Bezpieczeństwo aplikacji robotów z wykorzystaniem ROS

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

Wprowadzenie

Roboty stają się częścią naszego codziennego życia w społeczeństwie. Oprócz szerokiego zastosowania w przemyśle, obecnie roboty odgrywają ważną rolę w zastosowaniach domowych, medycznych, kosmicznych i wojskowych. Ostatnie badania zaprezentowały znaczące innowacje również w innych dziedzinach, takich jak robotyka społeczna, robotyka polowa czy inteligentne pojazdy. Powszechne zastosowanie robotów w tych dziedzinach zostało znacznie przyspieszone ze względu na ostatnie postępy w dziedzinie kontroli, sztucznej inteligencji (SI), podejmowania decyzji, uczenia się robotów, lokalizowania i mapowania, planowania ruchu, wizji komputerowej, rozwoju czujników i innych istotnych dziedzin [1].

Oczywiście wraz z rosnącą liczbą robotów w naszym społeczeństwie rodzi się obawa o bezpieczeństwo. Jest to jednak często pomijany problem w systemach robotycznych, ponieważ nacisk kładziony jest na funkcjonalność i innowacje robotów. Nieautoryzowany dostęp do robota lub sieci używanej przez roboty może poważnie zagrozić systemowi, potencjalnie prowadząc do niedopuszczalnych konsekwencji, takich jak narażenie ludzi, którzy dzielą środowisko z robotem lub robotami [2].

Potencjalne zalety robotyki są jasne i szeroko udokumentowane. Wprowadzają jednak nowe obawy dotyczące bezpieczeństwa i prywatności. Systemy zrobotyzowane są budowane na tradycyjnych platformach komputerowych, połączonych z siłownikami oraz innymi czujnikami i sprzętem, takim jak kamery. W momencie gdy systemy takie zostają połączone w sieć i łączą się z Internetem, nie tylko stają się narażone na te same cyberataki co tradycyjne systemy komputerowe, ale także ujawniają zupełnie nowy zestaw problemów bezpieczeństwa związanych z obawami o prywatność w przypadku zhakowania lub, co gorsza, ze spowodowaniem krzywdy fizycznej.

Masowe wykorzystanie robotów prawdopodobnie zwiększy możliwości takich ataków. Jest to szczególnie problematyczne w obronie, medycynie i innych krytycznych dziedzinach dotyczących ludzi [2]. Jednakże uwzględnienie zagrożeń dla bezpieczeństwa i prywatności najwyraźniej nie było priorytetem dla twórców robotów wywodzących się zarówno ze środowisk akademickich, jak i z przemysłu. Wraz z tym nowym typem zagrożenia, wprowadzonym przez roboty mające dostęp do sieci, obecny moment jest idealny na dokładne przyjrzenie się aktualnym praktykom bezpieczeństwa, zanim roboty z poważnymi wadami staną się wszechobecne [3].

Komisja Europejska (KE) na bieżąco śledzi kwestie prawne i etyczne związane z nowymi technologiami i niedawno zaproponowała szereg zaleceń w sprawie przepisów prawa cywilnego dotyczącego robotyki i SI [4]. W trosce o bezpieczeństwo i prywatność KE opracowała kodeks etycznego postępowania dla badaczy i twórców technologii robotycznych. Kodeks jest dobrowolny i służy jako wytyczne dla osób zaangażowanych w rozwój i wykorzystanie robotyki i technologii SI w celu zapewnienia zgodności z ustalonymi standardami. Zaproponowano także utworzenie europejskiej agencji robotyki i SI, której głównym celem byłoby dostarczanie władzom publicznym informacji dotyczących kwestii technicznych, etycznych oraz regulacyjnych w tych dziedzinach.

Pomimo ostatnich postępów w finansowanych przez KE projektach z zakresu badań i rozwoju robotyki często pomija się kwestie bezpieczeństwa i prywatności. Jest to również powszechna praktyka w badaniach dotyczących robotyki, a nawet w dostępnych komercyjnych rozwiązaniach robotycznych (porównaj część „Kontekst"). Z tego powodu tworzenie bezpiecznych aplikacji robotów w środowiskach programowania ogólnego przeznaczenia jest konieczne. Jest to jeden z motywów trwającego projektu STOP R&D[1], którego celem jest wdrożenie komercyjnego systemu bezpieczeństwa rozproszonych i współpracujących robotów do roku 2020. Pomimo ukierunkowanej przełomowej aplikacji w świecie rzeczywistym zespół składający się z podmiotów publicznych i prywatnych zajmuje się znalezieniem rozwiązania typowych błędów popełnianych podczas projektowania złożonych systemów, takich jak rozwiązania wykorzystujące dużą liczbę robotów [5].

W ostatnich dziesięcioleciach opracowano wiele różnych struktur robotów, jednak bez wątpienia najbardziej popularnym jest *Robot Operating System* (ROS)[2] [6]. Szereg funkcjonalności tego systemu doprowadziło do wprowadzenia ROS na całym świecie, które stało się niemal standardowym oprogramowaniem pośredniczącym, tak wyczekiwanym przez społeczność zajmującą się rozwojem robotów. A zatem kilka robotów od dostawców i instytutów badawczych obsługuje ROS. W większości przypadków korzystanie z robotów podłączonych do Internetu pozostaje ograniczone tylko do laboratoriów badawczych, jednak ich szersze zastosowanie w przyszłości wydaje się nieuniknione [7]. W tym rozdziale omówiono zagadnienie bezpieczeństwa związane z aplikacją robotów, które wykorzystają ROS. Opierając się na wnioskach wyciągniętych z projektu STOP, dokonano przeglądu i omówiono kilka inicjatyw mających na celu

zabezpieczenie ROS, a także ogólne środki bezpieczeństwa, które należy wdrożyć, aby uniknąć groźnych dla prywatności i bezpieczeństwa konsekwencji.

W kolejnej części tego rozdziału omówiono przełomowe prace dotyczące promowania bezpieczeństwa i prywatności w robotyce, natomiast w części „Obawy dotyczące bezpieczeństwa ROS" zidentyfikowano znane problemy z zakresu bezpieczeństwa ROS. Następnie, w części „Koncepcje dotyczące zabezpieczania aplikacji robotów bazujących na ROS", przedstawiono i omówiono kilka ostatnich koncepcji, które podnoszą poziom bezpieczeństwa aplikacji ROS. Dodatkowo opisano ogólne badania i zalecenia mające na celu ochronę danych w projektach robotycznych, a na koniec rozdziału przedstawiono wnioski i perspektywy przyszłych prac.

Kontekst

Dyskusja na temat bezpieczeństwa i ochrony robotów sięga „Trzech zasad robotyki" Asimova z 1950 roku [8], które stwierdzają, że:

1. Robot nie może zranić człowieka ani w wyniku bezczynności pozwolić, by człowiekowi stała się krzywda.
2. Robot musi być posłuszny rozkazom wydawanym przez ludzi z wyjątkiem przypadków, gdy takie rozkazy byłyby sprzeczne z pierwszym prawem.
3. Robot musi chronić swoje istnienie, o ile taka ochrona nie jest sprzeczna z pierwszym lub drugim prawem.

Jednak wraz z postępem w robotyce naukowcy wykazali, że same te prawa nie są wystarczające do zarządzania zachowaniem robota [3].

Termin *Cryptobotics* został zaproponowany przez Morante'a i innych [2] jako jednolity termin na badania i zastosowania środków bezpieczeństwa komputerów i mikrokontrolerów w robotyce. W swojej pracy autorzy podkreślają potrzebę implementacji szyfrowanej komunikacji i analizowania wpływu szyfrowania na wydajność w czasie rzeczywistym, podnosząc w ten sposób świadomość programistów na temat opłacalności integrowania tych mechanizmów w zależności od konkretnego przypadku. Finnicum i King [9] scharakteryzowali kluczowe czynniki poprawiające bezpieczeństwo i prywatność w aplikacjach robotów, takie jak identyfikacja użytkownika, prawidłowe ujawnianie uprawnień aplikacji i kontrola nad własną prywatnością. Ponadto autorzy zaproponowali warstwową architekturę oprogramowania opartą na jądrze bezpieczeństwa dla aplikacji robotów. Co więcej, Adi [10] skoncentrował się na wymaganiach bezpieczeństwa w zakresie projektowania technologii identyfikacji robotów na zasadzie analogii do ludzkiego społeczeństwa, umożliwiając w ten sposób bezpieczne transakcje między robotami o unikatowej, weryfikowalnej tożsamości.

W pracy [11] opisano ulepszenie bezpieczeństwa *Interoperable Telesurgery Protocol* (ITP). Odnosi się to do czterech kluczowych aspektów bezpieczeństwa: komunikacji, uwierzytelniania, autoryzacji oraz opracowywania i egzekwowania polityki bezpieczeństwa, opartych na opublikowanych standardach dotyczących rygorystycznych wymagań robotyki telechirurgicznej. Pomimo zapewnienia prywatności i integralności informacji w poszczególnych kanałach komunikacyjnych komunikacja pomiędzy telechirurgiczną jednostką *master* i jednostką *slave* wymaga bardzo solidnego, redundantnego i bezpiecznego łącza komunikacyjnego. Autorzy twierdzą, że protokół spełniający te wymagania wciąż nie istnieje.

Z drugiej strony Yong i inni [12] zidentyfikowali zagrożenia dla rodzin z dziećmi podczas korzystania z robotów-zabawek sterowanych bezprzewodowo i zaproponowali rozwiązania różnego poziomu zmniejszające ryzyko firmom zabawkarskim oraz konsumentom. Należą do nich: (I) strategie ochrony VoIP (kontrola rodzicielska i anonimowość), (II) bezpieczne połączenia zdalnego sterowania (szyfrowanie i uwierzytelnianie), (III) połączenia bezprzewodowe (funkcje bezpieczeństwa domowej sieci Wi-Fi), (IV) strategie kontroli rodzicielskiej wbudowane w komercyjne zabawki–roboty, (V) strategie ochrony kamer (efekty dźwiękowe, uwierzytelnianie), (VI) wspólna perspektywa ryzyka (odpowiedzialność rządu i przedsiębiorstwa zabawkarskiego, wymagane są prawne zobowiązania w celu ochrony bezpieczeństwa konsumentów), (VII) ogólne strategie ochrony (silne hasła, wymagana wstępna konfiguracja połączenia przewodowego, unikanie trybu *ad hoc*, używanie zaktualizowanego firmware'u, unikanie domyślnych portów itp.).

Istnieją również obawy związane z bezpieczeństwem sieci wieloagentowych. Na przykład Caiti i inni [13] opisali metodologię bezpiecznej współpracy w sieci autonomicznych mobilnych podwodnych czujników połączonych siecią akustyczną. Zaproponowany algorytm jest z natury niezawodny: wraz z utratą komunikacji pomiędzy pojazdami stopień pokrycia, czyli cel misji, ulega pogorszeniu, ale nie zostaje utracony. Proponowana forma wdzięcznej degradacji stanowi środek reakcji przeciwko atakowi *Denial of Service* (DoS). Aby zapewnić wiarygodność dostępnych informacji, zaprojektowano pakiet bezpieczeństwa oparty na paradygmacie komunikacji grupowej, w celu zminimalizowania wymiany informacji pomiędzy pojazdami i zmniejszenia narzutu związanego z komunikacją pod względem liczby i rozmiaru wiadomości. Autorzy przedstawili dwie główne usługi bezpieczeństwa danych: *Secure Dispatching Service* (SDS) i *Key Management Service* (KMS). SDS dotyczy ochrony poufności i autentyczności wiadomości przez ich szyfrowanie i deszyfrowanie, a także generowania i weryfikacji dowodów ich autentyczności. KMS jest związany z cofaniem bieżącego klucza i dystrybucją nowego klucza, okresowo lub w momencie wyjazdu pojazdu.

Co więcej, wyzwania bezpieczeństwa w robotyce roju zostały zidentyfikowane w pracy [14]. Obejmują one: (I) ograniczenia zasobów w zakresie przechowywania, przepustowości komunikacji, ograniczenia obliczeniowe i – co najważniejsze – ograniczenia energii ze względu na małe rozmiary urządzeń, (II) fizyczne przechwytywanie i manipulowanie robotami, które mogą wpływać na zachowanie roju, (III) trudną kontrolę podejścia do roju z powodu braku hierarchicznych punktów kontroli i nieodłącznego rozproszonego podejmowania decyzji, (IV) stosowanie różnego rodzaju jawnej i dorozumianej komunikacji, która może być zagłuszana, przechwytywana lub w inny sposób zakłócana, (V) mobilność fizyczną, która ma wpływ na uwierzytelnianie jednostki i dodatkowe kwestie

bezpieczeństwa, (VI) tożsamość w roju w celu zagwarantowania i potwierdzenia pochodzenia danych, legalnej komunikacji, poufności, integralności i dostępności, (VII) zarządzanie kluczami kryptograficznymi ze względu na dynamiczny i interaktywny charakter roju, (VIII) wykrywanie wtargnięcia ze względu na autonomiczny charakter robotów i zbiorowe zachowanie w sytuacjach awaryjnych, (IX) zarządzanie uczeniem się przez adaptację ze względu na wprowadzanie zmian przez złośliwe jednostki powodujące niepożądane dostosowywanie robotów. Wyzwania zidentyfikowane przez autorów, pomimo że dotyczące robotyki roju, przenoszą się do większości rozproszonych sieci wieloagentowych. Z tego powodu wczesne zajęcie się takimi problemami mogłoby zapobiec niepożądanym konsekwencjom dla wielu zastosowań tego typu technologii.

W innym kontekście w pracy [15] przedstawiono informacje o licznych lukach, z którymi borykają się użytkownicy inteligentnych urządzeń gospodarstwa domowego, pokazując tym samym, w jaki sposób narzędzia do monitorowania sieci i regularne techniki ataku DDoS mogą być łatwo wykorzystane do ataku na sieć Internetu Rzeczy (IoT, ang. *Internet of Things*) z zamiarem osiągnięcia nielegalnych celów. Dotychczas wdrożono obiecujące anonimowe podejście do komunikacji oparte na TOR[3], tak aby pomóc użytkownikom inteligentnego domu chronić ich prywatność i zwiększyć bezpieczeństwo systemu inteligentnego sprzętu gospodarstwa domowego przed cyberatakami. Wyniki sugerują, że jest to odpowiednie podejście do rozwiązywania ostatnich problemów bezpieczeństwa związanych z inteligentnymi urządzeniami domowymi, które zostały oparte na TCP.

W literaturze przedstawiono także kilka prac identyfikujących luki w architekturze robotów. Naukowcy z University of Washington wykazali możliwość złośliwego sterowania szeroką gamą funkcji robota Raven II wykorzystywanego do zdalnego przeprowadzania operacji chirurgicznych, całkowicie ignorując lub zastępując polecenia wydawane przez chirurga. Odkryli również, że do wykonywania ataków może być także wykorzystywany mechanizm zatrzymania awaryjnego [16]. W podobnym badaniu zhakowano quadkopter AR.Drone 2.0, odsłaniając oczywiste luki w zabezpieczeniach [17]. Przedstawiono kilka scenariuszy ataku pokazujących, w jaki sposób złośliwy atakujący może uzyskać pełną kontrolę nad quadkopterem. Inne badania wskazały dodatkowe, komercyjnie dostępne platformy, w których występowały krytyczne problemy z cyberbezpieczeństwem [3, 7]. Oprócz zaproponowania w swoich pracach kilku środków zaradczych, wszyscy autorzy stwierdzili, że większości tych ataków można łatwo zapobiec, stosując dobrze ugruntowane i łatwo dostępne mechanizmy bezpieczeństwa, w tym szyfrowanie i uwierzytelnianie.

Wszystkie omówione wyzwania i luki w zabezpieczeniach są przesłankami do tego, że obecni i przyszli właściciele robotów muszą stanąć wobec istotnych wyzwań związanych z bezpieczeństwem, które są bezpośrednią konsekwencją tego, że producenci priorytetowo traktują czas wprowadzenia produktu na rynek, a badacze przyznają większy priorytet funkcjonalności i innowacji niż wdrożeniu i testowaniu bezpieczeństwa. Metodologie projektowania wrażliwego na wartości (VSD, ang. *Value-sensitive design*) [18], które uwzględniają kluczowe wartości ludzkie, są odpowiednim przykładem metod, które dostawcy i badacze powinni uwzględnić na etapie projektowania systemów robotyki i systemów sieciowych.

Warto jednak zauważyć, że pomimo ogólnego braku koncentracji na cyberbezpieczeństwie robotów, robot teleobecności BeamPro jest godnym uwagi wyjątkiem [2]. BeamPro korzysta z bezpiecznych protokołów szyfrowania symetrycznego i uwierzytelniania danych. Niemniej jednak, pomimo istnienia sterowników ROS dla BeamPro, technologia ta jest zastrzeżona, a jej replikacja jest ograniczona do innych produktów sprzedawanych przez tę samą firmę. W tym rozdziale skupiono się na standardzie *de facto* w robotyce i ROS, z zamiarem znacznej poprawy wydajności i bezpieczeństwa w tworzeniu oprogramowania dla robotów nie tylko w badaniach, ale także w przypadku start-upów związanych z robotami i większych firm branży robotycznej.

Obawy dotyczące bezpieczeństwa ROS

ROS jest bardzo popularnym oprogramowaniem pośredniczącym dla robotyki, którego głównymi zastosowaniami są abstrakcja sprzętu, kontrola urządzeń na niskim poziomie, wdrażanie często używanych funkcjonalności, przekazywanie komunikatów pomiędzy procesami i zarządzanie pakietami [6]. ROS promuje ponowne użycie kodu na innym sprzęcie, zapewniając dużą liczbę dostępnych dla społeczności bibliotek, takich jak laserowe SLAM [19], rozpoznawanie obiektów na podstawie chmury punktów 3D [20], a także narzędzia do wizualizacji, nagrywania eksperymentów i wiele innych. ROS został oparty na systemie publikowania – subskrybowania i przekazywania wiadomości, które wykorzystuje protokół XML-RPC (ang. *Remote Procedure Call*). Umożliwia to rodzimym klientom z wielu platform i języków wysyłanie i odbieranie danych w trybie *peer-to-peer*.

Pomimo wyraźnych zalet integracji robotów w ROS nie istnieje żadna domyślna funkcja zapewnienia bezpieczeństwa, co czyni roboty podatnymi na złośliwe ataki. Jest to w rzeczywistości jeden z powodów, dla których ROS jest preferowany w badaniach i nie został jeszcze w pełni wprowadzony do zastosowań przemysłowych. W tej części przeanalizowano znane problemy bezpieczeństwa związane z ROS.

Komunikacja między węzłami w ROS wykorzystuje *czysty tekst* przesyłany przez TCP/IP i UDP/IP. ROS jest odpowiedzialny tylko za sprawdzenie sumy MD5 struktury wiadomości, tak aby zagwarantować, że strony uzgodnią układ wiadomości. Nieszyfrowany tekst ma zalety związane z łatwością użycia, debugowania i wydajnością. Pozwala to jednak nieautoryzowanemu odbiorcy łatwo przechwycić i zinterpretować formę wiadomości, zebrać informacje i przesłać fałszywe wiadomości do systemu [21].

Modułowość ROS została wysoko oceniona w społeczności. Ma to jednak również wady, takie jak *ujawnienie portów TCP*, które nie zapewniają uwierzytelnienia. Korzystanie z niezabezpieczonych i niechronionych portów TCP oraz *brak mechanizmów uwierzytelniania* może stwarzać możliwości

dla złośliwych nadawców wiadomości, którzy mogą manipulować systemem, wstrzykiwać wiadomości (*person-in--the-middle*) i zastępować istniejących nadawców i odbiorców, kierując w ten sposób zewnętrzne pakiety w kierunku portów ROS.

Ponadto ROS ma anonimową semantykę publikowania i subskrybowania. Z tego powodu ogólne węzły nie są świadome tego, z kim się komunikują [2]. System wykorzystuje *słabe schematy autoryzacji* bez weryfikacji nadawcy, bez sprawdzania integralności i autentyczności danych oraz bez definicji poziomów dostępu. Niektórzy zdalni klienci nie powinni mieć dostępu do całego systemu ROS, ponieważ umożliwia to użytkownikom wysyłanie bezpośrednich poleceń do robotów, które w ten sposób mogą omijać progi bezpieczeństwa. Ze względu na to, że ROS nie zapewnia pomiaru rozdzielczości konfliktu, różne węzły mogą jednocześnie wstrzykiwać szybkie polecenia do bazy mobilnej, sterować siłownikiem, manipulatorem i tak dalej.

Jeśli chodzi o komunikację, ROS *wymaga dwukierunkowej sieci* pomiędzy wszystkimi komputerami. Z tego powodu ustawienia zapory w używanych węzłach muszą być mniej surowe, stwarzając dodatkowe zagrożenie bezpieczeństwa i powodując dodatkowe obciążenie sieci związane z wymaganiami obliczeniowymi i opóźnieniami.

ROS również nie wspiera *jakości usług* (QoS). Każdy węzeł odpowiada za zarządzanie własną komunikacją, podobne komunikaty nie są kompaktowane ani nie podejmuje się żadnych wysiłków w celu ograniczenia komunikacji sieciowej. Wiadomości tego samego typu mogłyby zostać potencjalnie skompresowane, ponieważ duża ilość komunikacji utrudnia tworzenie rozwiązań w przypadkach, gdy czas ma krytyczne znaczenie.

Nie przeprowadzono żadnych badań dotyczących wydajności ROS z dużą liczbą węzłów. Jednak, ze względu na scentralizowanie w węźle głównym ROS, usługa nazewnictwa ROS prowadzi do problemów ze *skalowalnością* [22], ponieważ nie została zaprojektowana do obsługi dużej liczby żądań. Efektem tego jest powstanie specyficznego wąskiego gardła, ponieważ system nie jest w stanie wygenerować odpowiedzi w rozsądnym czasie, stając się tym samym narażonym na ataki DoS. Ponadto z powodu całkowitego braku zabezpieczeń nie jest jasne, czy przeprowadzono analizę w celu wyszukania innych luk w zabezpieczeniach, takich jak przepełnienie bufora lub możliwości zdalnego wykonania kodu w systemie ROS.

Według [23] dostępność kodu źródłowego ROS pozwala na dostosowanie go do podejścia opartego na wielu wzorcach, jak na przykład przedstawiono to w [24]. Dzięki tej modyfikacji ROS może obsługiwać większą liczbę urządzeń i poprawiać jego użyteczność w bardziej złożonych instalacjach inteligentnych środowisk, bezprzewodowych sieci czujników i/lub systemów z wieloma robotami. Zidentyfikowano ograniczenie każdego węzła nadrzędnego ROS do utrzymywania pełnej przestrzeni nazw w swojej pamięci oraz przedstawiono system stosowania reguł do przestrzeni nazw oraz zmniejszania zużycia pamięci i przepustowości. Ponadto autor opisał obsługę protokołu IPv6 w ROS[4] i wykazał, że „gadatliwość" i złożoność stosowanego protokołu XML-RPC stanowi duże obciążenie dla węzłów rozproszonych w sieci. Z tego powodu rozszerzenie ROS o obsługę często używanych formatów danych, takich jak JSON lub bufory protokołów sparowane z HTTP, może zwiększyć jego zasięg.

Po uruchomieniu węzła głównego ROS otwierany jest port, do którego może się podłączyć dowolny komputer sieciowy. Z kolei maszyny te mogą następnie wysłać zapytanie do systemu głównego ROS w celu wykonania kluczowych zadań, takich jak konfigurowanie połączeń ROS TCP/UDP z innymi węzłami ROS, subskrybowanie dowolnego tematu, zamykanie dowolnego węzła w sieci itd. W rezultacie złośliwa jednostka, znajdująca się w tej samej sieci co robot z systemem ROS, najprawdopodobniej będzie miała dostęp do wszystkich danych robota i zarazem wiele możliwości ich wykorzystania, takich jak wysyłanie poleceń, blokowanie połączeń sieciowych, dostęp do strumieni kamer wideo i do ogólnego powodowania niepożądanych wydarzeń.

Ostatnio doniesiono o ważnych badaniach mających na celu rozwiązanie problemu niektórych ograniczeń bezpieczeństwa ROS. Na przykład w [25] położono nacisk na uwierzytelnianie i autoryzację (AA) w systemach ROS 1.x. Przed transmisją i odbieraniem danych jednostki systemowe muszą się uwierzytelnić przy użyciu pary login i hasło, natomiast w węźle autoryzującym następuje sprawdzenie danych uwierzytelniających, ról i związanych z nimi uprawnień. W węźle AA generowane są wszystkie klucze komunikacyjne, tak aby zagwarantować, że komunikacja jest zawsze analizowana niezależnie od tego, czy pochodzi z zaufanego źródła czy nie. W pracy [26] autor przeanalizował również luki ROS i wynikające z nich zagrożenia, które mogą być wykorzystane przez atakujących, i zaproponował standaryzację formatów dzienników bezpieczeństwa, składnię profilowania dla polityk bezpieczeństwa oraz przedstawił nowe narzędzia do introspekcji zarejestrowanych dzienników bezpieczeństwa. W pracy [27] zaproponowano struktury weryfikacji środowiska wykonawczego dla aplikacji robotów opartych na ROS. Zapewniają one sposób ciągłej obserwacji wszystkich żądań, komunikatów i wiadomości przez dodanie narzędzia monitorującego do połączenia, przy użyciu techniki *man-in-the-middle*. Narzędzie takie służy do wykrywania potencjalnie niechcianych wiadomości, egzekwując zasady kontroli dostępu, takie jak umożliwianie tylko niektórym węzłom ROS publikowania wiadomości na określony temat. Ponadto w 2015 roku rozpoczęto opracowywanie podstawowej struktury oprogramowania środowiska ROS 2.0, a wersja alfa przeznaczona do testów społecznościowych została już udostępniona w momencie pisania tego rozdziału. Poświęcono wiele wysiłków na rzecz zintegrowania DDS (ang. *Data Distribution Service*), standardowej specyfikacji dla komunikacji publikacja – subskrypcja w systemach czasu rzeczywistego i systemach wbudowanych, jako warstwy transportowej dla ROS 2.0. Zgodność ze specyfikacją DDS Security zapewnia uwierzytelnianie, kontrolę dostępu i szyfrowanie danych. Ponadto oczekuje się, że ROS 2.0 zapewni dodatkowe bezpieczeństwo, od razu obsługując IPv6, dzięki czemu

skanowanie hosta i identyfikacja będą trudniejsze dla atakujących.

Propozycje dotyczące zabezpieczania aplikacji robotów bazujących na ROS

Społeczność zajmująca się rozwojem robotów musi poważnie zająć się problemami związanymi z cyberbezpieczeństwem robotów. Oczekuje się, że w dziedzinie SI i robotyki pojawią się problemy bezpieczeństwa podobne do tych, które wynikały z rewolucji komputerowej w czasach boomu internetowego [2]. Oprócz ataków, z którymi obecnie zmagają się systemy komputerowe, na przykład DoS, podsłuchiwanie, fałszowanie, manipulowanie, eskalacja uprawnień, ujawnianie informacji itp., w przypadku robotów występuje dodatkowo czynnik interakcji fizycznej, zwiększający ich podatność na ataki. W związku z tym nieautoryzowany dostęp do robota może mieć katastrofalne konsekwencje, takie jak wyłączenie go, spowodowanie jego nieprawidłowego działania, uszkodzenie samego siebie, uszkodzenie otoczenia lub, co gorsza, zranienie kogoś w pobliżu.

Co więcej, eksperymenty pokazały, że w robotyce szczególnie trudne może być odróżnienie exploita cyberfizycznego od błędu sprzętowego lub programowego [21]. Z tego powodu wykrywanie złośliwych włamań może być trudne, a podczas gdy system będzie faktycznie atakowany, exploity mogą zostać zamaskowane jako proste błędy.

ROS jest wiodącym oprogramowaniem pośredniczącym w dziedzinie badań związanych z robotyką, korzysta z ogromnego wsparcia społeczności, regularnych wydań oprogramowania, powszechnego zastosowania w środowisku akademickim i niektórych sektorach przemysłu, od małych wbudowanych urządzeń po roboty usługowe na dużą skalę. Jednak w systemie opartym na ROS atakujący może łatwo utworzyć węzeł, wysłać zapytanie do mastera o stan systemu i następnie wysłać polecenia w celu zamknięcia dowolnego węzła lub opublikować zafałszowane wiadomości na ważne tematy, na przykład błędne dane czujnika. Co więcej, węzły są jednoznacznie identyfikowane po nazwie, a nowo utworzone węzły zastępują istniejące węzły o tej samej nazwie. Z tego powodu osoba atakująca może łatwo sfałszować węzeł, tak aby opublikować fałszywe wiadomości na ważne tematy. Na przykład węzeł nawigacyjny w robocie może zostać zamknięty i zastąpiony fałszywym węzłem, który źle pokieruje robota.

Poniżej omówiono pięć różnych ostatnich propozycji, mających na celu zabezpieczenia ROS. W następnej części omówiono rezultaty przeprowadzonych eksperymentów, których zamierzeniem było potwierdzenie i porównanie wydajności każdej z tych propozycji. Skoncentrowano się głównie na kosztach komunikacji tych propozycji, porównując je z niezabezpieczonym systemem ROS.

SROS

SROS, które oznacza Zabezpieczanie ROS (ang. *Securing ROS*), zostało zaproponowane jako dodatek do ROS API oraz jako ekosystem do obsługi nowoczesnej kryptografii i środków bezpieczeństwa w celu usunięcia istniejących luk [28]. Pomimo że nadal jest on wysoce eksperymentalny i jest w trakcie intensywnego rozwoju[5], to zdaniem autorów SROS obsługuje „natywny Transport Layer Security (TLS) dla całego transportu gniazd w ramach ROS, użycie certyfikatów x.509 zezwala na łańcuch zaufania, definiowalne globalizacje przestrzeni nazw dla ograniczeń węzłów ROS oraz dozwolone role, a także zapewnia wygodne narzędzia przestrzeni użytkownika do automatycznego generowania par kluczy dla węzłów, audytu sieci ROS oraz konstruowania/uczenia zasad kontroli dostępu".

W SROS korzystanie z TLS pomiędzy dwiema komunikującymi się aplikacjami sprzyja uzyskaniu prywatności, uwierzytelnionej tożsamości i integralności danych. Od momentu uruchomienia SROS zapewnia niezależny od ROS serwer kluczy do generowania i dystrybucji kluczy oraz certyfikatów do węzłów ROS. Serwer kluczy upraszcza korzystanie i rozwój systemów obsługujących SROS dla użytkowników końcowych, płynnie integrując się ze SROS, generując i dystrybuując elementy infrastruktury klucza publicznego (PKI), w tym: klucze asymetryczne, urzędy certyfikacji (CA) oraz podpisane certyfikaty, zapewniając zachowawcze domyślne konfiguracje bezpieczeństwa.

W czasie pisania tego rozdziału SROS nie obsługiwał biblioteki ROS C++ (*roscpp*), obejmując tylko węzły zakodowane w Pythonie (*rospy*). W celu uruchomienia SROS konieczne było zainstalowanie go ze źródła i wprowadzenie kilku wstępnych konfiguracji, a mianowicie pozyskanie konfiguracji SROS, uruchomienie trybu uczenia SROS, tak aby niezbędne parametry tematów/usług mogły zostać nauczone podczas sesji ładowania oraz została dokonana zmiana niektórych z konfiguracji wygenerowanych przez serwer kluczy.

Szyfrowanie ROS-AES

Ostatnie badanie [29] wykazało, że stosowanie szyfrowanej komunikacji w ROS zapewnia minimalny narzut wydajności procesora i obciążenia komunikacyjnego dla systemów bez twardych ograniczeń w czasie rzeczywistym. Opisane prace polegały na szyfrowaniu danych przesyłanych pomiędzy procesami ROS przy użyciu algorytmu 3DES przez dodawanie pary węzłów ROS do zadań szyfrowania i deszyfrowania, bez zmieniania struktury komunikatów ROS i standardowych funkcjonalności ROS związanych z wysyłaniem danych. Autorzy zakodowali węzły w Pythonie (biblioteka *rospy*) i ocenili wydajność systemu zarówno pod względem obliczeniowym, jak i komunikacyjnym.

Taktując to jako inspirację, zaproponowano bardzo podobne podejście, z dwiema głównymi różnicami projektowymi podyktowanymi kwestią wydajności: węzły szyfrowania oraz deszyfrowania zostały zakodowane w C++ (z *roscpp*) przy użyciu biblioteki Crypto++, natomiast w zaproponowanym podejściu wykorzystano Advanced Encryption Standard (AES) [30], o którym wiadomo, że jest szybszy niż 3DES. Algorytm AES to symetryczny szyfr blokowy, który konwertuje dane w postaci zwykłego tekstu do postaci niezrozumiałej, czyli tekstu zaszyfrowanego. Algorytm AES może wykorzystywać 128-, 192- i 256-bitowe klucze kryptograficzne do szyfrowania i deszyfrowania danych w blokach po 128 bitów.

Przedstawiono niezależne badanie z wykorzystaniem tej strategii

szyfrowania, którą nazwano *szyfrowaniem ROS-AES*. Ze względu na to, że *szyfrowanie ROS-AES* nie zmienia podstawowych pakietów ROS, wszelkie zapytania do wzorca ROS są spełnione, gdyż działający system jest standardowym systemem ROS. Z tego powodu szyfrowanie tylko komunikacji nie jest całkowicie bezpiecznym rozwiązaniem, co zostało przedstawione w części „Wyniki i dyskusja".

Secure ROS autorstwa SRI

Bezpieczny ROS został opracowany przez SRI International i zapewnia alternatywne wersje podstawowych pakietów ROS, które umożliwiają bezpieczną komunikację między węzłami ROS [31]. Głównym celem Secure ROS jest umożliwienie bezpiecznej komunikacji zwykłym użytkownikom ROS. W tym celu autorzy zintegrowali rozszerzenie IP z zabezpieczeniami (IPSec) w systemie ROS. IPSec jest używany w trybie transportowym, szyfrując i uwierzytelniając ładunek wymienianych wiadomości. Ponadto warstwy transportowe i aplikacyjne są zawsze zabezpieczone skrótem, tak więc nie można ich w żaden sposób modyfikować.

Użytkownik może określić autoryzowanych subskrybentów i wydawców *tematów*, ustawiających i pobierających *parametry*, a także dostawców (serwery) i żądających (klientów) *usług* w pliku konfiguracyjnym dla nadrzędnego ROS w czasie wykonywania. Z tego względu Bezpieczny ROS zezwala autoryzowanym węzłom łączyć się tylko z określonymi tematami, usługami i parametrami wymienionymi w określonym pliku konfiguracyjnym.

Biorąc pod uwagę wdrożenie, Secure ROS ma pewne podobieństwa do SROS. Proces instalacji jest jednak szybszy, ponieważ dostępne są kompilacje Debiana. Ponadto Secure ROS obsługuje zarówno *rospy*, jak i *roscpp*, można go więc łatwo skonfigurować, by był prosty i przejrzysty dla zwykłego użytkownika, który musi tylko dostarczyć wymagany plik konfiguracyjny z regułami dostępu do każdej jednostki ROS. Wadą Secure ROS jest to, że nie zapewnia formalnych środków weryfikacji gwarantujących, że pożądane właściwości są zgodne ze specyfikacjami.

Secure-ROS-transport

Dieber i inni zaproponowali architekturę bezpieczeństwa ROS do uruchomienia na poziomie aplikacji [32]. Korzystając z dedykowanego serwera uwierzytelniania, umożliwili bezpieczną komunikację pomiędzy węzłami ROS, wykorzystując metody kryptograficzne zapewniające poufność i integralność danych oraz unikające niektórych najpoważniejszych problemów związanych z bezpieczeństwem ROS. Jednakże zaproponowana architektura bezpieczeństwa przeznaczona była tylko do użycia z ROS. W związku z tym nadal występowały niektóre kluczowe luki, których nie można było rozwiązać na poziomie aplikacji, takie jak arbitralna subskrypcja danych pomimo ich szyfrowania oraz podatność na ataki DoS związana z wysoką częstotliwością publikowania fałszywych danych w tematach. W rezultacie autorzy przeprowadzili modyfikację podstawowych pakietów ROS w celu zwiększenia bezpieczeństwa ROS, tak jak opisano to w [33] i [34]. Doprowadziło to do powstania bezpiecznego kanału komunikacji, oznaczonego jako *secure-ROS-transport*, umożliwiającego węzłom ROS autentyczną i poufną komunikację na zasadzie *peer-to-peer*. Autorzy wykorzystali TLS dla TCP i Datagram TLS (DTLS) dla UDP, aby zabezpieczyć komunikację pomiędzy węzłami i urządzeniem głównym ROS, dodając dodatkowy krok uzgadniania i wykonując szczegółową autoryzację dla poszczególnych tematów. Takie podejście zmniejsza prawdopodobieństwo ataków DoS oraz arbitralnej subskrypcji i publikacji wiadomości w tematach ROS. Jednak, jak wskazali autorzy, nadrzędny ROS nie pozostaje bezpieczny, ponieważ ciągle przekazuje informacje o węzłach i tematach do dowolnego podmiotu, który go prosi (lista *rosnode*, lista *rostopic* itp.) i zezwala na specjalne wywołanie XMLRPC umożliwiające zewnętrzne wyłączenie dowolnego działającego węzła (*rosnode kill <node>*).

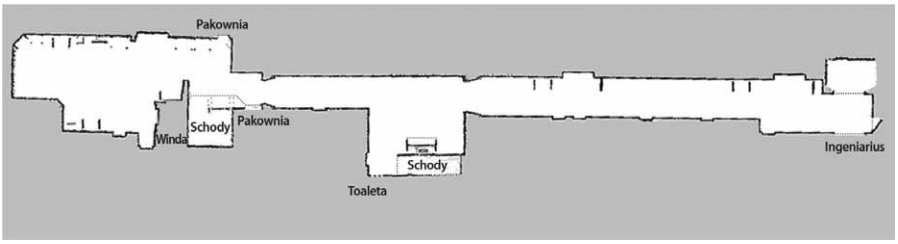
Konieczna była instalacja źródłowa *secure-ROS-transport*, ponieważ modyfikacje zostały zintegrowane z ROS. Dla użytkownika oznacza to, że nie ma potrzeby ponownej kompilacji węzłów w celu korzystania z *secure-ROS-transport*. Warto również zauważyć, że autorzy nie skupili się na obsłudze certyfikatów i zwracają uwagę na potrzebę właściwego zarządzania kluczami. Również w przypadku dostępnej wersji używana biblioteka Botan Crypto (v1.11) była już nieaktualna. Pomimo że studium przypadku zostało omówione w pracy [33], skupiając się głównie na powstałych wymaganiach wynikających z zabezpieczenia kanału komunikacyjnego, to dalszą szczegółową analizę przedstawiono w części „Wyniki i dyskusja".

Rosauth

Prace społeczności o nazwie *rosbridge* zyskują coraz większą uwagę. Koncentrują się one głównie na rozwiązaniach w chmurze i są skierowane na połączenie pomiędzy nienatywnymi klientami z systemami ROS, takimi jak roboty z obsługą ROS [35]. Kierując się koniecznością rozważenia krytycznego problemu bezpieczeństwa w tych systemach, w pracy [36] zaproponowano mechanizm uwierzytelniania *rosbridge* pod nazwą *rosauth*, który ma na celu osiągnięcie bezpiecznego uwierzytelnienia dla zdalnych, nienatywnych klientów w szeroko używanym oprogramowaniu pośredniczącym ROS.

Rosauth wykorzystuje tokeny uwierzytelniające do weryfikacji zdalnych klientów za pomocą dowolnego zewnętrznego systemu zarządzania użytkownikami, który jest zintegrowany jako część protokołu *rosbridge*. Zainspirowane kodami uwierzytelniania wiadomości (MAC, ang. *Message Authentication Codes*) są skracane za pomocą zdefiniowanego klucza i znanego algorytmu, który umożliwia uwierzytelnianie wiadomości. Serwer przechowuje kilka kluczy i jeśli otrzyma komunikat, to porównuje odebraną wiadomość zaszyfrowaną z wynikiem funkcji skrótu. Jeśli są one zgodne, to serwer zaakceptuje wiadomość. W przeciwnym razie będzie to oznaczało, że wiadomość pochodzi z niezaufanego źródła i zostanie zignorowana.

Protokół SSL służy do zapewnienia poufności, integralności i autentyczności poszczególnych pakietów. Używając certyfikatów wydanych przez zaufane urzędy certyfikacji, wykorzystując SSL, można zapewnić klientów zewnętrznych, że każdy system ROS, w tym zewnętrzny wystawca uwierzytelnienia, jest legalny.



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

Opracowany schemat tokenu zabezpieczającego zapewnia, że tylko klienci, którzy zostali uwierzytelnieni przez niektóre zaufane i zewnętrzne źródła uwierzytelnienia, mają dostęp do systemu ROS. Pomimo rozwiązania problemu z uwierzytelnianiem *rosauth* nie zapewnia poziomów autoryzacji. Zatem po uwierzytelnieniu każdy zdalny klient oraz każdy natywny podmiot uruchamiający ROS w tej samej sieci mogą uzyskać dostęp do całego systemu ROS, na przykład wysyłając bez ograniczeń bezpośrednie polecenia do robotów. Z tego powodu konieczne jest przeprowadzenie dalszych prac przez twórców *rosauth* związanych z rozszerzeniem mechanizmu uwzględniającego poziomy autoryzacji.

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 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 dostarcze-

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

niu 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

```
(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

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

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

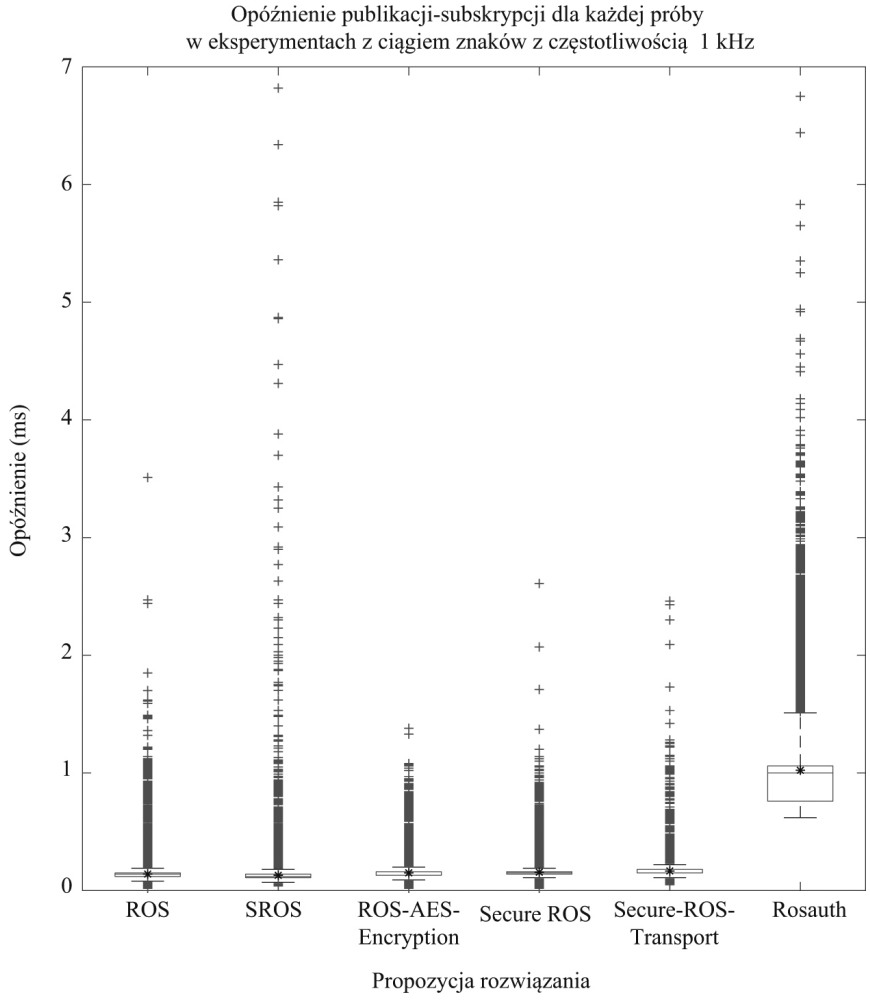
Tabela 3. Wyniki eksperymentów z mapami. Dla każdej linii publikowano i subskrybowano 150 tysięcy map, każda o wielkości 343 kB

	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

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

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



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

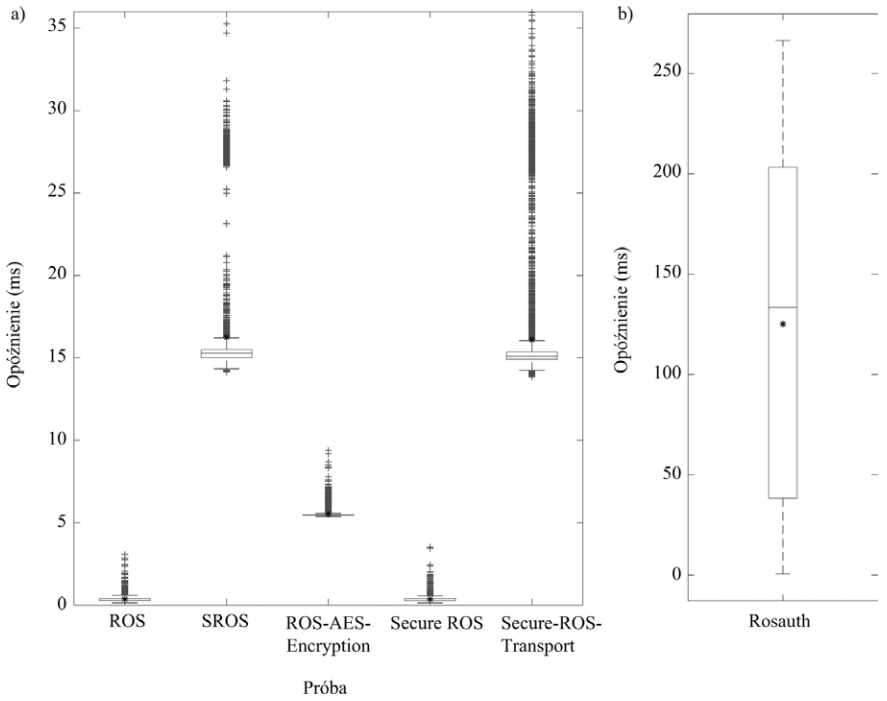
wiadomości w czasie trwania eksperymentu, który rozpoczął się z opóźnieniami wynoszącymi 0,7 s, podczas gdy na jego zakończeniu opóźnienia wynosiły 266,2 s. Z tego samego powodu co poprzednio testy dla 2,5 kHz i 7,5 kHz zostały pominięte dla *Rosauth*.

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



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	+	–	–	–	–	+

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 roboty, 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.

Przypisy

[1] http://stop.ingeniarius.pt.
[2] http://www.ros.org/.
[3] TOR oznacza *The Onion Router*, światową sieć serwerów, które umożliwiają anonimowe przeglądanie Internetu.
[4] http://wiki.ros.org/ros_comm6.
[5] http://wiki.ros.org/sros.

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