Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Aspekty bezpečnosti Robot Operating System

Bakalářská práce

Petr Kubovský

Vedoucí práce: Mgr. Jiří Pech, Ph.D.

České Budějovice 2021

Bibliografie

Kubovský, P., 2021: Aspekty bezpečnosti Robot Operating System. [Security aspects of the Robot Operating System. Bc. Thesis, in Czech.] – 45 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace/Annotation

This Bc. thesis focuses on the problematics of security of Robot Operating System (ROS). Except short introduction and description of basic ROS principles the main goal of this thesis is to show possible security problems, their analysis and misuse for some possible attacks on system. Some of the attacks will be shown in practical part. Another goal of this thesis is evaluation of overall security of ROS, recommendations how to use ROS safely and how how to effectively protect the system against some of the described attacks.

Prohlášení

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

České Budějovice, 13.4. 2021

.....

Petr Kubovský

Cíle Práce

1) Představit Robot Operating System, popsat jeho úrovně a základní pojmy.

2) Nastínit možné zranitelnosti ROS.

3) Teoreticky popsat možné vektory útoku na ROS.

4) Popsat, jak se proti útokům bránit a jak dále zabezpečit ROS.

5) Prakticky předvést některé z útoků.

Použitý software a jeho verze:

ROS Melodic/Kinetic

Linux Ubuntu 18.04

Kali Linux 2020.3

Oracle VM VirtualBox 6.1

Klíčová slova

Robotika, ROS, Robot, Bezpečnost, Útok, Zranitelnost

Poděkování

Za pomoc při tvorbě práce bych chtěl poděkovat vedoucímu práce, panu Mgr. Jiřímu Pechovi, Ph.D.

Obsah

1. Úvod

S roboty se můžeme setkat každý den v nejrůznějších odvětvích a je nejspíš jen otázka času,než se stanou nedílnou součástí našeho každodenního života. Mezi hlavní výhody robotů patří jejich spolehlivost, přesnost, schopnost pracovat téměř nepřetržitě apod. V dnešní době se můžeme setkat se širokou škálou robotů. Od jednodušších robotů vykonávajících jednoduché, monotónní úkoly v továrnách až po humanoidní roboty, se kterými lze komunikovat, jako s běžnými lidmi.

Je pravděpodobné, že v budoucnosti bude velká část robotů připojena k internetu za účelem komunikace s jinými vzdálenými robotickými systémy nebo například cloudy. S rapidně rostoucím množstvím kybernetických útoků je ovšem nutné zamyslet se nad množstvím otázek týkajících se bezpečnosti robotických systémů připojených do sítě.

Při napadení běžného počítače, může dojít ke ztrátě nebo poškození dat, úniku osobních údajů apod. I když toto je velmi nepříjemné, tak při napadení robota se objevují úplně nové, ještě větší bezpečnostní hrozby. Pokud si robota představíme jako počítač, který má ruce, nohy, kolečka nebo různé kamery a senzory, není těžké představit si, jaké bezpečnostní hrozby vznikají, když robot není dostatečně zabezpečen a útočník nad ním získá kontrolu.

Robot se tak může rázem proměnit z lidského pomocníka ve zbraň schopnou ničit majetek kolem sebe nebo ublížit lidem pohybujícím se poblíž. Toho je často možné docílit jednoduchou změnou bezpečnostních nastavení nebo podvrhnutím falešných senzorových dat, které informují o tom, jestli je někdo poblíž.

Tato práce se soustředí na problematiku bezpečnosti robotů využívajících celosvětově rozšířený open-source framework Robot Operating System (ROS), který je ovšem známý tím, že bezpečnost u něj není úplně tou nejsilnější stránkou.

2. Robot Operating System (ROS)

Ještě předtím, než se dostaneme z otázkám bezpečnosti si ROS představíme, popíšeme jeho princip a vysvětlíme si nějaké základní pojmy. Robot Operating System je open-source framework vydaný pod licencí BSD, který slouží k programování robotických aplikací. Nejedná se o operační systém v pravém slova smyslu, jak by se mohlo zdát z názvu, ale spíše o middleware, který programátorovi poskytuje kolekci různých nástrojů, knihoven a konvencí, které mají sloužit především ke zjednodušení vývoje komplexních robotických systémů.

Jako příklady toho, co ROS nabízí, můžeme uvést třeba správu balíčků, různé simulátory, vizualizační nástroje nebo vlastní mezi-procesorovou komunikaci. Jako příklady balíčků lze uvést například měřič vzdálenosti nebo velké množství balíčku sloužících pro sběr dat ze senzorů. Pro psaní jak knihoven, tak i aplikací jsou využívány především jazyky C++ a Python. Tyto jazyky jsou podporované již v základu. Experimentálně je možné používat i například jazyky Java, Ruby atd. Je důležité zmínit, že je možné do systému integrovat real-time kód i přesto, že systém sám o sobě real-time není.

Mezi jednu z nevýhod systému patří omezená podpora na operačních systémech Microsoft Windows, OS X a Android. Tyto systémy buď nejsou podporovány vůbec, nebo jen neoficiálně. Pokud se tedy ROS na některých z výše uvedených systémů vůbec podaří spustit, tak nemusí fungovat se vším všudy a jeho běh může být nestabilní. Jediný oficiálně podporovaný operační systém je Linux Ubuntu a jeho další varianty Kubuntu, Xubuntu, Lubuntu atd. Další nevýhodou jsou problémy s bezpečností, což je jedním z hlavních témat této práce.

Poslední nevýhodou, kterou zde zmíníme je velké množství změn v a velké množství verzí. První verze vyšla v roce 2007 a i přesto, že rozdíly mezi jednotlivými verzemi nejsou nijak velké, tak pořád můžeme občas narazit na problémy s kompatibilitou. U staršího robotů, které neběží na nejnovějších verzích ROS je možné, že budeme muset vzít v potaz to, jakou verzi Ubuntu potřebujeme nebo jaké verze knihoven musíme použít.

Obr. 1 – Podpora ROS.

Zdroj: https://slideplayer.com/slide/6202942/

Mezi výhody patří velká open source komunita okolo ROS. Uživatelé mohou snadno sdílet kód či balíčky mezi sebou. Je tedy velmi jednoduché sdílet vlastní aplikace s ostatními uživateli nebo naopak vyzkoušet aplikace vytvořené někým jiným. Mezi další výhody patří již zmiňovaný velký počet knihoven a nástrojů. Z výše zmíněných výhod a nevýhod tedy můžeme usoudit, že ROS je vhodný například pro open source vývojáře, pro výzkum nebo pro nadšence, kteří chtějí někde začít a nevědí kde. Kvůli mnohým nedokonalostem a bezpečnostním dírám není ale ROS ideální volbou pro tzv. „mission-critical“ aplikace.

3. Úrovně ROS a základní pojmy

Strukturu ROS je v principu možné rozdělit do třech hlavních úrovní. První úrovní je souborový systém. Druhou výpočetní grafy a třetí uživatelská komunita. Nás budou zajímat především první dvě úrovně. K oběma z nich si později vysvětlíme nějaké základní pojmy. Ještě předtím si něco povíme o klientských knihovnách.

3.1 Klientské knihovny

Jak už bylo, řečeno ROS není operační systém v pravém slova smyslu, ale jedná se o tzv. meta-operační systém. V podstatě je to framework vybudovaný nad operačním systémem Unix. Abychom mohli komunikovat s ROS entitami, budeme k tomu potřebovat nějaké knihovny. Knihoven je velké množství, stejně jako jazyků, ve kterých je možné psát kód. Bohužel velké množství jazyků je podporováno jen experimentálně a psaní kódu v nich je na vlastní riziko.

Jediné robustněji podporované jazyky, ve kterých se doporučuje ROS programy psát, jsou C++ a Python. Příslušné knihovny se nazývají roscpp a rospy. S využitím těchto knihoven je možné psát kód ROS uzlů, publikovat a odebírat z topics, psát a volat služby atd. Co se týče volby mezi C++ a Pythonem, doporučuje se využívat C++ pro aplikace, u kterých je důraz kladený na rychlost. Python je naopak doporučeno využít tam, kde preferujeme rychlý čas implementace oproti rychlému běhu programu. Například, pokud chceme rychle vytvořit prototypy programů nebo otestovat algoritmy.

3.2 Souborový systém

- **Balíčky (packages)**: Jsou hlavní organizační jednotkou softwaru v ROS, sloužící k udržování přehlednosti v uložených datech a k tomu aby se software dal snadno znovu využít. Balíček může obsahovat například seznam ROS uzlů (nodes) reprezentujících běžící procesy, různé knihovny, konfigurační soubory, databáze apod. Balíčky jsou nejmenší individuální jednotkou v ROS, kterou je možné vydat nebo sdílet s ostatními.
- **Metabalíčky (metapackages)**: Jedná se o speciální typ balíčků, které neobsahují žádné soubory, testy nebo kódy, které se vyskytují v běžných balíčcích a ani neinstalují žádné soubory, kromě popisného souboru package.xml, o kterém je řeč níže. Metabalíčky slouží pouze k seskupení více balíčků dohromady pomocí referencí. Na podobném principu fungují například virtuální balíčky v Debianu.

- **Popisné soubory (manifests)**: Jedná se o soubory package.xml, poskytující informace (metadata) o balíčku, ve kterém se nachází. Mezi těmito informacemi je například jméno, verze, popis, informace o licenci, závislosti a podobná metadata.
- **Repozitáře (repositories)**: Kolekce balíčků, které sdílejí společný systém zprávy verzí (VCS). Slouží k tomu, aby balíčky se stejnou verzí mohly být vydány společně.
- **Typy zpráv (message types)**: Soubor definující datové typy zpráv posílaných v rámci ROS. Soubory mají příponu .msg a nacházejí se v msg/ podadresáři daného balíčku.
- **Typy služeb (service types)**: Soubor definující datové typy zpráv request a response pro služby v ROS. Typy zpráv request a response jsou oddělené pomocí "---".

3.3 Výpočetní grafy

- **Uzly (nodes)**: Jedna ze základních věcí, na kterou musíme při programování v ROS myslet je rozdělení našeho kódu do jednotlivých modulů, které by měly provádět pokud možno jeden specifický úkol. Úkoly by ideálně měly být jednoduché a relativně krátké, co se týče velikosti kódu i délky výpočtu. Každý kus program provádějící určitý úkol se nazývá uzel, což je vlastně jeden proces. Z těchto uzlů je sestaven výpočetní graf a jednotlivé uzly si mezi sebou mohou posílat zprávy po komunikačních kanálech, kterým se říká topics. Uzlů je obvykle velké množství. Jednou z výhod velkého množství uzlů je odolnost vůči různým nepříjemným událostem, jako jsou výpadky nebo poruchy. Ty se týkají pouze jednotlivých uzlů a neovlivňují celý systém.
- **Hlavní uzel (ROS master)**: V současné době je ROS master asi nejdůležitější komponenta implementovaná v ROS. Jedná se o server implementovaný přes XML-RPC protokol (bezstavový protokol založený na HTTP), který poskytuje množství funkcí ostatním agentům (uzlům). Master slouží například k udržování metainformací o všech ostatních uzlech, topics a také o některých dalších věcech, jako jsou například parametry. Dále je jeho úkolem poskytovat ostatním uzlům informace o výpočetním grafu, čímž jim umožňuje se navzájem alokovat a komunikovat spolu. Také jim poskytuje možnost registrace a volby jména.

  Uzly mohou navzájem komunikovat přímo spolu (peer-to-peer) až poté, co došlo ke vzájemné alokaci. Při startu, kdy se uzly ještě vzájemně nealokovaly, je důležité, aby znaly adresu ROS mastera. Navázání spojení funguje následujícím způsobem. Nějaký

uzel informuje ROS mastera o svojí IP adrese a o tom, na který topic rozesílá data. Tuto informaci si master zapamatuje. Pokud dá později jiný uzel najevo, že by měl zájem odebírat z tohoto topic, informuje ROS master odběratele o adrese uzlu, který rozesílá data. Tyto dva uzly spolu navážou peer-to-peer spojení a nyní spolu mohou komunikovat přímo.

- **Parametrický Server (Parameter Server)**: V současné době je tato komponenta součásti ROS Master. Stejně jako ROS Master i tento server je implementován přes XML-RPC protokol. Využití tohoto protokolu umožňuje jednoduchou integraci s klientskými knihovnami a také poskytuje lepší typovou flexibilitu při ukládání nebo načítání dat. Parametrický server může uchovávat běžné XML-RPC skaláry, seznamy a binární data zakódovaná na principu base64, což je kódování umožňující převádět binární data na posloupnost tisknutelných znaků. Může také uchovávat slovníky, kde každý slovník má speciální význam. Například tzv. dictionary-of-dictionary je struktura sloužící k reprezentaci jmenných prostorů. Jeden slovník reprezentuje jednu úroveň v hierarchii jmen a další slovník reprezentuje vnitřní strukturu větvení daného jmenného prostoru.
- **Zprávy (messages)**: Jedná se o jednoduché datové struktury, pomocí kterých spolu uzly komunikují. Mohou být složené z primitivních datových typů nebo polí primitivních datových typů. Datová struktura zprávy je specifikovaná v textovém souboru s příponou .msg, který se nachází v msg podadresáři v balíčku. Před použitím musí být .msg soubory zkompilovány do C++ nebo Python tříd. To ale probíhá automaticky, takže se nemusíme starat o serealizaci a deserializaci dat.
- **Témata (topics)**: Zprávy v ROS se posílají přes jakési pomyslné komunikační kanály, které se nazývají topics (témata). Jedná se vlastně o jméno sloužící k identifikaci obsahu zprávy. Uzly spolu komunikují prostřednictvím publish/subscribe mechanizmu (vydavatelé/odběratelé). Uzel může zveřejnit data v podobě zprávy na daný topic. Jiný uzel, který má o tyto data zájem o může na tom samém topic odebírat. Pravidla pro zveřejňování/odebírání jsou následující:

  1) Jakýkoli uzel může zveřejňovat nebo odebírat na jakýkoliv topic.

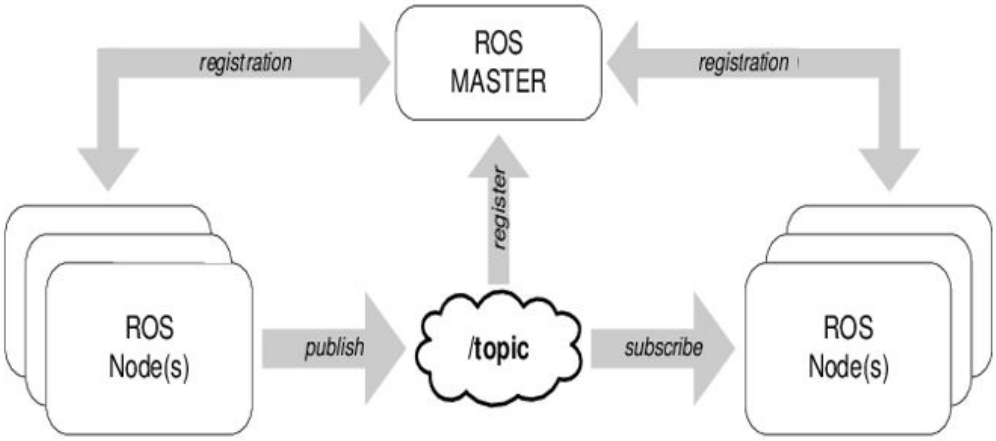
  2) Více uzlů může konkurenčně zveřejňovat nebo odebírat na ten samý topic.

  3) Jeden uzel může zveřejňovat nebo odebírat na více topics.

  4) Jeden uzel nemůže zveřejňovat a odebírat ve stejnou dobu.

5) Obecně platí, že uzly, které zveřejňují a odebírají o sobě vůbec nevědí.

- **Služby (services)**: Služby jsou definovány jako dvojice zpráv, kde jedna slouží k odeslání dotazu (request) a druhá k přijmutí odpovědi (response). Tím že uzel poskytuje službu jinému, mu dává možnost vykonání svých vlastních operací.



Obr. 2 - Základní komunikační model.

Zdroj: https://www.researchgate.net/figure/Nodes-communication-model-in-the-ROS-environment-Fig-5-describes-the-proposed-ROS_fig2_319566597

4. Zranitelnosti

Jak už bylo řečeno dříve Robot Operating System je relativně známý tím, že obsahuje velké množství bezpečnostních děr, kterých potenciální útočník může využít. V této části si některé z nich představíme.

- **Komunikace v plain-textu:** V základě je komunikace jak s externím webovým rozhraním, tak mezi uzly, prováděna v plain-textu . To má určité výhody, např. jednoduchost použití, debugging a zvýšení výkonu, protože nejsou nutné další kryptografické kroky. V případě externího webového rozhraní by byl přechod na zabezpečení HTTPS relativně snadný s moderními webovými frameworky. V případě komunikace mezi uzly je zřejmé, že komunikace v plain-textu umožňuje neautorizovanému uživateli snadno interpretovat formu zprávy a spoofovat falešné zprávy (Mcclean a Farrar, 2013, s. 5).
- **Nešifrované uložení dat:** Ve velkém množství aplikací je neakceptovatelné, aby v případě nějakého problému získal někdo neoprávněný přístup k potenciálně citlivým datům. Z toho důvodu se uložená data šifrují. Šifrování ovšem značně snižuje šanci záchrany dat v extrémních situacích, jako jsou třeba poruchy hardwaru. Záchrana dat v takovýchto situacích je někdy vhodná třeba pro výzkumné účely. Z toho důvodu jsou veškerá data v ROS uložena nešifrovaná.
- **Nechráněné TCP porty:** Jedna z věcí, díky které ROS velmi oblíbený je vysoká modularita systému. Díky tomu je možné vytvářet modulární robotické systémy, které mohou být distribuovány po celém světě. Přináší to ale i určité nevýhody.

  Interní komunikační struktura ROS je vybudovaná okolo TCP portů, což umožňuje, aby modulární robot mohl být distribuován po celém světě, pokud je potřeba. Nicméně nevýhodou této modularity je zranitelnost TCP portů, které momentálně nabízejí velmi malou úroveň autentizace (Mcclean a Farrar, 2013, s. 5). Toho může opět zneužít potenciální útočník například k útokům typu man-in-the-middle. Typicky se jedná o neautorizovaný odposlech přenášených zpráv, jejich modifikaci nebo nahrazení existujících telkers/listeners, čímž lze přesměrovat externí pakety směrem k ROS portům.

- **Chybějící autentizace a řešení konfliktů:** Dalším bezpečnostním problémem je, že ROS používá tzv. slabé autentizační schéma. To znamená, že systém např. neumožňuje definovat přístupová práva, nepodporuje ani kontrolu integrity

  a autentičnosti dat a nevyžaduje verifikaci odesílatele. Díky tomu mohou robotovi posílat příkazy i uživatelé, kteří by k němu jinak přístup mít vůbec neměli. Mezi nedostatky patří i to, že systém nedokáže rozlišit konflikty, díky tomu může příkazy rozesílat i více uzlů naráz. Toho lze zneužít například pro útoky typu Denial of service (DoS). Jednotlivé uzly navíc vůbec neví, s kým komunikují, protože ROS používá tzv. anonymní publish/subscribe sémantiku.

- **Chybějící aspekt Quality of Service (QoS**): V ROS si každý uzel zodpovídá sám za sebe a za svou vlastní komunikaci. Bohužel se nijak neřeší věci jako např. komprese podobných zpráv, kterou by šlo využít ideálně pro zprávy stejného typu. Tím by se značně snížilo i množství síťové komunikace. Tento nedostatek bohužel částečně brání ve vývoji tzv. time-critical aplikací, ve kterých je potřeba množství komunikace co nejvíce omezit.

5. Útočení na ROS

V této sekci si představíme některé z možných vektorů útoku na systémy využívající ROS. V podstatě všechny možné útoky můžeme rozdělit do jedné ze dvou skupin. Jedná se o útoky fyzické a logické. My se budeme soustředit především na útoky logické a o fyzických si pouze něco málo řekneme.

Fyzické útoky jde rozdělit do dalších dvou skupin, a to kontaktní a bezkontaktní. Kontaktní útokem může být jakékoliv fyzické napadení robota, ať už pomocí vlastního těla nebo zbraně. Typicky se může třeba o sestřelení dronu, hození robota na zem, kopnutí do něj ale třeba i krádež robota. Ve skupině bezkontaktních fyzických útoků nalezneme útoky na komunikační kanály. Může se jednat o rušení signálu nebo manipulaci se zprávami signálem na stejné vlnové délce.

Teď již přejdeme k útokům logickým. Pokud se bavíme o tak komplexním problému, jako o otázkách bezpečnosti, je důležité vědět, co se pojmem bezpečnost v IT vlastně rozumí. K tomu je nutné zajistit čtyři klíčové aspekty bezpečnosti, kterým je důležité porozumět. Například, pokud někdo chce vyvinout bezpečnostní systém pro ROS postavený nad existující infrastrukturou, kde veškerá komunikace probíhá na principu publish/subscribe bez využití jakýkoliv bezpečnostních metod, musí se snažit tyto aspekty zajistit, jak jen to jde.

- **Integrita:** Tato podmínka je splněná, pokud se přečtená data shodují s těmi uloženými. Během přenosu dat tedy nesmí dojít k modifikaci dat nějakou neautorizovanou entitou. Integritu mohou narušit typicky útoky typu „man in the middle“ nebo spoofing. Proti těmto útokům se lze bránit například pomocí hashování nebo kontrol integrity.
- **Autenticita:** Jedná se o vlastnost subjektu, která se ověřuje procesem autentizace. Autentizace ověřuje, jestli je subjekt opravdu tím, za koho se vydává. Autenticitu pomáhají zajišťovat například různé certifikáty nebo digitální podpisy.
- **Dostupnost:** Znamená, že data nebo zařízení jsou k dispozici v případě potřeby. Vyjadřuje se v procentech vyjadřujících dostupný čas za dané období, obvykle rok. Například dostupnost 99% znamená, že data mohou být nedostupná maximálně 3,65 dne v roce.

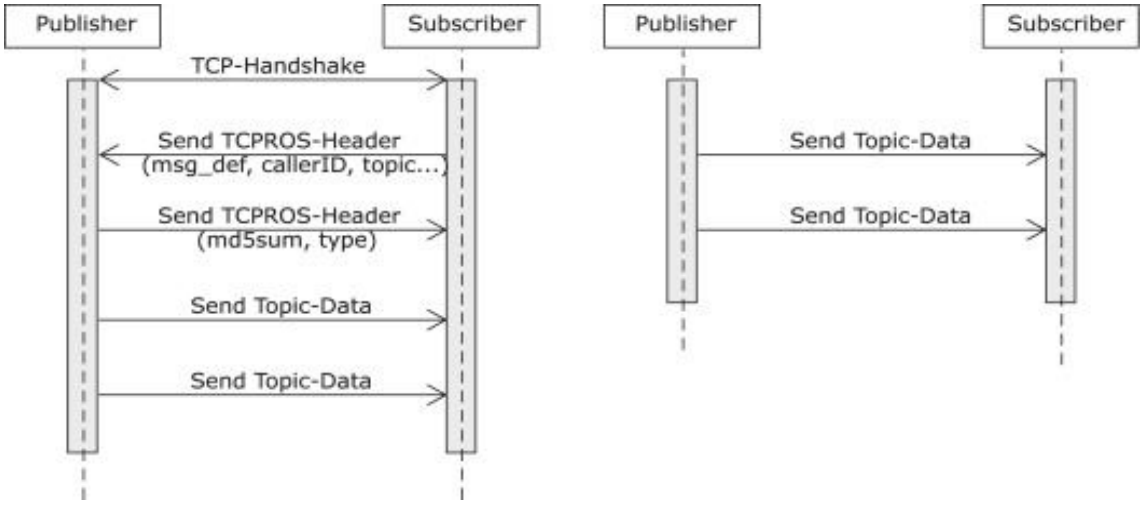
- **Důvěrnost:** Zajištění důvěrnosti znamená zabránit vyzrazení informace nebo přístupu k datům subjektům třetích stran a podobným neautorizovaným entitám. V každém okamžiku zpracování dat musí existovat určitá míra utajení. Důvěrnost je nutné zajistit jak během uchovávání dat, tak během přenosu. Důležité je zajistit, aby neautorizovaný subjekt nemohl vniknout do systému. Důvěrnost zajišťuje například šifrování

5.1 Registrace nového uzlu

Předtím, než se podíváme na konkrétní vektory útoku na ROS, si popíšeme, co se vlastně děje při spouštění ROS aplikace a jak probíhá registrace nového uzlu. Znalost spouštěcí procedury je klíčová, pokud chceme na ROS útočit. Jako úplně první musí být spuštěn ROS master, ke kterému se poté zaregistruju uzel, který chce publikovat. Ten pomocí XML-RPC zavolá na ROS masterovi proceduru registerPublisher s názvy všech topics na které může publikovat, parametry a jejich typy.

Tyto informace budou v budoucnu potřebovat všechny uzly, které budou mít zájem z nějakého topic odebírat. Po obdržení odpovědi lze uzel považovat za registrovaný. Podobný postup platí i pro registraci nových odběratelů. Ti akorát zavolají pomocí XML-RPC proceduru registerSubscriber. V obdržené odpovědi jsou obsažené informace o parametrech, které umožňují odběrateli kontaktovat správného vydavatele. Pomocí volání procedury requestTopic může odběratel vydavateli poskytnout seznam protokolů potřebných pro komunikaci. Vydavatele poté vrátí vybraný protokol spolu s dalšími parametry potřebnými pro navázání úspěšného spojení.

V této fázi je odběratel už schopen komunikovat a odebírat zprávy z topics, na které druhý uzel publikuje. Komunikace je možná pomocí protokolů TCP nebo UDP. V případě použití TCP před vlastní výměnou dat proběhne ještě handshake, při kterém si uzly vymění některé dodatečné informace. V případě komunikace přes UDP uzly komunikují rovnou bez handshake. Na obrázkách dole je znázorněno, co se děje po počátečním zavolání XML-RPC a to jak v případě použití TCP, tak i UDP.



Obr. 3 - Průběh komunikace mezi dvěma uzly po XML-RPC handshake. Nalevo TCP, napravo UDP.

Zdroj:

https://www.sciencedirect.com/science/article/abs/pii/S0921889017302762

5.2 Možné vektory útoku

5.2.1 Neautorizované publikování (Data Injection)

Tento typ útoku funguje na principu vložení falešných dat do již běžící ROS aplikace. Jiný ROS uzel pak tato data odebere od falešného vydavatele. V tomto typu útoku útočník využije volání publisherUpdate k izolování odběratele od jednoho nebo více regulérních vydavatelů. Nakonec odběratele donutíme k navázání spojení s neautorizovaným vydavatelem, o kterém ROS master nemusí ani vědět, že existuje. .

Další volání, která v tomto typu útoku útočník využije využijeme jsou volání getSystemState a lookupNode z ROS Master API a requestTopic ze XML-RPC Slave API. V tomto scénáři budeme uvažovat 4 entity. ROS Master, odběratel, který odebírá z topic, publikující subjekt, který zveřejňuje zprávy a útočník , který útočí na odběratele.

Scénář je v podstatě rozdělen do přípravné fáze, kdy útočník získá potřebné informace pro spuštění útoku a fáze útoku, kdy jsou komunikační vztahy týkající se odběratelů a topics manipulovány tak, že odběratel přijímá zprávy pouze od útočníka, a to až dodatečně. V prvním kroku útočník požaduje aktuální stav systému v ROS síti od mastera, a to voláním metody getSystemState. Nyní útočník ví, které uzly právě komunikují přes která topics. Zejména útočník ví, že odběratel a vydavatel komunikují přes cílený topic. Následným

dvojitým voláním metody lookupNode získá útočník XML-RPC URI odběratele a vydavatele. S touto informací může útočník nyní přejít do útočné fáze.

První útočník odešle volání publisherUpdate odběrateli, které obsahuje pouze útočníkovu XML-RPC URI v seznamu vydavatelů, aktuálně známých odběrateli. V důsledku toho odběratel ukončí spojení s vydavatelem a zahájí komunikaci s útočníkem odesláním volání requestTopic.

Od této chvíle musíme rozlišovat, zda chce odběratel používat TCPROS nebo UDPROS pro přenos dat. V případě TCPROS útočník přeposílá přijaté volání vydavateli a přijímá mimo jiné i port, kde vydavatel naslouchá novým TCP spojením. Před odesláním odpovědi odběrateli musí útočník změnit informace o hostiteli a portu. Jako další odběratel naváže TCP spojení s útočníkem a odešle hlavičku TCPROS. Útočník pak stejným způsobem přepošle tuto hlavičku vydavateli, aby obdržel správnou zprávu TCPROS header, kterou může poslat zpět odběrateli. Poté může útočník začít posílat vlastní zprávy odběrateli.

Při použití UDPROS je hlavička UDPROS zahrnuta ve volání request Topic Call odběratele. V důsledku toho musí útočník znovu přeposlat volání vydavateli, aby získal správnou hlavičku pro odpověď.Poté, co útočník poslal korektní odpověď odběrateli, Útočník okamžitě začne odesílat zprávy na topics.

Důležité je, že počet vydavatelů, které lze vyloučit, není v tomto scénáři omezen. Pokud bychom měli více vydavatelů, odběratel by ukončil veškerou komunikaci k nim a útočník by si vybral jednoho z vydavatelů, aby získal správné informace o záhlaví. Teoreticky by informace získané od mastera v první fázi mohly být vyžádány i od odběratele nebo vydavatele odesláním volání getBusInfo, ale to by předem vyžadovalo další informace jako URI XML-RPC od odběratele a vydavatele. Bez ohledu na to, jak vypadá přípravná fáze, master si není vědom změn v ROS grafu, které z útoku vyplývají. Detekce tohoto útoku proto vyžaduje pokročilé analytické metody ROS grafů.

5.2.2 Neautorizovaný update parametru

ROS Parameter API poskytuje dvě různé možnosti pro ROS, jak získat aktuální hodnoty parametru uloženého na parametrickém serveru. První a nejspíš i běžnější způsob je zavolat metodu getParam. Libovolný ROS uzel si zažádá o hodnotu parametru od ROS Master a ten mu zašle aktuální hodnotu parametru. Druhou možností je odběr konkrétního parametru pomocí metody subscribeParam. V tomto případě uzel ukládá aktuální hodnotu parametru do lokální proměnné. Pokud se na parametrickém serveru změní hodnota parametru, server

zavolá metodu paramUpdate uzlu, což má za následek změnu jeho lokální proměnné pro tento parametr. V tomto typu útoku útočník využije tohoto chování ke změně hodnoty parametru lokálně přímo na uzlu, aniž by se jakkoli dotkl odpovídající hodnoty na parametrickém serveru.

První se uzel musí přihlásit k odběru parametru zavoláním metody subscribeParam, čímž dojde k předání parametru obsahující název uzlu, URI místního XML-RPC serveru a název požadovaného parametru ROS masterovi. Přijatá odpověď pak obsahuje hodnotu se současným názvem parametru. Při úspěšném odběru požaduje uzel hodnotu parametru podruhé, a to s využitím metody getParam. Nyní přichází na řadu útočník, který získá XML-RPC URI uzlu pomocí metody lookupNode. Dále útočník zruší odběr parametru uzlem tak, že předstírá volání metody unsubscribeParam z uzlu. V rámci této konfigurace přestane parametrický server odesílat aktualizace pro parametr uzlu, zatímco uzel stále čeká na požadavky paramUpdate od parametrického serveru. Útočník může tento stav využít k získání plné kontroly nad hodnotou parametru, který je lokálně uložen na uzlu, a to pouhým odesláním vlastních požadavků paramUpdate uzlu.

Zde popsaný scénář lze také použít na více uzlech, které jsou přihlášeny k odběru stejného parametru. V nejhorším případě se může stát, že každý uzel uvidí v určitém okamžiku jinou hodnotu stejného parametru, což může vést například k nepředvídatelnému chování v distribuované ROS aplikaci.Důležité je zmínit, že parametry v mezipaměti jsou podporovány pouze balíčkem Roscpp. Útok využívající update parametrů lze tedy spustit pouze na uzlech implementovaných v C + +.

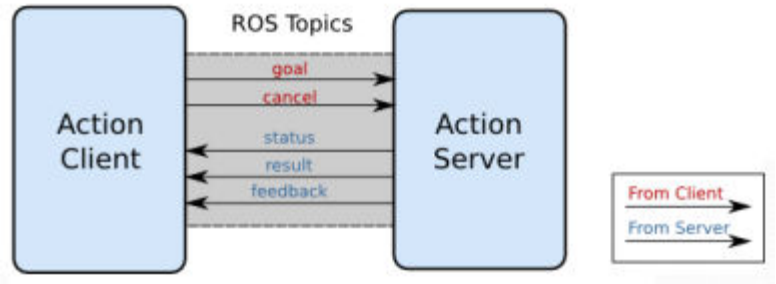
5.2.3 Modifikace dat (Man in the Middle)

Předpokládejme, že máme klienta a server, kde server poskytuje klientovi libovolnou akci. Pokud chce klient spustit tuto akci, odešle zprávu na topic specifický k dané akci. Ke zrušení akce musí klient odeslat preemptivní požadavek na topic určený ke stornování akce. Útočník nyní může tuto komunikaci zachytit spuštěním útoku data injection popsaném v sekci 5.2.1 na tyto dva topics.

Kromě toho, může útočník pomocí publikování vlastních zpráv jednoduše vyvolat a zrušit upravenou instanci akce sám, bez jakékoli další pomoci. Až doposud útočník nebrání serveru v odesílání zpráv o zpětné vazbě, výsledcích a stavu klientovi. Proto může klient odhalit útok interpretováním neočekávaných zpráv přijatých od serveru. To se změní, jakmile útočník provede další tři další data injection útoky na topics status, feedback a result. Nyní útočník

může publikovat zprávy, které by klient očekával jako odpověď na své vlastní zprávy na topics goal a cancel.

Z hlediska komunikace, je tento scénář je jen mnohonásobné uplatnění dříve popsaného útoku data injection. Hlavní výzvou pro útočníka jsou kontextově citlivé znalosti potřebné k předstírání rozumného chování klienta a serveru, aby útočník zůstal nezjištěn. Kromě toho samotný útok opět není viditelný v ROS grafu, takže je stejně těžké ho odhalit jako útok data injection.



Obr. 4 – Diagram systému klient - server z příkladu nahoře.

Zdroj: https://thinkcse.wordpress.com/2015/01/13/ros-101-the-action-server-client-model/

5.2.4 DoS (Denial of Service)

Útok typu denial of service lze v principu provést třemi způsoby. Prvním způsob je v celku přímočarý. Jedná se o to, že vytvoříme uzel, který se bude jmenovat stejně, jako některý z již existujících uzlů. Tím by se měl původní uzel zcela vypnout.

Druhým způsobem provedení DoS útoků v ROS je pomocí publikování velkého množství falešných dat. Toto může vést k velké zátěži při zpracování na všech uzlech a může vést k potenciální neschopnosti provádět smysluplné výpočty. Protože zde není žádná kontrolou toho, jaký uzel by měl publikovat jaká data, tak jakýkoli uzel v síti může být použit k publikování dat na topic, který odebírá cílový uzel.

To je možné využít k cíleným DoS útokům na tento uzel (Breiling, Dieber a Schartner, 2018, s. 3). Uzle může být vypnut i pomocí jednoduchého volání XML-RPC. Tento způsob shození uzlu má ale tu nevýhodu, že uzel viditelně zmizí z ROS grafu, čímž se útočník v podstatě prozradí. Pokud útočník chce zůstat krytý, je tedy lepší předchozí metoda vysokofrekvenčního publikování.

6. Obrana proti útokům

6.1 Zabezpečení na úrovni konfigurace

Možností, jak se bránit proti potenciálním útokům je velké množství. První možností, jak znesnadnit útočníkovi práci a zvýšit bezpečnost na úrovni konfigurace je vypnout ROS mastera, hned po tom, co se aplikace inicializuje. Na druhou stranu se tím ale značně sníží robustnost aplikace kvůli tomu, že parametrický server po vypnutí mastera není dostupný pro ROS uzly. Navíc pokud by bylo nutné ROS master znovu zapnout nebo restartovat kterýkoli jiný uzel, tak to nebude možné, protože restartující se uzel se nemůže znovu připojit do grafu uzlů.

Další možné opatření, které je možné udělat na úrovni konfigurace je změnit síťový port ROS mastera. Tím lze značně zpomalit postup útočníka. Pokud totiž chce zjistit, na kterém portu ROS master běží, nemá v podstatě jinou možnost, než provést skenování celého rozsahu portů, což je možné snadno detekovat pomocí různých bezpečnostních systémů.

6.2 Zabezpečení na aplikační úrovni

Druhou alternativou k zabezpečení ROS je zabezpečení na aplikační úrovni. To je možné provést například za pomoci dedikovaného autentizačního serveru. Tento server má sloužit k tomu, aby sledoval, který uzel je oprávněn publikovat na který topic nebo z něj odebírat.

Jak už název napovídá, dalším úkolem autentizačního serveru je starat se o autentizaci nově příchozích uzlů a také generovat klíče specifické pro dané topics. Server by pokud možno měl být ve vyšší bezpečnostní doméně, než zbytek systému, aby umožňoval silnější přístupovou kontrolu jak na úrovni fyzické, tak na úrovni logické.

Nevýhodou tohoto řešení s využitím autentizačního serveru u systému, který není distribuovaný je, že veškerá bezpečnostní opatření jsou soustředěná v jednom bodě, kterým je právě tento server. Celý systém je tak poměrně zranitelný v případě útoku na server nebo v případě nějakého jiného problému s ním. Toto řešení se tedy vyplatí hlavně v případě vysoce distribuovaných systémů.

6.3 Zabezpečení komunikačního kanálu

V této části si teoreticky popíšeme, na jakém principu a s využitím jakých protokolů by bylo možné zabezpečit komunikační kanál v ROS. Tyto úpravy by bylo nutné implementovat

přímo do roscore. Tyto úpravy by měli fungovat na většině verzí ROS, jelikož část roscore obstarávající komunikaci nebyla modifikována už velmi dlouho. Samotnou implementací se zabývat nebudeme, jelikož to není předmětem této práce.

Předchozí způsob zabezpečení na úrovni aplikace nás sice ochrání před množstvím bezpečnostních problémů, ale má bohužel tu nevýhodu, že všechny moduly musí být znovu zkompilovány. Je to z toho důvodu, že každý uzel musí implementovat bezpečnostní funkce. Jedním ze způsobů, jak tento problém odstranit, je provedením úpravy komunikace přímo mezi dvěma ROS uzly. To lze udělat typicky přidáním autentizačního mechanismu. Tím zajistíme některé z klíčových aspektů bezpečnosti, hlavně důvěrnost.

Zde představený způsob zabezpečení komunikačního kanálu je založený na použití protokolů TLS (Transport Layer Security) pro TCP a DTLS (Datagram Transport Layer Security) pro UDP. Toto je možné provést ve třech krocích. Prvním krokem provedením dodatečného handshake se vzájemnou autentizací a autorizací založenou na veřejném klíči a certifikátech. K zabezpečení další komunikace se využívá symetrické šifrování (AES-256), které zajistí důvěrnost dat. Integritu dat zajistí použití tzv. MAC (Message Authentication Code) řetězců.

MAC je symetrická kryptografická funkce, podobná hashovacím funkcím, která slouží k šifrování zpráv předem známým algoritmem s využitím předdefinovaného soukromého klíče. Právě využitím klíče se liší oproti běžným hashovacím funkcím, které ho nevyužívají a tím pádem nepodporují autentizaci dat, ale pouze jejich integritu.

Aby se zajistilo, že k topics budou mít přístup jen důvěryhodné uzly, je potřeba po provedení TLS/DTLS handshake ještě provést autorizaci každého uzlu. K tomu lze použít X.509 certifikáty, které mají možnost zakódovat veřejné klíče, identity uzlů nebo informace o tom, jaké uzly mají jaká práva (např. na jaké topics mohou publikovat nebo ze kterých mohou odebírat).

Změny v implementaci ROS core jsou provedené tak, aby se projevily hned po tom, co klient začne komunikovat (např. pokud požaduje od jiného klienta nějakou službu). Komunikace probíhá následujícím způsobem v případě, pokud nevyužíváme bezpečnostní rozšíření popsané níže. Důvodem k začátku komunikace v ROS je obvykle to, že nějaký uzel chce odebírat nebo publikovat, Jako první se provede počáteční zavolán XMLRPC. Volání jde směrem od odběratele k vydavateli. Klientský uzel obdrží informace o tomto volání od ROS mastera. Toto volání XMLRCP se považuje za první handshake mezi dvěma uzly

a provádí se z důvodu výměny informací ohledně budoucí komunikace pomocí protokolů TCP nebo UDP.

Uzly potvrdí, že dotazovaný topic nebo název služby odpovídá a poté si vymění informace o portu pro TCP/UDP spojení. V případě TCP komunikace je poté vytvořen TCP kanál mezi těmito uzly. V případě UDP vydavatele pouze rozesílá UDP datagramy v případě, že jsou nová data k dispozici.

Při použití zabezpečeného komunikačního kanálu si uzly po navázání TLS/DTLS spojení vymění příslušné certifikáty, které mají za úkol zajistit autorizaci každého uzlu pro daný typ komunikace. Odběratel například kontroluje, zda data, která nabízí vydavatele mohou být zpracována a přijata jiným uzlem. Vydavatele naopak zase kontroluje, jestli odběratel může přijmout jeho data atd.

Pokud ověření certifikátu uzlu selže, nelze provádět žádnou další komunikaci. Tuto proceduru by bylo nutné doimplementovat přímo do TCP/UDP kanálu, který je implementovaný v balíčku roscpp. Jak pro TCP, tak pro UDP se ještě před výměnou hlaviček a zahájením vlastní komunikace provede počáteční handshake.

Výhodou zde představeného řešení zabezpečení na úrovni komunikace je vysoká odolnost vůči DoS útokům a také možnost používat již existující uzly bez potřeby je rekompilovat. Zajištěná je navíc důvěrnost, integrita i dostupnost. Všechny uzly potřebují platné certifikáty, aby bylo možné při začátku komunikace provést TLS/DTLS handshake. Pokud certifikát chybí, není možné kontaktovat žádný z uzlů.

Nevýhodou je naopak to, že toto řešení nezabezpečuje XMPRPC API. Útočník díky tomu může stále získat přístup ke grafu uzlů, vypínat uzly nebo posílat zprávy typu publisherUpdate, které slouží k informování o tom, když se objeví nový vydavatele. Určitou nevýhodou je i to, že strana ROS mastera v této API je implementovaná v Pythonu.

Pokud bychom chtěli tedy dosáhnout vyššího stupně bezpečnosti, bylo by nutné modifikovat jak roscpp, tak rospy balíčky. Další možností by bylo zkombinovat modifikaci roscpp s použitím SROS, který je představen .v kapitole 7.1. Toto nám zajistí zabezpečení i v případě, že použijeme uzly psané v Pythonu.

6.3.1 TCP handshake

V případě TCP je tento počáteční handshake určen k výměně hlaviček a dat z topics obsahujících definice zpráv, topics a ID volajících. Tím, že se TLS handshake provede

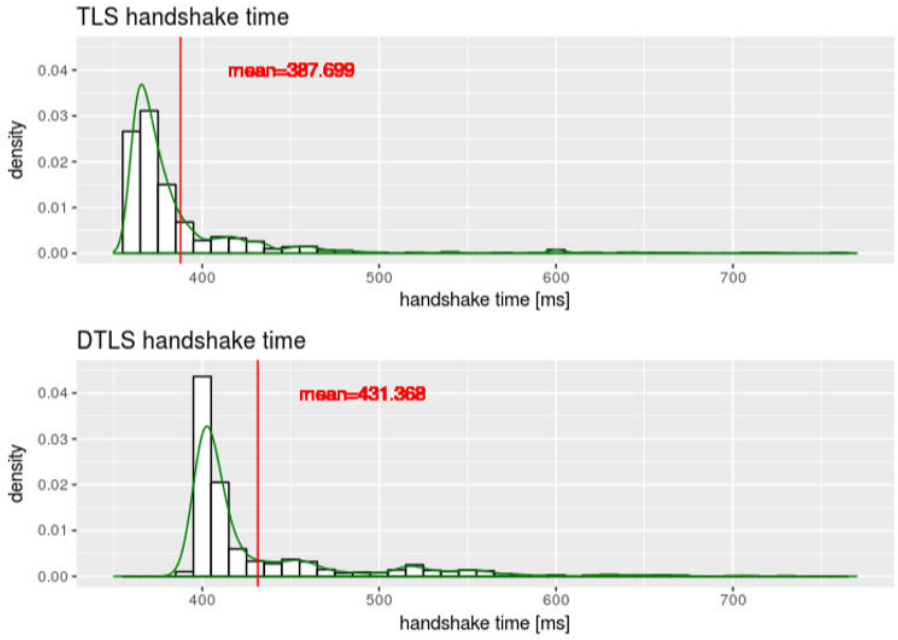
jako první zároveň zajištěno i provedení TCP handshake komunikačního protokolu, což bylo prvním krokem na původním diagramu (*obr. 3* v kapitole **5.1**). Poté jsou nad všemi daty, která se posílají během komunikace, provedeny všechny potřebné kroky pro šifrování a dešifrování. Pro provedení TLS handshake, autentizace a autorizace jsou použity certifikáty X.509.

6.3.2 UDP handshake

V případě, že využíváme nemodifikovanou verzi ROS bez jakýchkoli bezpečnostních opatření, probíhá komunikace v UDP následujícím způsobem. Uzel, který chce publikovat, začne rozesílat UDP datagramy obsahující informace o topics odběratelům. Tyto datagramy se rozesílají pokaždé, když se publikují nová data. Nevýhodou tohoto řešení je, že komunikace je pouze jednosměrná od vydavatele k odběrateli.

My bychom potřebovali, aby komunikace byla obousměrná, a to z důvodu, že je to potřeba pro challenge-response protokol, který bychom využili pro zabezpečení kanálu pomocí DTLS handshake, což je obdoba TLS pro protokol UDP. Uzel, který chce publikovat, musí otevřít UDP server socket pro příjem datagramů od odběratele. Aby se zabezpečil i první datagram jdoucí k odběrateli, provede se DTLS handshake paralelně s voláním XMLRPC, které jde opačně (směrem od odběratele).

Volání XMLRPC je synchronizované s DTLS handshake, tudíž není možné, aby volání skončilo ještě před tím, než je handshake kompletní. Po provedení všech těchto kroků je komunikace opět jednosměrná.



Obr. 5 – Grafy porovnání časů potřebných k provedení TLS/DTLS handshake.

Zdroj: https://www.researchgate.net/figure/A-histogram-of-the-time-required-to-perform-a-TLS-or-DTLS-handshake-respectively-1000_fig2_314082519

Obr. 6 – Diagram ukazující průběh komunikace mezi dvěma uzly. zabezpečenou komunikací. Modře je indikovaný nový zabezpečený průběh komunikace, červeně starý. Opět nalevo TCP a napravo UDP.

Zdroj: https://www.researchgate.net/figure/A-sequence-diagram-showing-the-changed-communication-pattern-between-two-nodes-Green_fig1_314082519

7. Další možnosti zabezpečení ROS aplikací

7.1 SROS

Prvním navrhovaným bezpečnostním rozšířením je SROS, což je zkratka pro Securing ROS. SROS byl poprvé představen na konferenci ROSCon v roce 2016. Toto rozšíření ROS API a ekosystému by mělo vyřešit některé z nyní známých bezpečnostních problémů díky tomu, že disponuje množstvím různých bezpečnostních opatření. Bohužel největší nevýhodou je, že SROS zatím nepodporuje ROS C++ knihovnu, takže je možné pro něj využívat jen uzly psané v Pythonu.

I když je SROS stále vyvíjen a jedná se tedy spíše o experimentální záležitost, můžeme zde najít například podporu moderní kryptografie, jako například nativní podporu kryptografického protokolu Secure Sockets Layer (SSL), konkrétněji jeho nástupce Transport Layer Security (TLS) pro veškerou komunikaci na úrovni IP/socket.

To znamená, že veškerá síťová komunikace v SROS je šifrovaná. Použití TLS dále podporuje integritu dat, autentizovanou identitu a zvyšuje soukromí. SROS také využívá standard X.509. Jedná se o kryptografický standard, určený pro systémy založené na veřejném klíči, jehož výhodou je jednoduché podepisování.

SROS poskytuje při spouštění server, který slouží ke generování Public Key Infrastructure (PKI) prvků, jako jsou asymetrické klíče, certifikáty a také API sloužící k distribuci šifrovaných certifikátů uzlům. Tento server je nezávislý na ROS díky tomu, že je kompletně oddělen od roscore. Tím pádem může být spuštěn kdekoliv v síti nebo být kompletně offline, což zvyšuje potřebu existence certifikační autority (CA) na robotovi, protože certifikáty jsou snadno padělatelné.

Server jinak ale výrazně ulehčuje používání i vývoj systémů podporujících SROS a také poskytuje konzervativní defaultní nastavení bezpečnosti, které si uživatel může podle potřeby upravit. Upravit lze například restrikce, informace o CA, algoritmus pro generování klíčů, bitovou délku atd. SROS dále umožňuje definovat tzv. globbing pro přiřazování rolí jednotlivým uzlům nebo restrikcím uzlů. Globbing slouží k usnadnění práce s výrazy, které jsou si podobné, s výrazy obsahujícími nějaký konkrétní znak nebo řetězec znaků apod. Najdeme zde také uživatelské nástroje pro automatické generování veřejného a soukromého páru klíčů pro uzel nebo politiky pro řízení přístupu.

Nejnovějším doplňkem pro SROS je přidání AppArmor profilů, což jsou Linuxové bezpečnostní moduly, které umožňují administrátorovi omezit možnosti programů pomocí profilů jednotlivých programů. Jedná se o software mezi aplikací a systémem. V profilu lze aplikaci přesně vymezit, co může dělat a co ne, čímž chráníme systém před zneužitím různých bezpečnostních děr, které ještě nebyly zjištěny nebo opraveny. Profily mohou povolit věci, jako je přístup k síti, přístup k socketu, oprávnění ke čtení, zápisu nebo spouštění souborů na odpovídajících cestách.

7.2 SRI's Secure ROS

Další alternativou pro bezpečnější používání ROS je Secure ROS od společnosti SRI International. Co se týče implementace, tak existuje určitá podobnost mezi Secure ROS a SROS zmíněným výše. Jinak se ale jedná o dva rozdílné systémy.

První výhodou oproti SROS je podpora rospy i roscpp knihoven. Je tedy možné psát kódy uzlů jak v C++, tak v Pythonu. Secure ROS je oproti SROS jednodušší na instalaci a je tak vhodnější po běžného uživatele, který musí jen poskytnout konfigurační soubor s definovanými pravidly. Secure ROS poskytuje alternativní verze ROS core balíčků, které umožňují zabezpečenou komunikaci mezi ROS uzly. Modifikované balíčky jsou rosmaster, rosgraph, roscpp, rospy, xmlrpcpp a nodelet.

Kromě modifikovaných balíčků Secure ROS používá v transportním módu ještě IPSec. Jedná se o bezpečnostní rozšíření protokolu IP založené na autentizaci a šifrování IP paketů a datagramů. V Secure ROS se dále mohou k topics připojit pouze autorizované uzly. Které uzly mají oprávnění k publikování nebo odběru zpráv může při běhu specifikovat uživatel v konfiguračním souboru YAML. V tomto souboru je dále možné specifikovat parametry (gettery a settery) a poskytovatele (servery) a žadatele (klienty) o služby.

Pokud není k dispozici žádný YAML konfigurační soubor, chová se Secure ROS jako normální ROS. Nevýhodou Secure ROS je, že neposkytuje formální ověřovací prostředky, které zaručují, že požadované vlastnosti odpovídají specifikacím.

7.3 Secure-ROS-Transport

Ještě před uvedením secure-ros-transport navrhli jeho vývojáři Bernhard Dieber a spol. bezpečnostní architekturu na aplikační úrovni. Tato architektura umožňovala zabezpečenou komunikaci mezi ROS uzly a zajišťovala, že pouze validní, registrované uzly jsou autorizované k provedení určitých operací díky využití dedikovaného autentizačního serveru

a certifikátůx.509 pro autentizaci, které potvrzují, že klíč patří uvedenému vlastníkovi. Každý uzel využívající tento certifikát má své unikátní sériové číslo. Uzly se autentizují tím, že poskytnou svůj certifikát, čímž se ověří, jestli jsou uzel autorizovány k provedení dané operace. Využité kryptografické metody, měly zajišťovat důvěrnost a integritu dat, čímž by se eliminovaly některé z největších bezpečnostních hrozeb, kterými ROS disponuje.

Bohužel jednou z nevýhod této architektury bylo to, že bylo nutné manuální generování certifikátů, jejich distribuce uzlům a registrování seznamu uzlů na autentizačním serveru. Největší nevýhodou ale bylo to, že tato architektura byla myšlena pro použití na aplikační úrovni a tím pádem zůstávaly pořád některé nevyřešené bezpečnostní hrozby, které na této úrovni nejdou vyřešit.

Může jít třeba o náhodný přístup k datům při odebírání nebo citlivost vůči DoS útokům, které jdou provést pomocí opakovaného publikování zpráv, které zahltí odebírající uzly. Jako důsledek toho autoři přišli s vylepšením tohoto rozšíření založeném na modifikací ROS core balíčků, která by měla zvýšit celkovou bezpečnost ROS. Jedná se v podstatě zabezpečený komunikační kanál umožňující uzlům komunikovat na bázi peer-to-peer a zároveň zachovat důvěrnost a integritu dat. Tato modifikace se označuje jako secure-ros-transport.

Autoři využili TLS pro protokol TCP a DTLS (Datagram TLS) pro UDP k zabezpečení komunikace mezi uzly a ROS masterem tím, že přidali dodatečný handshake krok a zpřísnili pravidla pro autorizaci k přístupu k jednotlivým topics. Tento přístup redukuje možnost provedení DoS útoků a náhodnému přístupu (publikování/odběru) ke zprávám.

Nicméně, jak autoři již naznačili, sám ROS master momentálně nedisponuje žádným zabezpečením a stále přenáší informace o uzlech a topics kterékoli entitě, která si o tyto informace požádá. Může se jednat třeba o rosnode list, rostopics list atd. Stále je také možné externě shodit jakýkoli uzel s využitím protokolu XML-RPC, pomocím něhož jde provádět vzdálené volání procedur

7.4 Rosauth

Posledním bezpečnostním rozšířením, které si zde představíme je Rosauth (dříve Rosbridge). Jedná se o nástroj vytvořený komunitou ROS a vydaný pod licencí BSD, sloužící pro autentizaci a autorizaci klientů na straně serveru. Důraz je kladen především na bezpečnost při připojování vzdálených klientů, kterým může být libovolný robot běžící na ROS.

Autentizační schéma využívá token sloužící k autentizaci připojujících se vzdálených klientů. Toto schéma zajišťuje, že se k systému může připojit pouze klient, jehož identita byla ověřena z důvěrného, externího, autentizačního zdroje, který je již integrovaný jako součást rosbridge protokolu. Tento externí systém má za úkol starat se o správu uživatelů. Pro autentizaci zpráv se využívají již dříve zmíněné MAC řetězce a provádí se v uzlu ros_mac_authentication.

Na straně serveru je uloženo několik klíčů. Příchozí zpráva, která je zašifrovaná se porovná s výsledkem hashovací funkce a k přijetí zprávy na straně serveru dojde pouze tehdy, pokud se obě zprávy shodují. Pokud ke shodě nedojde, zpráva se označí jako z nevěrohodného zdroje a zahodí se.

Rosauth dále využívá Protokol SSL, který s využitím certifikátů od ověřených certifikačních autorit zajistí, že externí klienti mají jistotu, že daný ROS je legitimní. Protokol také zajišťuje důvěrnost, integritu a autentičnost jednotlivých paketů. Rosauth nyní disponuje kromě autentizace i autorizačními úrovněmi klientů, které ještě do nedávna neměl. Už se tedy nestane to, že by klient, který byl pouze autentizovaný, měl hned přístup k celému systému. Toho šlo zneužít k celé řadě útoků, protože klient mohl víceméně bez omezení posílat příkazy robotovi.

8. Útoky na ROS - Praktická část

V první části této kapitoly si podrobně popíšeme a prakticky ukážeme, jak provádět penetrační testování ROS pomocí nástroje zvaného ROSPenTo. Pomocí ROSPenTo si prakticky ukážeme například, jak analyzovat ROS systém, jak zjistit jazyk, ve kterém jsou uzly napsané, také si ukážeme útok typu stealth publisher, pomocí kterého podvrhneme odběrateli falešná data, které bude odebírat, aniž by si toho kdokoli všiml. Také si ukážeme, jak lze izolovat služby, aby byly nedostupné jiným ROS uzlům. Ve druhé části si pak ukážeme, jak provést odposlech pomocí ARP spoofingu. Nejdříve si ale představíme nástroj ROSPenTo.

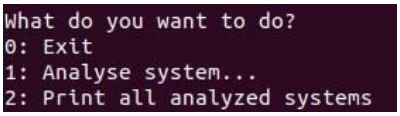
8.1 ROSPenTo

ROSPenTo je penetrační nástroj založený na .NET frameworku, který lze použít k analýze spuštěných ROS aplikací a manipulaci s nimi. ROSPenTo umožňuje běh na libovolné .NET-enabled platformě, včetně open-source platformy Mono. ROSPenTo je schopen analyzovat více ROS aplikací současně, čehož lze později využít k manipulaci s jednotlivými aplikacemi a k reorganizaci jejich uzlů.

ROSPenTo lze spustit dvěma různými způsoby: s argumenty příkazového řádku a bez argumentů. Pokud nepředáváme žádné argumenty při spuštění ROSPenTo, tak se aplikace spustí v interaktivním režimu a uživatel si může vybrat procedury, které mají být provedeny. Při spuštění s argumenty (tedy v ne-interaktivním režimu) aplikace provádí jen jednu úlohu v závislosti na předaných argumentech příkazového řádku.

8.1.1 ROSPenTo - Interaktivní mód

Spuštění aplikace v interaktivním režimu provedeme pomocí příkazu *$ mono RosPenTo.exe*. Poté se nám zobrazí následující menu, ve kterém si můžeme vybrat, co chceme dělat dál



Obr. 7 - Úvodní menu ROSPenTo.

- Možnost 0 jednoduše jen ukončí program.
- Možnost 1 slouží k analýze systému. Aby tato analýza mohla proběhnout, potřebuje si program zjistit informace o ROS systému, určeného k analýze. Proto potřebuje URI

ROS Mastera. Ten Poskytuje informace o běžících uzlech, topics, přes které je možné komunikovat, službách a uložených parametrech-

- Možnost 2 vypíše seznam všech analyzovaných ROS systémů. ROS systém je reprezentovaný jedinečným číslem a URI ROS Mastera.

Pokud zvolíme možnost 1 (analýzu systému), zobrazí se nám další menu, tentokrát se 16 položkami.

```
What do you want to do?
0: Exit
1: Analyse system...
2: Print all analyzed systems
3: Print information about analyzed system...
4: Print nodes of analyzed system...
5: Print node types of analyzed system (Python or C++)...
6: Print topics of analyzed system...
7: Print services of analyzed system...
8: Print communications of analyzed system...
9: Print communications of topic...
10: Print parameters...
11: Update publishers list of subscriber (add)...
12: Update publishers list of subscriber (set)...
13: Update publishers list of subscriber (remove)...
14: Isolate service...
15: Unsubscribe node from parameter (only C++)...
16: Update subscribed parameter at Node (only C++)...
```

Obr. 8 - Rozšířené menu po analýze systému.

Tři tečky za konci řádků indikují, že je potřeba dodatečný vstup od uživatele. První 2 možnosti jsme si už popsali. Nyní se podíváme, co dělají možnosti 3-16.

- Možnost 3 vypíše informace o běžících uzlech, topics, službách a uložených parametrech.
- Možnost 4 vypíše seznam všech spuštěných uzlů s unikátním ID, jménem a URI uzlu.
- Možnost 5 vypíše, jestli je uzel implementován v Pythonu nebo v C++.
- Možnost 6 vypíše seznam všech uzlů, které se účastní na komunikaci mezi uzly.
- Možnost 7 vypíše všechny dostupné služby v ROS systému.
- Možnost 8 vypíše seznam všech komunikačních vztahů. Každý komunikační vztah obsahuje jednoho nebo více vydavatelů, kteří publikují data na specifický topic pro jednoho nebo více odběratelů.
- Možnost 9 vypíše jediný komunikační vztah, pro specifický topic, který musí být definovaný uživatelským vstupem.

- Možnost 10 vypíše všechny parametry uložené v ROS systému.
- Možnost 11 přidá nového vydavatele do komunikačního vztahu s odběratelem. Dojde k aktualizaci seznamu vydavatelů pro konkrétní odběratele, což umožní, že si mohou posílat data.
- Možnost 12 je stejná, jako předchozí, s tou výjimkou, že vydavatelé jsou explicitně přiřazeni odběratelům. To znamená, že všichni existující vydavatelé budou přepsáni.
- Možnost 13 odstraní vydavatele ze seznamu. Odběratelé tak nebudou dostávat žádná data od odebraného vydavatele.
- Možnost 14 zruší registraci služby u ROS Mastera. Služba je ale stále dostupná u poskytovatele služby. ROS Master už jen nepředává informace o službě ostatním uzlům.
- Možnost 15 zruší odebírání parametru uzle. Uzle tedy nebude dostávat žádné další informace o aktualizacích parametru
- Možnost 16 aktualizuje parametr pro jeden konkrétní uzel v ROS systému. Tato možnost funguje pouze u uzlů napsaných v C++.

8.1.2 ROSPenTo - Ne-interaktivní mód

Ne-interaktivní mód ROSPenTo provádí jeden konkrétní úkol v závislosti na argumentech předaném příkazovou řádkou a poté končí. V současné době jsou dostupné pouze funkce typu publisher update podobné možnostem 11-13 z interaktivního režimu. Je ale možné, že se v dalších verzích ROSPenTo dočkáme nějakých rozšíření. Chceme-li provést publisher update v ne-interaktivním režimu, musíme pomocí příkazové řádky předat několik argumentů.

--t nebo --target: (povinný parametr). ROS Master URI systému, na který chceme útočit.

--p nebo --pentest: (povinný parametr). ROS Master URI systému, ze kterého útočíme.

--sub: (povinný parametr) Jméno cílového odběratele, na kterého útočíme.

--top: (povinný parametr) Jméno cílového topic.

--pub: (povinný parametr) Jméno nového vydavatele v cílovém systému.

--add: () V příkazu publisherUpdate přidá vydavatele k již existujícím.

--set: () V příkazu publisherUpdate nastaví nového vydavatele.

--remove: ()V příkazu publisherUpdate odebere vydavatele ze stávajících.

Na pořadí parametrů nezáleží, nicméně tam, kde je to uvedeno, je nutné za parametrem poskytnout korespondující hodnotu. Ze zbylých tří parametrů, které nepřijímají žádnou hodnotu, je vyžadován alespoň jeden z nich. Například následujícím příkazem můžeme přidat do systému nového vydavatele ***/útočník*** , který bude komunikovat pomocí ***/topic*** s odběratelem ***/oběť***.Můžeme si všimnout, že oběť se nachází na systému -t s defaultním číslem portu ROS-mastera (11311), zatímco útočník se nachází na systému -p s číslem portu ROS-mastera o jedno vyšší (11312)

-t http://localhost:11311 --sub /oběť --top /topic -p http://localhost:11312 --pub /útočník --add

8.2 Ukázky útoků pomocí ROSPenTo

V této sekci si ukážeme nějaké útoky pomocí penetračního nástroje ROSPenTo. Ještě předtím si ale vysvětlíme, jakým způsobem ROSPenTo adresuje entity v analyzovaném ROS systému, tak abychom se ve všem vyznali. Toto je důležité pro efektivní provádění útoků na ROS, protože nějaké ROS systémy (také zvané ROS sítě) mohou být velice rozsáhlé a je tak dobré mít ve věcech trochu pořádek.

ROSPenTo dokáže rozeznat tři druhy entit a to uzel, topic a službu. Každé entitě je pak přiděleno číslo, které je v rámci entity unikátní a toto číslo se páruje ještě s číslem systému. Výsledek má tedy běžně formu X.Y, kde X je číslo systému a Y číslo entity. Prakticky to pak může vypadat tak, že například máme topic 0.1, což je první nalezený topic v prvním analyzovaném systému.

Důležité je ale, že identifikátor entity (v tomto případě 0.1) musí být unikátní pouze v rámci entity v systému, to tedy znamená, že další topic 0.1 již existovat nemůže, nicméně může existoval třeba služba 0.1 nebo uzel 0.1. To, o jaký druh entity se jedná není nutné nijak explicitně specifikovat, protože ROSPenTo se vždy ptá na uživatelský vstup pouze v rámci dané entity. ROSPenTo se nás například zeptá "jakou službu chcete izolovat?" a my zadáme jen 0.1 a tím pádem je jasné, že se jedná o službu 0.1 a ne o něco dalšího.

8.2.1 Analýza ROS sítě

První si ukážeme, jak analyzovat ROS síť a jaké užitečné informace nám analýza prozradí. Pro demonstraci následujících útoků budeme používat jednoduché ROS aplikace ***roscpp_tutorials*** a ***rospy_tutorials***, které jsou již obsažené v instalaci ROS, takže není nutné je stahovat zvlášť.

Nejdříve si spustíme uzel talker pomocí příkazu r***osrun roscpp_tutorials talker***. Poté si v jiném okně přepneme do adresáře, ve kterém se nachází nástroj ROSPenTo a spustíme jej pomocí příkazu ***$ mono RosPenTo.exe***. V dalším kroku zvolíme možnost 1 (analýza sítě) a zobrazí se nám dotaz na zadání ROS-master URI.

Tady se chvíli pozastavíme, protože toto URI je ta nejdůležitější věc, kterou potřebujeme znát, pokud chceme útočit na ROS pomocí ROSPenTo. Pokud URI známe, tak máme alespoň částečně vyhráno. Tak tomu ale v praxi většinou není. Naštěstí zjištění ROS-master URI by neměl být problém. Toto URI se skládá z IP adresy zařízení, na kterém běží ROS-master a čísla portu.

IP adresu můžeme jednoduše zjistit pomocí nějakého volně dostupného skeneru sítě. Například pomocí aplikace Fing pro Android. ta nám zobrazí i názvy zařízení v síti, takže pokud například ROS běží na Raspberry-pi, tak nám aplikace vedle nalezené IP adresy zobrazí i Raspberry-pi. Další možnost, jak zjistit zařízení v síti je například využít příkaz ***netdiscover*** v Kali Linuxu.

Co se týče portu, tak tam máme v podstatě dvě možnosti. První je zkusit defaultní port ROS-mastera, což je port 11311. Mastera jde sice spustit i na jiném portu, ale to musí uživatel ROS systému explicitně uvést při spuštění a je tak velká pravděpodobnost, že master poběží na defaultním portu.

Pokud ale přeci jenom na defaultním portu nepoběží, máme druhou možnost a to použít opět nějaký skener. Již dříve zmíněná aplikace Fing dokáže skenovat i porty. Bohužel v tom ale není moc spolehlivá, takže zde bych doporučil nějaký jiný skener sítě, např. Angry IP Scanner. Až nám sken vrátí seznam otevřených portů na zařízení, budeme se soustředit na nějaká nezvyklá čísla portů, tzn. jiná, než 80, 53 apod. Pokud tam nezvyklých čísel portů bude víc, nezbývá nic, než je zkoušet v ROSPenTo jedno po druhém, dokud neuspějeme. To poznáme tak, že nám ROSPenTo vrátí analýzu systému a ne chybovou hlášku.

Analýza sítě nám poskytne spoustu užitečných informací o systému, jako třeba všechny komponenty systému, tzn. uzly, topics, služby a také komunikační vztahy a parametry. Jak již bylo zmíněno dříve, každém entitě je přiřazeno číslo, které je v rámci entity v systému unikátní. Dále zde nalezneme URI XML-RPC dotazu pro každý uzel. V prvním bloku výstupu jsou zobrazeny všechny uzly v systému. V dalších blocích jsou všechny topics a služby. Zde jsou také v závorkách zobrazeny typy zpráv pro každý topic. Jako další jsou zobrazeny komunikační vztahy.

ROSPenTo vypisuje všechny spojení mezi vydavateli a odběrateli. Na posledním místě jsou vypsány parametry. Takto vypadá výstup analýzy systému po spuštění uzlu talker. Můžeme vidět, že komunikace 0.0 se neúčastní žádný odběratel. To proto, že jsme ještě nespustili listener.

```
System 0: http://10.42.0.1:11311/
Nodes:
        Node 0.1: /rosout (XmlRpcUri: http://10.42.0.1:35335/)
        Node 0.0: /talker (XmlRpcUri: http://10.42.0.1:45755/)
Topics:
        Topic 0.0: /chatter (Type: std_msgs/String)
        Topic 0.1: /rosout (Type: rosgraph_msgs/Log)
        Topic 0.2: /rosout_agg (Type: rosgraph_msgs/Log)
Services:
        Service 0.3: /rosout/get_loggers
        Service 0.2: /rosout/set_logger_level
        Service 0.0: /talker/get_loggers
        Service 0.1: /talker/set_logger_level
Communications:
        Communication 0.0:
                Publishers:
                        Node 0.0: /talker (XmlRpcUri: http://10.42.0.1:45755/)
                Topic 0.0: /chatter (Type: std_msgs/String)
                Subscribers:
        Communication 0.1:
                Publishers:
                        Node 0.0: /talker (XmlRpcUri: http://10.42.0.1:45755/)
                Topic 0.1: /rosout (Type: rosgraph_msgs/Log)
                Subscribers:
                        Node 0.1: /rosout (XmlRpcUri: http://10.42.0.1:35335/)
        Communication 0.2:
                Publishers:
                        Node 0.1: /rosout (XmlRpcUri: http://10.42.0.1:35335/)
                Topic 0.2: /rosout_agg (Type: rosgraph_msgs/Log)
                Subscribers:
Parameters:
        Parameter 0.0:
                Name: /roslaunch/uris/host_ubiquityrobot_local__33677
        Parameter 0.1:
                Name: /rosdistro
        Parameter 0.2:
                Name: /rosversion
        Parameter 0.3:
                Name: /run_id
```

Obr. 9 - Výsledek analýzy systému pomocí ROSPenTo.

Nyní pomocí příkazu ***rosrun roscpp_tutorials listener*** spustíme uzel listener a budeme pozorovat rozdíl po opětovné analýze systému. Jak můžeme vidět do komunikací nám mezi odběratele přibyl uzel 0.1 (listener). Komunikace s uzlem 0.0 (talker) probíhá pomocí topic 0.0 (chatter)

```
Communication 0.0:
        Publishers:
                Node 0.0: /talker (XmlRpcUri: http://10.42.0.1:45755/)
        Topic 0.0: /chatter (Type: std_msgs/String)
        Subscribers:
                Node 0.1: /listener (XmlRpcUri: http://10.42.0.1:36235/)
```

Obr. 10 - Vypsané komunikace v rámci ROS systému.

8.2.2 Analýza typu uzlů

Další užitečná informace, která se nám při útočení na ROS může hodit, a kterou dokážeme pomocí ROSPenTo zjistit, je programovací jazyk, pomocí kterého je uzel implementován.

Implementace ROS-master a slave API pomocí C++ a Pythonu jsou navzájem kompatibilní, nicméně samotný kód se samozřejmě musí v nějakých ohledech lišit a díky tomu se mohou lišit i potenciální zranitelnosti.

ROSPenTo dokáže zjistit použitý programovací jazyk tak, že zavolá XML_RPC metodu *getName* a pak analyzuje odpověď. Tato metoda je totiž implementována pouze v Pythonu a při jejím úspěšném zavolání vrací tři údaje a to stavový kód, prázdnou stavovou zprávu a jméno uzlu. Pokud je uzel implementován pomocí C++, tak tuto metodu neposkytuje a tím pádem pokus o její zavolání selže a je vyvolána výjimka. Znalost jazyka použitého k implementaci uzlu nám může pomoci zneužít specifické zranitelnosti systému.

Opět si spustíme uzly talker a listener, jako v předchozím příkladu. Teď ovšem spustím uzel talker implementovaný v C++ pomocí příkazu ***roscpp_tutorials talker***, a uzel listener implementovaný v Pythonu pomocí příkazu ***rospy_tutorials listener***. Dále spustíme ROSPenTo a provedeme analýzu sítě, tak jako v minulém příkladu. Můžeme vidět, že se v systému nachází celkem 3 uzly. Kromě námi spuštěných uzlu listener a talker se v systému nachází ještě uzel rosout, který nás ale momentálně nezajímá.

```
System 0: http://10.42.0.146:11311/
Nodes:
        Node 0.1: /listener_2281_1601660165128 (XmlRpcUri: http://10.42.0.146:40855/)
        Node 0.2: /rosout (XmlRpcUri: http://10.42.0.146:38687/)
        Node 0.0: /talker (XmlRpcUri: http://10.42.0.146:44507/)
```

Obr. 11 - Uzly nacházející se v analyzovaném ROS systému.

Dále zvolíme možnost 5 pro analýzu typu uzlů a zadáme číslo systému 0. ROSPenTo nám vypíše v jakém jazyce jsou všechny uzly v systému naimplementovány. Když porovnáme čísla uzlů s předchozím obrázkem můžeme vidět, že uzel 0.0 (talker) je opravdu implementován v C++ a uzel 0.1 (listener) je implementován v Pythonu, což odpovídá tomu, jak jsme uzly na začátku spustili.

```
5
Please enter number of analysed system:
0
Node 0.0: C++
Node 0.1: Python
Node 0.2: C++
```

Obr. 12 - Výsledek analýzy typu uzlů.

8.2.3 Přerušení komunikace mezi dvěma vzdálenými uzly

V předchozích částech jsme si ukázali, jak získat nějaké užitečné informace o ROS systému. Teď znalost těchto informací využijeme k manipulaci se systémem. Konkrétně si ukážeme,

jak odříznout naslouchající uzel listener od dat, které publikuje uzel talker. Opět si nastartujeme uzly /talker a /listener tak, jako v předchozích příkladech. Tentokrát je jedno, jestli použijeme uzly implementované v C++ nebo v Pythonu. Oba uzly spustíme na stejném ROS systému. V jednom okně můžeme vidět zprávy, které odesílá talker a ve druhém ty samé zprávy, které přijímá listener a vypisuje je.

Nyní je čas spustit nástroj ROSPenTo. Aby tento útok měl nějaký smysl měli bychom nástroj spustit na úplně jiném zařízení, něž na kterém běží uzly /talker a /listener. Je to samozřejmě z toho důvodu, abychom nasimulovali co nejrealističtější podmínky pro útok, protože v reálu vždy budeme útočit z jiného zařízení.

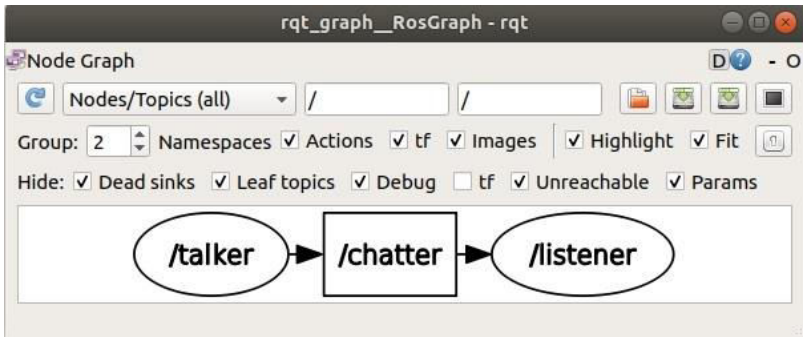
Jako první provedeme analýzu systému zadáním ROS Master URI a dále zvolíme možnost 13 (Update publishers list of subscriber (remove)), která slouží k odebrání vydavatele ze seznamu odběratele, což vlastně znamená roztrhnutí komunikačního vztahu mezi uzly. V zápětí dostaneme dotaz, kterému odběrateli chceme poslat zprávu ***publisherUpdate***. Zde napíšeme 0.1, čímž zvolíme uzel */listener*. Dále se nás ROSPenTo zeptá, který topic by měl být ovlivněn. Zvolíme topic */chatter* pomocí toho, že zadáme jeho číslo 0.0. Jako poslední zbývá zvolit, kterého vydavatele chceme odebrat ze seznamu. Zde zadáme také číslo 0.0, což je identifikátor uzlu */talker*.

```
13
To which subscriber do you want to send the publisherUpdate message?
Please enter number of subscriber (e.g.: 0.0):
0.1
Which topic should be affected?
Please enter number of topic (e.g.: 0.0):
0.0
Which publisher(s) do you want to remove?
Please enter number of publisher(s) (e.g.: 0.0,0.1,...):
0.0
sending publisherUpdate to subscriber '/listener (XmlRpcUri: http://192.168.0.106:46273/)'
over topic '/chatter (Type: std_msgs/String)' with publishers ''
PublisherUpdate completed successfully.
```

Obr. 13 - Postup útoku publisherUpdate.

Když se podíváme do okna, ve kterém se nám zobrazovaly zprávy, které přijímal listener, můžeme si všimnout, že se v něm najednou nic nezobrazuje. Je to samozřejmě tím, že jsme odebrali ze seznamu vydavatelů uzel */talker,*který byl zdrojem těchto zpráv. ROSPenTo provedl útok tak, že zavolal XML.RPC funkci ***publisherUpdate*** s prázdným seznamem vydavatelů jako parametrem. To způsobilo, že se listener začal domnívat, že pro topic */chatter* nejsou dostupní žádní vydavatelé a ukončil tak spojení.

Z praktického hlediska je zde nejdůležitější věcí pro útočníka fakt, že nic z toho, co jsme provedli nebylo zaznamenáno ROS-Masterem. To si můžeme ověřit třeba tak, že necháme vygenerovat RQT graf před a po provedení útoku. V obou případech bude úplně stejný. RQT graf můžeme jednoduše vygenerovat tak, že si stáhneme potřebný nástroj. To lze pomocí příkazů ***sudo apt-get install ros-distro-rqt*** a ***apt-get install ros-distro-rqt-common-plugins***, kde distro nahradíme naší verzí ROS (např. melodic). Nástroj pak spustíme tak, že v novém terminálu napíšeme příkaz ***rosrun rqt_graph rqt_graph***.



Obr. 14 - Vygenerovaný RQT graf.

V tomto příkladě jsme si útok předvedli na zjednodušené aplikaci, kde spolu pouze komunikují dva uzly a jejich komunikace se vypisuje v terminálu. Nabízí se tedy otázka, co bychom mohli způsobit, kdybychom stejný typ útok provedli na reálném robotovi a jak by vlastně na útok reagoval? Pohyboval by se nekontrolovatelně dál nebo by se zastavil?

Jako konkrétní příklad si můžeme uvést robota MiR 100, což je robotické vozítko, které jde řídit pomocí konzole se dvěma slidery. Jeden slider slouží pro pohyb s vozítkem vpřed a vzad a druhý slider k pohybu vpravo a vlevo. Celá tato řídící konzole je reprezentována, jako jeden uzel. Údaje o pozici obou sliderů jsou publikovány na topic, ze kterého odebírá uzel, který se stará o pohyb vozítka.

Pokud bychom roztrhli komunikaci mezi uzly, tak jako v předchozím příkladě, způsobíme tím to, že se robot jednoduše jenom zastaví. Důvodem, že se robot zastaví, místo toho aby například pokračoval nekontrolovatelně v pohybu je ten, že uzel, který odebírá údaje o pozici sliderů přestane dostávat veškeré informace a tím pádem nemá na základě čeho by s robotem pohyboval. Stejně by tomu bylo u všech robotů fungujících na tomto principu, kdy odběratel řídí robota na základě nějakých informací, které publikuje jiný uzel.

Tento typ útoku by nejspíš ve většině případů nebyl pro oběť fatální, vzhledem k tomu, že robota pouze zastavíme, ale nepřebereme nad ním kontrolu. Vlastně asi to nejhorší, co bychom tímto útokem mohli způsobit je, že oběť zdržíme, popř. zmateme. Důležité je také

podotknout, že tímto útokem robota zastavíme jen dočasně, dokud oběť nerestartuje buď celý systém nebo daný uzel. Při tomto typu útoku si totiž oběť okamžitě všimne, že je něco špatně.

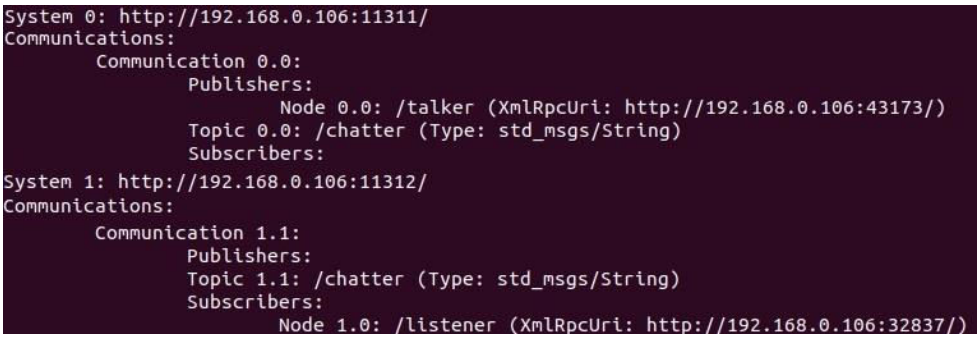
8.2.4 Odposlech komunikace a injekce falešných příkazů

Ve všech předchozích případech jsme pracovali pouze s jedním běžícím ROS systémem. Jak bylo ale zmíněno na začátku, ROSPenTo dokáže manipulovat i s více systémy naráz. V tomto příkladu si ukážeme, jak jeden systém propojit s druhým tak, abychom od něj přijímali zprávy. Jedná se v podstatě o určitý druh odposlechu. Jako první spustíme ROS-master pomocí příkazu ***roscore***. Další, co musíme udělat je spustit druhého ROS mastera, a to buď na jiném počítači třeba na defaultním portu (opět příkaz ***roscore***), nebo na stejném počítači, ale na jiném portu. Pro tento příklad využije druhou možnost, tzn. spustíme druhého ROS mastera na stejném systému.

Důležitý je fakt, že to, jestli použijeme dvě různá fyzická zařízení, nebo pouze jedno se dvěma ROS mastery, nebude mít vliv na postup provádění útoku ani na výsledek. Můžeme použít hned další dostupný port 11312. Spuštění mastera na tomto portu provedeme pomocí příkazu ***roscore -p 11312***. Parametr -p slouží právě k možnosti využít jiný port. Dále spustíme uzel talker na systému s defaultním portem. Ještě než uzel spustíme tak, je potřeba zadat příkaz ***export ROS_MASTER_URI=http://localhost:11311***. To kvůli předchozímu příkazu roscore s pozměněným portem.

Ve druhém okně spustíme uzel listener na námi zvoleném portu 11312. Předtím, než uzel spustíme je opět nutné exportovat master URI pomocí příkazu ***export ROS_MASTER_URI= http://localhost:11312***. Po úspěšném spuštění obou uzlů si můžeme všimnout, že listener nevypisuje žádný výstup. Je to proto, že je spuštěn na jiném ROS systému než talker a uzly tak spolu zatím nedokážou komunikovat.

Detailněji se na to můžeme podívat, pokud provedeme analýzu obou systémů pomocí ROSPenTo a vygenerujeme RQT graf. Z analýzy systémů a vygenerovaného grafu vidíme, že v obou případech je přítomen topic */chatter*, ale komunikace 0.0 v systému 0 (port 11311) nemá žádného odběratele a naopak komunikace 1.1 v systému 1 (port 11312) nemá žádného vydavatele.



Obr. 15 - Komunikace v systému.

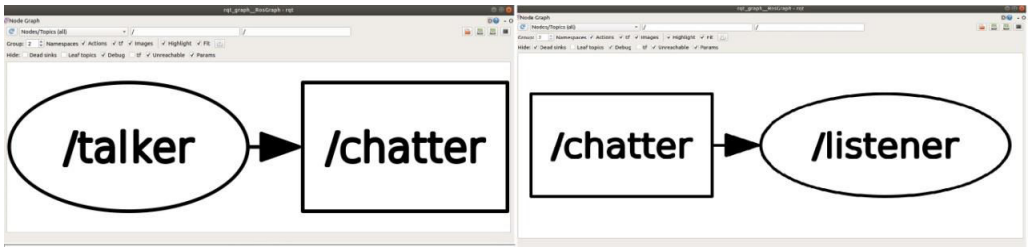
Nyní si ukážeme, jak pomocí ROSPenTo propojit oba systémy tak, aby spolu mohly komunikovat. Pokud jsme již provedli analýzu systémů, zvolíme možnost 11, která umí vytvořit nový komunikační vztah mezi odběratelem a vydavatelem(Update publishers list of subscriber (add)). Poté, co zvolíme tuto možnost, je potřeba zadat identifikátor odběratele, kterému chceme poslat zprávu ***publisherUpdate***. Zadáme číslo 1.0, což je identifikátor uzlu listener.

Dále je potřeba zadat ID topic. Zde zvolíme topic */chatter* pomocí jeho ID 1.1.V tomto kroku je důležité správně zvolit správné ID pro topic */chatter*. Nástroj nám totiž vypíše, že topic */chatter* se nachází v obou analyzovaných systémech a my musíme zvolit ten, který běží na stejném systému, jako uzel */listener*. Pokud zvolíme topic špatně, útok nebude fungovat. Jako poslední je musíme zadat ID vydavatele, kterého chceme přidat do komunikace. My chceme přidat uzel */talker*, jehož ID je 0.0.

Obr. 16 - Postup útoku.

Toto byl poslední krok a my bychom měli konečně vidět nějaký výstup v okně, kde běží listener. Stejně, jako v předchozím případě ROS master nezaznamenal žádnou změnu a vygenerovaný RQT graf je shodný před i po provedení útoku. Jedná se tedy o efektivní způsob, jak odposlouchávat komunikaci v ROS a nebýt odhalen. Samozřejmě za předpokladu, že známe ROS-master URI cílového systému nebo alespoň jeho IP adresu.

Stejně jako v případě přerušení komunikace mezi uzly, ani tímto útokem nejspíš nezpůsobíme nějakou kritickou škodu na cílovém systému. Nicméně takto můžeme získat přístup k cenným informacím, které mohou být dále užitečné pro další typy útoků. Například můžeme získat přístup k nějakým „business critical" informacím, které mohou být zneužity k reverznímu inženýrství aplikace. Také takto můžeme odposlouchávat údaje ze senzorů nebo údaje o pohybu robota, popř. jeho poloze atd. Důležité je také to, že na rozdíl od dříve zmíněného útoku s přerušením komunikace, si zde oběť ničeho nevšimne a komunikace na její straně stále probíhá. Díky tomu můžeme odposlouchávat, jak dlouho chceme, a zároveň se vyhnout detekci.



Obr. 17 - Vygenerovaný RQT graf.

Téměř úplně stejným způsobem, jako odposlech, by mělo být možné i vložení falešných příkazů do cílového systému. Hlavním rozdílem by bylo pouze to, že útočník by v tomto případě pracoval na zařízení, kde běží uzel talker. Oba systémy bychom poté propojili tak, aby uzel listener na systému oběti naslouchal tomu, co publikuje náš „škodlivý" uzel talker, takže bychom mu v podstatě podstrkávali falešné příkazy. V případě odposlechu tomu bylo naopak v tom smyslu, že jsme útok prováděli ze zařízení, kde běžel uzel listener.

Bohužel tento útok ale ne vždy funguje. Například právě na zde představených aplikacích talker a listener útok nefunguje. Pokud stejně jako u odposlechu provedeme analýzu obou systému a poté zadáme korespondující identifikátory uzlů, mezi kterými chceme navázat komunikaci, a topic, přes který chceme komunikovat, tak nám ROSPenTo sice vypíše, že vše proběhlo v pořádku a volání ***publisherUpdate*** se zdařilo, nicméně v okně oběti se nezačne nic vypisovat.

Fakt, že tento typ útoku nefunguje na této demonstrační aplikaci ale neznamená, že nebude fungovat jinde. Například na robotovi MiR 100 zmíněném v minulé kapitole, by útok měl fungovat a útočník by tak měl být schopný kompletně převzat kontrolu nad robotem. Jediný předpoklad k úspěšnému útoku by byl ten, že útočník musí mít na svém zařízení úplně stejný uzel reprezentující konzoli pro ovládání robota, jako má oběť. Vzhledem k tomu, že se ale

jedná o sériově vyráběného robota a ne žádný po domácku vyrobený prototyp, by v tom neměl být problém.

Jediné, co by útočník musel udělat, by bylo spustit na svém zařízení uzel s konzolí, provést analýzu obou systémů a v posledním kroku jen zadat správné identifikátory uzlů a topic. Poté by už měl útočník být schopný robota ovládat z konzole na svém zařízení, zatímco oběť by veškerou kontrolu ztratila. Tento útok tedy už není tak neškodný, jako útok s přerušením komunikace či odposlech, protože útočník si s robotem může dělat, co se mu zachce, zatímco pro oběť neexistuje žádný způsob, jak kontrolu získat zpátky.

8.2.5 Izolování služby od uzlu (odregistrace)

V tomto příkladu si ukážeme, jak využít ROSPenTo k izolování služby v cílovém systému tak, aby naše oběť nemohla službu dále využívat. Jedná se v podstatě o další typ DoS útoku. Stejně, jako v předchozím příkladu spustíme dva různé ROS mastery. Opět je jedno, jestli oba poběží na stejném počítači, nebo na jiných. Důležité je jen vědět jejich URI. Pokud máme oba spuštěné, můžeme přejít k dalšímu kroku, což je spuštění služby ***add_two_ints*** na cílovém systému.

Tato služba, jak už její jméno napovídá, sečte dvě celá čísla a vrátí výsledek. Služba je opět součástí tutoriálů, které jsou k dispozici v každé instalaci ROS, takže nemusíme nic stahovat. Nejdříve exportujeme ROS-master URI pomocí příkazu ***export ROS_MASTER_URI= http://localhost:11311*** a poté spustíme službu příkazem ***rosrun roscpp_tutorials add_two_ints_server***.

Dále jako obvykle spustíme ROSPenTo a analyzujeme oba systémy. Analýza cílového systému ukazuje uzel ***/add_two_ints_server*** a službu ***/add_two_ints***.

```
System 0: http://192.168.0.106:11311/
Nodes:
        Node 0.0: /add_two_ints_server (XmlRpcUri: http://192.168.0.106:34945/)
        Node 0.1: /rosout (XmlRpcUri: http://192.168.0.106:41425/)
Topics:
        Topic 0.0: /rosout (Type: rosgraph_msgs/Log)
        Topic 0.1: /rosout_agg (Type: rosgraph_msgs/Log)
Services:
        Service 0.2: /add_two_ints
        Service 0.0: /add_two_ints_server/get_loggers
        Service 0.1: /add_two_ints_server/set_logger_level
        Service 0.4: /rosout/get_loggers
        Service 0.3: /rosout/set_logger_level
```

Obr. 18 - Analýza cílového systému.

Analýza našeho útočícího systému vypadá následovně.

```
System 1: http://192.168.0.106:11312/
Nodes:
        Node 1.0: /rosout (XmlRpcUri: http://192.168.0.106:39955/)
Topics:
        Topic 1.1: /rosout (Type: rosgraph_msgs/Log)
        Topic 1.0: /rosout_agg (Type: rosgraph_msgs/Log)
Services:
        Service 1.1: /rosout/get_loggers
        Service 1.0: /rosout/set_logger_level
```

Obr. 19 - Analýza našeho systému.

Abychom si mohli otestovat, že služba na cílovém systému funguje, můžeme jí zkusit zavolat příkazem ***rosservice call /add_two_ints x x***, kde místo x dosadíme dvě libovolná čísla. Měli bychom dostat výsledek součtu obou čísel.

```
petr@PC:~$ export ROS_MASTER_URI=http://PC:11311
petr@PC:~$ rosservice call /add_two_ints 5 5
sum: 10
```

Obr. 20- Test funkčnosti služby na cílovém systému.

Nyní už k izolaci služby. V okně, kde běží ROSPenTo zvolíme možnost 14 (Isolate service...). Poté dostaneme dotaz, kterou službu chceme izolovat. Zadáme tedy ID služby ***/add_two_ints***, což je podle analýzy nahoře 0.2. Pokud vše proběhlo v pořádku, měla by se zobrazit zpráva, které říká, že se službu povedlo úspěšně odregistrovat.

```
14
Which service do you want to isolate?
Please enter number of service (e.g.: 0.0):
0.2
Service Isolation completed successfully.
/add_two_ints_server has been unregistered as provider of /add_two_ints
```

Obr. 21 - Postup při izolaci služby.

Nyní si můžeme ověřit, jestli se útok vydařil. Pokud ano, služba by na cílovém systému měla být nedostupná. Zkusíme ji tedy zavolat a pokud vše proběhlo v pořádku, zobrazí se chyba, která říká, že služba není dostupná. Výhodou pro útočníka je, že uzel, který nabízí službu, je i po útoku stále součástí cílového systému, což znamená RQT graf by měl být stejný před, i po útoku.

```
petr@PC:~$ rosservice call /add_two_ints 5 5
ERROR: Service [/add_two_ints] is not available.
```

Obr. 22 - Chybová hláška způsobená nedostupností služby

8.3 ARP spoofing

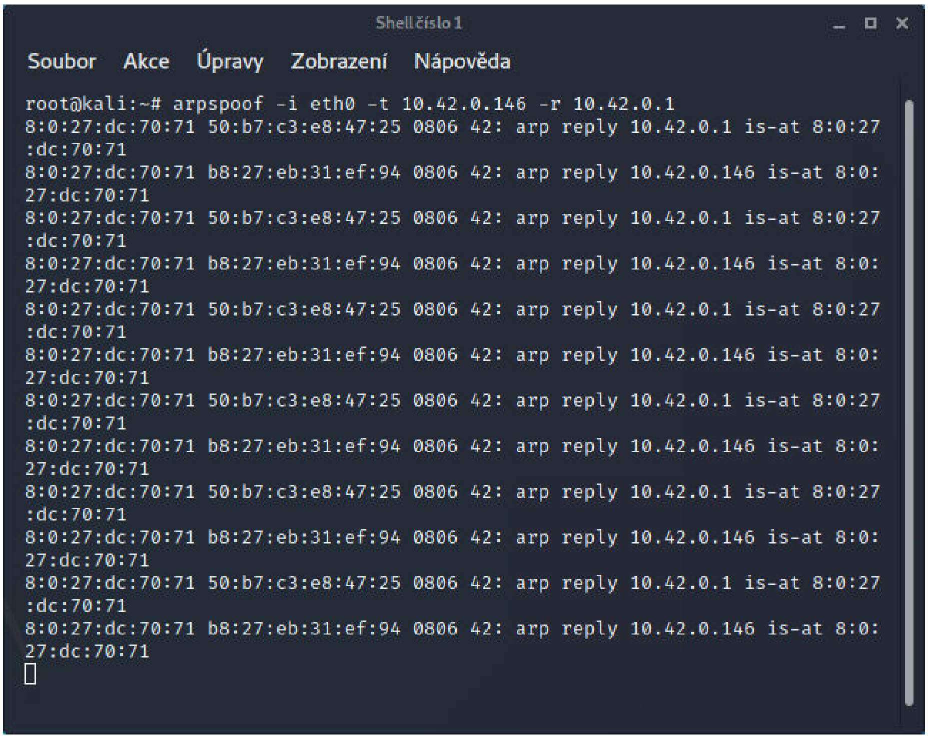
V této části si ukážeme, jak odposlechnout komunikaci mezi dvěma ROS uzly pomocí tzv. ARP spoofingu. Jedná se o útok typu Man-In-The-Middle, který spočívá ve zneužití protokolu ARP (Adress Resolution Protocol). ARP je síťový protokol, který slouží k "překladu" IP adres na MAC adresy. V praxi tento protokol slouží k informování routeru o tom, jaké IP adresy jsou přiřazeny jednotlivým zařízením v síti.

Podstatou ARP spoofingu je podvržením odpovědi na ARP dotaz, čímž router přesvědčíme o tom, že daná IP adresa odpovídá jiné fyzické adrese MAC. K tomuto útoku budeme potřebovat Kali Linux, což je speciální linuxová distribuce odvozená od Debianu, která slouží převážně pro penetrační testování a je i oblíbeným nástrojem hackerů. Dále budeme potřebovat ještě Wireshark, který je ale už součástí Kali Linuxu, takže jej nebudeme muset stahovat zvlášť.

Teď už k samotnému útoku. Opět budeme potřebovat dva nezávislé ROS systémy, běžící na dvou různých zařízeních ve stejné síti, které spolu komunikují. Do stejné sítě bude připojen i útočník s Kali Linuxem. Pro tento příklad budeme uvažovat, jedno ze zařízení bude např. RaspberryPi, kde ROS poběží na defaultním portu 11311. Na tomto zařízení spustíme uzel ***/talker*** z tutoriálů zmiňovaných už dříve. Druhým zařízením může být libovolný počítač, kde ROS poběží také na portu 11311. Na tomto počítači spustíme uzel ***/listener***. V tuto chvíli se v terminálů ještě nevypisují zprávy přijaté od uzlu ***/talker***, protože o sobě zařízení zatím neví.

Abychom navázali komunikaci mezi zařízeními, provedeme jejich propojení pomocí ROSPenTo tak, jak je popsáno v kapitole 8.2.4. Samozřejmě opět musíme znát ROS master URI obou zařízení. Pro tento konkrétní příklad předpokládejme, že IP adresa RaspberryPi, na kterém běží talker, je 10.42.0.1 a IP adresa počítače, na kterém běží listener je 10.42.0.146.

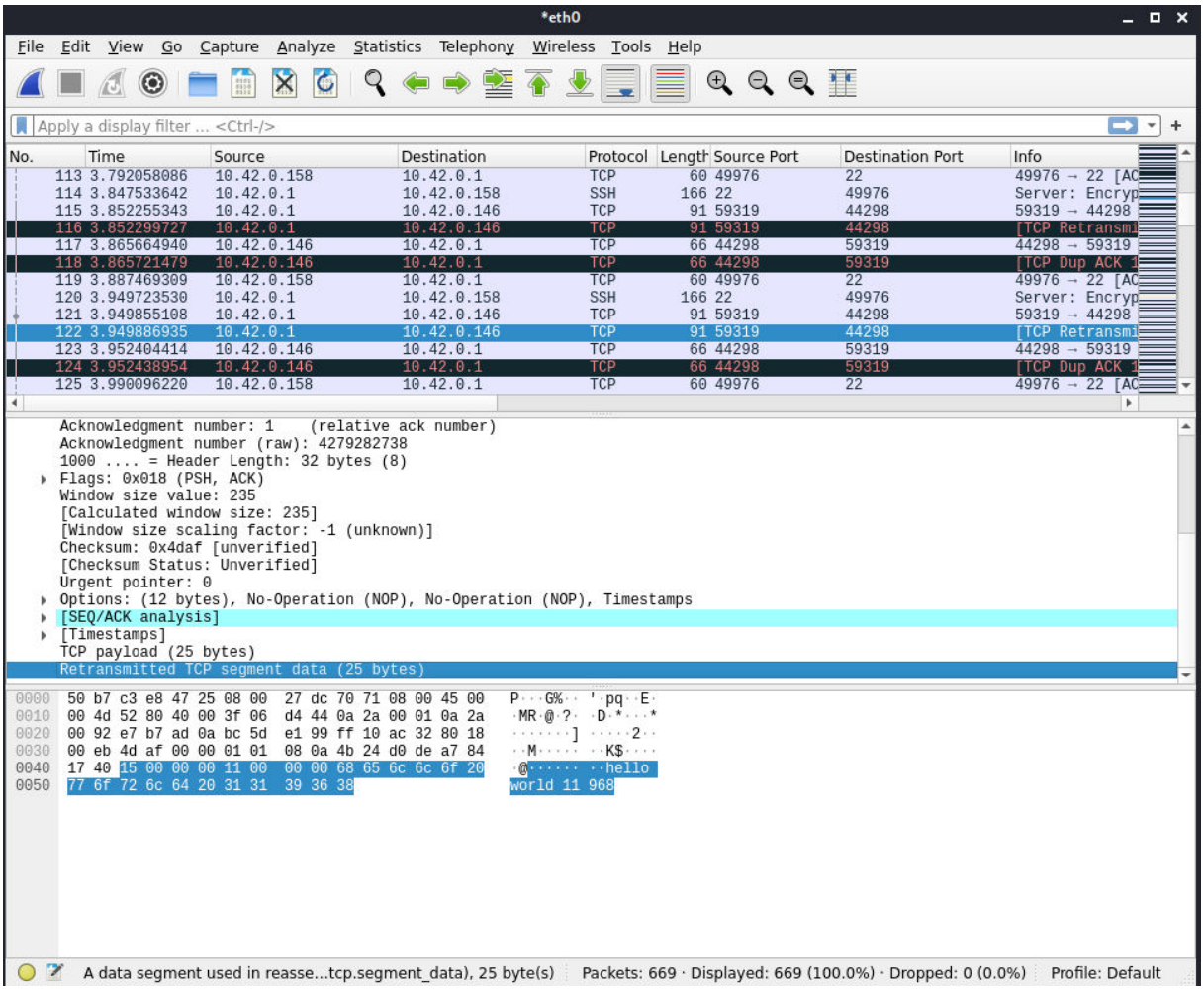
Nyní si spustíme Kali Linux, kam je třeba se přihlásit, jako root a otevřeme Wireshark, ale zatím nespouštíme odposlech. Zároveň si otevřeme terminál v Kali Linuxu, pokud jsme tak už neudělali a použijeme příkaz ***arpspoof***, který má ještě tři parametry a to -i (interface), -t (target) a -r (router) . Interface bude v našem případě -eth0, adresa cílového zařízení je 10.42.0.146 a adresa routeru 10.42.0.1. Výsledný příkaz tedy bude vypadat takto: ***arpspoof -i eth0 -t 10.42.0.146 -r 10.42.0.1***. Poté, co příkaz potvrdíme, bychom měli vidět, že systém začal podvrhovat odpovědi na ARP dotazy.



Obr. 23 - ARP spoofing pomocí Kali Linuxu.

Dalším krokem je spustit odposlech ve Wiresharku. Měli bychom vidět, že se nám občas objeví černě označené TCP pakety, které mají buď zdrojovou adresu 10.42.0.1 a cílovou 10.42.0.146 nebo naopak. Nás budou zajímat ty první. Pokud tyto pakety nevidíme, tak přerušíme jak ARP spoofing, tak i odposlech a zkusíme to znovu. Někdy se to hned na poprvé nepovede. Pokud tyto pakety ve Wiresharku uvidíme, je dobré ARP spoofing co nejrychleji přerušit, abychom se vyhnuli tomu, že nás někdo odhalí. Čím kratší dobu ARP spoofing provádíme, tím bezpečnější to pro nás je.

Pokud nyní ve Wiresharku klikneme na libovolný černě označený TCP paket se zdrojovou adresou 10.42.0.1 a cílovou adresou 10.42.0.146, měli bychom vidět obsah přenášené textové zprávy, který je v tomto konkrétním případě ***"hello world 11 968"***. Toto je tedy relativně jednoduchý a efektivní způsob, jak odposlouchávat komunikaci mezi ROS uzly. Jak bylo už řečeno dříve, jedna ze zranitelností ROS je komunikace v plain-textu a chybějící šifrování a právě toho jsme zde zneužili.



Obr. 24 - Odposlech komunikace mezi uzly pomocí Wiresharku

9. Závěrečné zhodnocení

V minulé kapitole jsme si prakticky ukázali, jak provést některé útoky na ROS aplikace a teď už nezbývá nic jiného, než závěrečné zhodnocení.

Na dvou demonstračních aplikacích talker a listener jsme si ukázali několik možných typů útoku. Některé byly úspěšné, některé méně. Některými bychom nejspíš nezpůsobili moc velkou škodu, zatímco některé by pro systém oběti mohly být fatální. Teď si tedy všechno v rychlosti zrekapitulujeme.

Obecně by se dalo říct, že nejnáchylnější je ROS k odposlechům. V minulé kapitole jsme si ukázali dva možné způsoby, jak odposlech provést. Prvním bylo použití nástroje na penetrační testování ROSPenTo k odklonění komunikace z cílového systému směrem k nám. Tento útok zneužil neautorizovaného volání zprávy ***publisherUpdade*** k manipulaci s cílovým systémem a k jeho propojení s naším odposlouchávajícím systémem. Jako druhou možnost, jak provést odposlech jsme si předvedli ARP spoofing pomocí operačního systému Kali Linux a programu Wireshark. Tento útok zneužil toho, že ROS neposkytuje možnost šifrování zpráv a komunikace tak probíhá v plain-textu. Oba útoky byly úspěšné, nicméně by pravděpodobně nezpůsobily fatální škodu. Nejspíš bychom takto získali nějaké informace o cílovém systému, které by se daly dále použít.

Dále jsme si úspěšně prakticky předvedli dva způsoby, jak provést DoS útok. V prvním příkladě jsme na dálku přerušili probíhající komunikaci mezi dvěma uzly na vzdáleném ROS systému. Ve druhém příkladě jsme úspěšně izolovali uzel od přístupu ke službě, která sčítá dvě čísla. Další možností, jak provést DoS útok, by bylo spustit na cílovém systému uzel, který se jmenuje stejně, jako nějaký z již existujících uzlů. Tím bychom původní uzel vypnuli. Nicméně tento útok není úplně praktický, protože bychom museli oběť buď nějakým způsobem donutit ke spuštění uzlu, nebo bychom museli získat přístup k zařízení oběti.

Posledním typem útoku, o který jsme se pokusili, bylo vložení falešných příkazů do cílového systému. Tento útok bohužel nebyl úspěšný na našich demonstračních aplikacích. Nicméně podle dostupných zdrojů by tento typ útoku měl být možný a záleží tak spíš na tom, jak je cílový systém naprogramován, popř. na tom, jakou verzi ROS nebo Ubuntu používáme. Tímto typem útoku bychom mohli způsobit největší škodu, pokud by se nám například podařilo vložit falešná senzorová data, nebo pokud bychom převzali kompletně kontrolu nad robotem.

Cílem této práce bylo mimo jiné také otestovat náročnost útoků na ROS, jejich praktičnost a zjistit, jaké možné překážky se mohou během útoků vyskytnout.

Samozřejmě jednou z největších překážek bránící úspěšnému útoku může být neznalost ROS master URI oběti. Zde často nezbývá nic jiného, než využít nějaké volně dostupné skenery sítě a portů a postupovat metodou pokus - omyl, hlavně pokud cílový ROS systém neběží na defaultním portu 11311. Z toho vyplývá, že jednou z možností, jak běžný uživatel může zabezpečit svojí ROS aplikaci bez nějakých dalších rozšíření, a alespoň trochu ztížit útočníkovi práci, je spustit ROS na jiném portu.

Nějakým typům útoků také může bránit nedostatečná znalost cílového systému. Například neznalost toho, které parametry uzel odebírá. V mnoha případech by útočník musel detailněji znát implementaci uzlů, nebo se pokusit zjistit informace jiným způsobem. To by mohlo zahrnovat vše od sociálního inženýrství, což je jednoduše řečeno manipulace s lidmi, až po pokusy o odposlech síťového provozu nebo reverzní inženýrství celé aplikace. I přes relativní jednoduchost některých útoků tedy nemusí být jejich provádění vždy úplně praktické, protože útočníkovi v lepším případě zabere velmi dlouho sbírání potřebných informací. V tom horším se k těmto informacím vůbec nedostane.

Nicméně bezpečnost robotických systémů by se v současné době určitě neměla podceňovat a je dobré se při vývoji aplikace využívající ROS zamyslet nad otázkou bezpečnosti. Jednou z možností, jak posílit bezpečnost by mohlo být využití některého z bezpečnostních rozšíření představených v kapitole 7. V neposlední řade by také uživatel měl zvážit, jestli nevyužít penetrační testování aplikace ve svůj prospěch, za účelem zjištění zranitelností. Pokud totiž víme, kde zranitelnosti jsou, je mnohem snadnější bránit se možným útokům.

Použité zdroje

1) *Robot Operating System* [online]. [cit. 2019-08-14]. Dostupné z: https://www.ros.org/

2) *Documentation - ROS Wiki* [online]. [cit. 2019-08-14]. Dostupné z: http://wiki.ros.org/

3) M. O'KANE, Jason. *A Gentle Introduction to ROS* [online]. 24.4.2018, 166 [cit. 2019-08-14]. Dostupné z: https://www.cse.sc.edu/~jokane/agitr/agitr-letter.pdf

4) NICKELS, Kevin a John KERR. *Robot Operating Systems:Bridging the Gap between Human and Robot* [online]. 13.11.2012, 7 [cit. 2019-08-14]. Dostupné z: https://www.researchgate.net/publication/254031552_Robot_operating_systems_Bridging_the_gap_between_human_and_robot

5) PRIYADARSHINI, Ishaani. *Cybersecurity risks in Robotics* [online]. 18.12.2017, 19 [cit.2019-10-15]. Dostupné z: https://www.researchgate.net/publication/319354229_Cyber_Security_Risks_in_Robotics

6) RIVERA, Sean, Sofiane LAGRAA a Radu STATE. *ROSploit: Cybersecurity tool for ROS* [online]. Lucembursko, 2019[cit.2019-10-15]. Dostupné z: https://orbilu.uni.lu/handle/10993/39278

7) MCCLEAN, Jarrod R. a Charles FARRAR. *A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS)* [online]. May 2013, 9 [cit. 2019-10-15]. Dostupné z: https://www.researchgate.net/publication/290997931_ROS_paper

8) ČERNEKOVÁ, ALŽBĚTA ČERNEKOVA. *ANALÝ ZA A DEMONSTRACE VYBRANÝCH SÍŤOVÝCH ÚTOKŮ POD OS LINUX* [online]. 2020, 46 [cit.2019-10-15]. Dostupné z: https://dspace.vutbr.cz/xmlui/handle/11012/52899

9) DIEBER, Bernhard, Severin KACIANKA, Stefan RASS a Peter SCHARTNER. *(Application-level) Security for ROS-based Applications* [online]. November 2016 [cit. 2019-11-25]. Dostupné z: https://www.researchgate.net/publication/309282648_Application-level_Security_for_ROS-based_Applications¨

10) J. RODRÍGUEZ-LERA, Francisco, Vicente MATELLÁN-OLIVERA, Jesús BALSA-COMERÓN, Ángel MANUEL GUERRERO-HIGUERAS a Camino FERNÁNDEZ-LLAMAS. *Message encryption in robot Operating system: collateral effects of hardening*

Mobile robots [online]. 02.03.2018, 12 [cit. 2020-02-21]. Dostupné z: https://www.frontiersin.org/articles/10.3389/fict.2018.00002/full

11) DIEBER, Bernhard, Ruffin WHITE, Sebastian TAURER, Benjamin BREILING, Gianluca CAIAZZA, Henrik CHRISTENSEN a Agostino CORTESI. *Penetration testing ROS* [online]. 2019, 41 [cit. 2020-02-21]. Dostupné z: https://bernharddieber.com/publication/dieber2019penetration/dieber2019penetration.pdf

12) WHITE, Ruffin a Henrik ISKOV CHRISTENSEN. *SROS: Securing ROS over the wire, in the graph, and through the kernel* [online]. November 2016, 3 [cit. 2020-03-14]. Dostupné z: https://www.researchgate.net/publication/310671472_SROS_Securing_ROS_over_the_wire_in_the_graph_and_through_the_kernel

13) PORTUGAL, David, Miguel A. SANTOS, Samuel PEREIRA a Micael S. Couceiro S. COUCEIRO. *On the Security of Robotic Applications Using ROS* [online]. November 2018, 17 [cit. 2020-03-14]. Dostupné z: http://ingeniarius.pt/davidbsp/publications/2018_PSPC18_Book_Chapter_Security_ROS.pdf

14) DIEBER, Bernhard, Benjamin BREILING, Sebastian TAURER, Severin KACIANKA, Stefan RASS a Peter S SCHARTNER. *Security for the Robot Operating System* [online]. 7.10.2017, 30 [cit. 2020-03-14]. Dostupné z: https://www.sciencedirect.com/science/article/abs/pii/S0921889017302762

15) DILUOFFO, Vincenzo, William R MICHALSON a Berk SUNAR. *Robot Operating System 2: The need for a holistic security approach to robotic architectures* [online]. 4.3.2018, 15 [cit. 2020-04-02]. Dostupné z: https://journals.sagepub.com/doi/pdf/10.1177/1729881418770011

16) CAIAZZA, Gianluca. *Security Enhancements of Robot Operating Systems* [online]. 23.3.2017, 111 [cit. 2020-04-11]. Dostupné z: http://dspace.unive.it/handle/10579/10238

17) BREILING, Benjamin, Bernhard DIEBER a Peter SCHARTNER. *Secure communication for the robot operating system* [online]. 11.1.2018, 7 [cit. 2020-06-08]. Dostupné z: https://ieeexplore.ieee.org/document/7934755

18)M. VILCHES, Víctor, Ibai A. APARICIO, Unai A. CARBAJO a Endika G. URIARTE. *Robot cybersecurity* [online]. 2019, 56 [cit. 2021-03-13]. Dostupné z: https://aliasrobotics.com/files/red_teaming_rosindustrial.pdf