatów, obowiązkowych podpisów cyfrowych i poziomów dostępu, umożliwiających ich bezpieczne przechowywanie [37].

Podsumowanie i przyszłe perspektywy

Pomimo że ogólna poprawa cyberbezpieczeństwa robota nie jest prostym zadaniem, to ważne jest uwzględnienie od samego początku takich zaleceń, jak: bezpieczne cykle życia oprogramowania, szyfrowanie komunikacji robota, aktualizowanie oprogramowania, udzielanie dostępu tylko autoryzowanym użytkownikom, dostarczanie metod przywracania robota do bezpiecznego stanu fabrycznego, wdrażanie najlepszych praktyk w zakresie bezpieczeństwa cybernetycznego, edukowanie osób zajmujących się rozwojem robotów i kadry kierowniczej w zakresie cyberbezpieczeństwa, zapewnianie użytkownikom możliwości wyrażania opinii na temat potencjalnych luk w zabezpieczeniach oraz promowanie audytów bezpieczeństwa przed fazą produkcji. W tym celu niezbędne jest egzekwowanie wczesnych i zapobiegawczych zasad bezpiecznego projektowania dla aplikacji robotów.

Celem tego rozdziału była identyfikacja potencjalnych zagrożeń dla bezpieczeństwa i prywatności w powszechnie stosowanym ROS, podnosząc w ten sposób świadomość na temat cyberbezpieczeństwa robotów i potrzebę dopracowania branżowych zasad bezpieczeństwa, tak aby uniknąć konsekwentnego wprowadzenia niepewnych robotów na rynek.

Oprócz dogłębnej analizy literatury dotyczącej bezpieczeństwa danych w robotyce ujawniono kilka błędów bezpieczeństwa w powszechnie przyjętym rozwiązaniu ROS oraz przedstawiono analizę propozycji mających na celu zabezpieczenie aplikacji robotów bazujących na ROS. Podano również ogólne zalecenia i środki bezpieczeństwa na różnych poziomach do kierowania wdrażaniem i rozlokowaniem systemów bazujących na jednym lub wielu robotach. W przyszłości planowane jest opracowanie komercyjnie użytecznego systemu opartego na ROS dla wielu robotów, który byłby przeznaczony do monitorowania infrastruktury obejmującej kluczowe środki bezpieczeństwa cybernetycznego i prywatności, w ramach trwającego projektu R&D STOP. ■

Bibliografia dostępna pod linkiem:
nis.com.pl/bibliografia.html

Fragment pochodzi z książki:
Sztuczna inteligencja. Bezpieczeństwo i zabezpieczenia,
Roman V. Yampolskiy (redakcja).
Wydawnictwo Naukowe PWN,
Warszawa 2020