Polytechnique Montréal

Département de génie informatique et génie logiciel

INF6103

Cybersécurité des Infrastructures Critiques

Co-simulation d'une infrastructure routière utilisant Modbus TCP avec Unity

REDACTED (REDACTED)
Diab Khanafer (REDACTED)
REDACTED (REDACTED)
REDACTED (REDACTED)

10 décembre 2023

Table des matières

1 Contexte **3**

2 Objectifs **3**

3 Installation et configuration **4**
- 3.1 Vue générale de l'architecture . . . 4
- 3.2 Simulation Unity . . . 4
  - 3.2.1 Unity . . . 4
  - 3.2.2 Simulation . . . 4
- 3.3 Logique du PLC . . . 7
- 3.4 ScadaBR . . . 7
- 3.5 Environnement Modbus TCP . . . 7

4 Choix et exploitation de la vulnérabilité **8**
- 4.1 Vulnérabilités . . . 8
  - 4.1.1 Communication sans-fil . . . 8
  - 4.1.2 Absence de chiffrement . . . 8
  - 4.1.3 Absence de mécanisme d'anti-rejeu . . . 9
- 4.2 Choix . . . 9

5 Expérimentation **9**
- 5.1 Scénarios expérimentaux . . . 9
  - 5.1.1 Scénario 1 - Le pont . . . 9
  - 5.1.2 Scénario 2 - L'intersection . . . 10
  - 5.1.3 Scénario 3 - Le compétiteur de l'épicerie . . . 10
- 5.2 Exploitation . . . 10
- 5.3 Résultats . . . 12

6 Propositions de rémédiation **13**
- 6.1 Encapsulation TLS . . . 13
  - 6.1.1 Mise en place de la remédiation . . . 14
- 6.2 Modbus Security . . . 14

7 Limites et suggestions d'amélioration **15**

8 Bilan et conclusion **15**
- 8.1 Travaux futurs . . . 16

A Captures des interfaces développées pour ScadaBR **18**

Abrégé (Abstract)

Avec la modernisation des modes de transport est arrivée l'automobile et avec elle le système routier utilisé par des milliards de personnes qui s'y fient pour se déplacer de la manière la plus sécuritaire et efficace possible d'un point A à un point B. D'une perspective de cyberterrorisme, cette infrastructure est une cible de choix, et il convient de se questionner sur les vulnérabilités qui pourraient y exister. Une perturbation de la confidentialité, de l'intégrité et/ou de la disponiblité, trois piliers essentiels en cybersécurité, pourrait avoir des conséquences catastrophiques. C'est la raison pour laquelle l'infrastructure routière et son contrôle constituent une infrastructure critique qui doit être étudiée. Puisque effectuer des tests sur des infrastructures réelles est complexe, il est essentiel d'investiguer la faisabilité de bancs d'essai produisant des résultats répétables et transférable dans la mesure du possible aux infrastructures réelles. Cet article explore la faisabilité d'une co-simulation utilisant Modbus TCP et le moteur de jeu Unity afin de tester des cyberattaques connues et observer leur impact d'un point de vue de sécurité informatique.

1. Contexte

La viabilité du système de trafic routier est essentielle pour la société. Cela signifie que sans la présence ou le bon fonctionnement de ce système, des répercussions significatives vont impacter les activités sociétales, l'économie et la sécurité des personnes et des biens. Notamment, des impacts importants sur la confidentialité, l'intégrité ou encore la disponibilité des services seraient potentiellement catastrophiques. La résistance de ce type d'infrastructure, qualifiée de critique, est donc importante à valider. Alors qu'effectuer des attaques de test sur des infrastructures réelles s'avère complexe et potentiellement dangereux, la science a recours aux bancs d'essai, cela pour s'assurer de détecter et de traiter les failles avant la mise en production de tels systèmes.

Cette étude est donc réalisée pour évaluer la faisabilité de déploiement d'un banc d'essai pour les réseaux routiers, dans l'objectif spécifique de mesurer les impacts de cyberattaques. Alors que l'intention initiale était de reproduire à l'identique une étude proposée par M. Urdaneta et al. [1], la complexité et la quantité impressionnante de données générées par la solution de simulation SUMO ne semblait pas pertinente dans le seul but de valider la faisabilité du banc d'essai. Après discussion des objectifs en section 2, ces enjeux seront traités davantage dans la section 3.

Le reste de ce document est organisé comme suit : la section 4 présente et explique différentes vulnérabilités présentes dans les infrastructures de contrôle des réseaux routiers, la section 5 mentionne les scénarios étudiés et l'applicabilité des attaques, la section 6 suggère des approches de remédiation vis-à-vis des vulnérabilités trouvées, la section 7 présente les limites de notre travail et finalement des conclusions composent la section 8.

2. Objectifs

Par ce travail, nous cherchons à démontrer les limites de la reproduction d'une infrastructure critique avec un banc d'essai. Par le biais de cyberattaques, nous exploitons des failles connues du protocole Modbus TCP, identifiées sur une infrastructure routière standard. Cela permet aussi d'observer les répercussions sur le réseau routier co-simulé. À partir de tout cela, le réalisme du banc d'essai est discuté par rapport à une infrastructure réel du réseau routier, puis des remédiations aux vulnérabilités constatées sont proposées.

Il est aussi montré comment Unity peut se substituer au logiciel SUMO présenté dans le papier de M. Urdaneta et al. [1], lorsque seuls les impacts routiers liés à la cybersécurité sont étudiés.

3. Installation et configuration

L'installation et la configuration ont été basés sur une idée générale fournie par l'article de Urdaneta et al. au niveau l'architecture, cela avec quelques changements notables afin d'adapter cette architecture à nos choix technologiques [1]. En effet, notre choix de logiciel pour simuler le trafic routier s'est arrêté sur le moteur de jeu Unity plutôt que sur la solution SUMO. Unity offre une approche moins complexe pour la simulation du réseau routier par rapport à SUMO. Unity offre des possibilités d'intégration avec d'autres technologies et plates-formes, ce qui est crucial puisque notre simulation de transport doit être connectée aux différentes composantes (interfaces de contrôle etc...). Dans notre cas, nous n'avons pas besoin de la complexité offerte par SUMO, qui convient plutôt aux scénarios nécessitant un niveau élevé de détail et de précision, comme les "noise emission, pollutant emision and fuel consumption outputs to quantify the economic, environmental and societal effects of road congestion" [1]. Ces facteurs ne sont pas nécessaire dans notre analyse.

3.1. Vue générale de l'architecture

La Figure 1 montre un diagramme général de l'architecture mise en place pour la co-simulation. Cette architecture est composée de différents éléments qui seront abordés dans les sous-sections suivantes. Il est essentiel de bien comprendre l'architecture de la co-simulation afin de saisir à quel endroits l'attaquant pourra interagir et de quelles façons la simulation dans Unity reflète l'état du PLC.

3.2. Simulation Unity

3.2.1. Unity

Unity est un puissant moteur de jeu qui inclut, entre autres, un moteur physique de simulation. Alors que cette plateforme est majoritairement utilisée dans l'industrie vidéoludique, de nombreux avantages la qualifient aussi comme candidate pour générer des simulations. Parmi ces avantages, outre son moteur physique, on retrouve sa prise en main relativement rapide pour qui est habitué des interfaces de modélisation 3D, la configuration minimale requise pour avoir une communication convaincante et la facilité de programmation d'une interface avec des outils externes. Cela ne veut pas dire qu'aucune difficulté n'a été rencontrée, loin de là, mais ces arguments qualifiaient Unity comme candidat à une simulation d'infrastructure routière.

3.2.2. Simulation

La mise en place de la simulation sur Unity a été un processus intéressant riche en découvertes. D'abord, il convient de comprendre le fonctionnement général de Unity. Unity

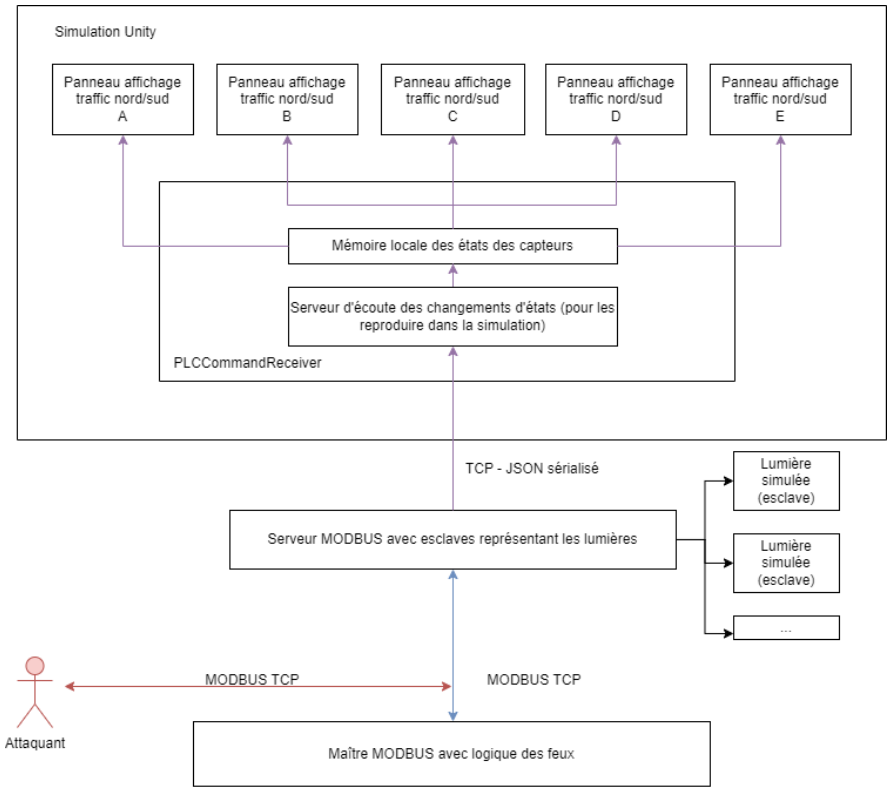

FIGURE 1 – Architecture générale

propose deux modes de création : le mode 2D et le mode 3D. Dans notre cas, il nous semblait pertinent de simuler la physique dans un monde qui se rapprochait comme possible du réel, et c'est la raison pour laquelle Unity 3D a été utilisé. L'interface de Unity permet de créer des objets dans le monde 3D à l'aide de modèles 3D préconçus. Il est aussi possible de concevoir nos propres modèles 3D.

Chaque objet dans le monde est appelé *GameObject* et peut se faire greffer un script de comportement *MonoBehaviour*. Ce script contient des fonctions du moteur de jeu que Unity appelle à des fréquences déterminées par un cycle de vie préétabli. Les fonctions les plus intéressantes pour notre simulation sont d'abord Start(), qui est invoquée une seule fois lors du démarrage, puis FixedUpdate() et Update(), dont on peut comprendre l'essentiel en disant qu'elles sont appelées à chaque génération d'une image *frame* dans la simulation. Ces deux dernières fonctions sont très utiles pour qu'un *GameObject* effectue des vérifications périodiques sur son environnement, comme si une voiture veut vérifier si une autre voiture est devant elle à courte distance pour un instant t [2].

Avec cette compréhension du cycle d'invocation des fonctions du moteur de jeu de Unity, il faut maintenant comprendre la manière dont les événements physiques se manifestent dans la simulation. Au-delà des scripts manuellement programmés qui sont greffés à des *GameObject*, le moteur physique de Unity 3D gère l'éclairage, les collisions et la gravité simulée dans l'environnement. Les *GameObject* présents dans l'environnement qui sont représentés par des modèles 3D ont aussi une boîte de collision 3D *hitbox*. Il en découle que deux boîtes de

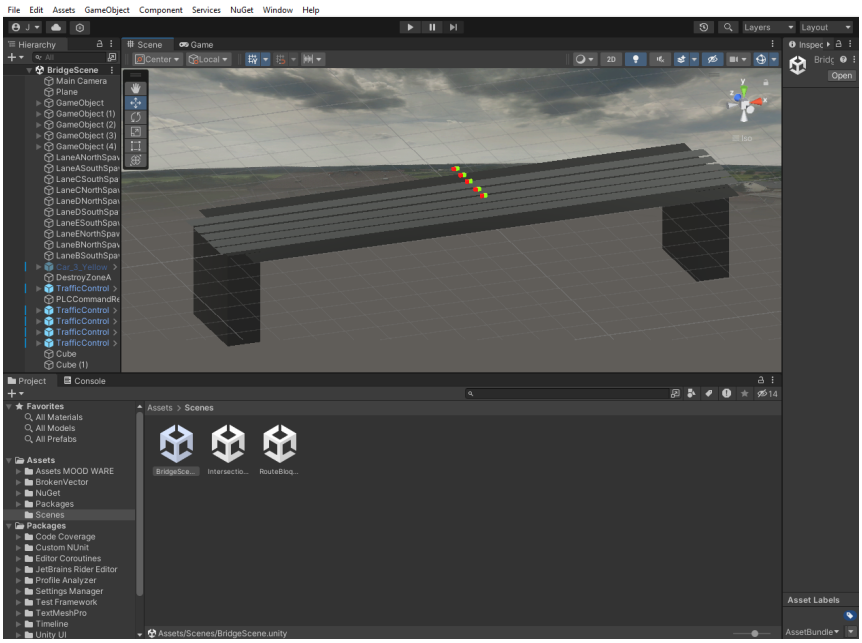

FIGURE 2 – Exemple de scène construite dans Unity (pont à 5 voies)

collision qui entrent en contact ne peuvent pénétrer une dans l'autre - on peut les considérer comme les parois du *GameObject*. Quant à la gestion de la gravité et de la physique, il est possible de désigner un *GameObject* comme étant un objet rigide, c'est-à-dire un objet qui sera affecté par le moteur physique 3D et ses contraintes, dont fait partie la gravité.

Afin de mettre en place une simulation d'apparence réelle en 3D, il a fallu construire plusieurs scènes. La Figure 2 montre un exemple de scène construite. Dans Unity, les scènes sont des instances d'environnement 3D qui représentent chacune une simulation distincte. Pour chaque scénario présenté dans la section 5, il a fallu développer une scène distincte avec un objet rigide représentant la route (un simple plan horizontal XZ dans l'espace dans un référentiel où Y représente l'axe vertical), la bonne colorisation pour distinguer la route du reste du plan, des objets invisibles en charge de l'apparition périodique des voitures, puis finalement des indicateurs lumineux faisant office de feux de circulation.

Des scripts spécifiques ont alors été écrits avec le langage C#. Les plus importants, en lien avec le comportement des voitures, sont pour gérer l'apparition des voitures, gérer l'accélération et la décélération des voitures individuellement, détecter les voitures arrêter devant soi, puis contrôler le comportement d'une voiture qui approche un feu jaune ou tout autre type de feu. D'autres scripts plus globaux incluent le serveur TCP qui reçoit les instructions de la simulation Modbus TCP pour en refléter l'état dans la simulation Unity, ou encore la destruction des voitures une fois leur trajet complété.

Pour sa part, le serveur TCP fonctionne avec des connexions par socket et il délègue les commandes reçues de la simulation MODBUS à des fils d'exécution qui se chargent de faire les changements dans la simulation visuelle.

3.3. Logique du PLC

Le Modbus esclave que nous avons programmé est un serveur Modbus à l'écoute des requêtes du Modbus maître. Le Modbus maître comporte une logique préprogrammé d'états des lumières pour chaque scénario de la section 5, et il transmet ses requêtes d'écriture au serveur Modbus esclave. Ce serveur garde ensuite les états en mémoire, puis convertit ces données en JSON avec les états du PLC, dans un format non-modbus qui sera communiqué au serveur TCP de la simulation Unity. La structure utilisée est la suivante :

```
    plc_json = {
        "id": address ,
        "states": [
            ## Northbound - East
            {
                "green": values[0],
                "yellow": values[1],
                "red": values[2]
            },
            ## Southbound - West
            {
                "green": values[3],
                "yellow": values[4],
                "red": values[5]
            }
        ]
    }
```

id : représente l'adresse et le nom des coils individuels
states : représente le statut du coil
yellow, green, red : représente les données actuelles du coil

Ce JSON est envoyé via la communication TCP à la simulation Unity. Les adresses utilisées sont prédéfinies selon la simulation en cours, allant de 0 à 6*n, n étant le nombre de lumières que nous avons dans la simulation.

3.4. ScadaBR

Pour la surveillance de l'infrastructure, ScadaBR a été utilisé. Il permet de connecter une source d'entrée de PLC qui permet de surveiller l'état de ses coils et de ses registres. L'outil permet aussi de directement effectuer des changements aux registres des PLC tel qu'un client maître. On peut voir le résultat de la vue sur Scada BR lorsque les simulations sont actives en regardant les figures 7, 8 et 9.

3.5. Environnement Modbus TCP

Le contrôleur maître (client Modbus) communique avec le Modbus esclave (ou serveur Modbus) en utilisant le protocole Modbus TCP. Le client Modbus initie des requêtes vers le Modbus esclave ou serveur, avec un numéro de simulation (parmi trois scénarios) et le mode de communication (TCP ou TLS). En utilisant la classe ModbusServer de PyModbus,

une instance du serveur esclave est créée en spécifiant l'adresse IP et le port sur lequel il écoutera les demandes. Traditionnellement, il s'agit du port 502, toutefois pour des raisons de permissions sur le système d'exploitation Windows, notre simulation s'est effectuée sur le port 5020. Des blocs de registres sont ajoutés pour stocker les données à lire ou écrire.

```
1     client.write_coils(0, [True, False, False, True, False False], slave=1)
2     client.write_coils(6, [False, False, True, False, False True], slave=1)
```

Comme le montre le code ci-dessus, le maître contrôleur, fonctionnant comme un client Modbus, émet des requêtes vers le serveur esclave à l'aide de la classe ModbusClient. L'adresse IP et le port du serveur esclave sont spécifiés lors de la création de l'instance du client. Les méthodes fournies par PyModbus, telles que "read_coils" et "write_coils" permettent au contrôleur de lire les données du serveur ou d'écrire de nouvelles valeurs [3].

4. Choix et exploitation de la vulnérabilité

4.1. Vulnérabilités

Plusieurs vulnérabilités ont été identifiées par rapport à l'infrastructure routière standard. Ces vulnérabilités se concentrent sur le protocole de communication utilisé entre les contrôleurs esclaves PLCs et les contrôleurs maîtres, soit Modbus TCP.

4.1.1. Communication sans-fil

Les communications sans-fil [1] augmentent la surface d'attaque pour l'interception des données par des parties malveillantes. Les acteurs malveillants ont potentiellement davantage d'opportunités de capturer et analyser le trafic sans-fil pour obtenir les données de commandes et/ou les états des capteurs, ce qui ne serait pas le cas avec un réseau physique qui nécessiterait un accès direct filaire au réseau, plus complexe à obtenir si des protections physiques adéquates sont en place.

4.1.2. Absence de chiffrement

Il n'y a pas de chiffrement utilisé dans le protocole Modbus TCP [1]. Cela peut être un point de vulnérabilité, car sans le chiffrement, les données échangées via Modbus TCP peuvent être interceptées et modifiées par des parties non autorisées ou malveillantes. En clair, il est possible de constater exactement quel code fonction est utilisé dans la communication, et d'en déduire le fonctionnement de manière plus approfondie.

4.1.3. Absence de mécanisme d'anti-rejeu

Dans l'absence d'un mécanisme d'anti-rejeu, un attaquant pourrait capturer des paquets de données Modbus TCP et les réinjecter dans le réseau à un moment ultérieur pour reproduire une action spécifique [1]. Ceci peut engendrer des perturbations sur le fonctionnement normal du contrôleur visé, avec des impacts variés dépendant du paquet réinjecté.

4.2. Choix

Les vulnérabilités de communication sans-fil et d'absence de chiffrement seront exploitées dans le cadre du banc d'essai. Il est important de comprendre que l'exploitation de ces vulnérabilités se fait en mode brèche assumée, c'est-à-dire qu'il est supposé que l'attaquant a déjà un pied à terre dans le réseau et qu'il l'a compromis. Ainsi, l'idée globale derrière l'exploitation de l'absence de chiffrement est la lecture et l'analyse du contenu des paquets. Puis, on suppose qu'on tire partie de la communication sans-fil pour que l'attaquant obtienne ce pied à terre supposé dans le réseau.

5. Expérimentation

Après la mise en place de notre banc d'essai, trois scénarios de circulation routière sont établis afin d'exécuter l'exploitation des vulnérabilités identifiées dans des contextes différents. Cela nous permet de concrètement démontrer les failles d'un réseau de contrôle routier, et aussi de distinguer les divers impacts d'une cyberattaque sur la circulation routière de manière visuelle.

5.1. Scénarios expérimentaux

Tous nos scénarios disposent de PLCs locaux aux outils d'information routière (i.e. le feu de circulation, le panneau d'affichage électronique), ainsi qu'un contrôleur local à la section routière (i.e. le pont, l'intersection).

Une logique prédéfinie est incorporée dans ce contrôleur local, afin qu'il puisse transmettre à chaque PLC son statut attendu.

5.1.1. Scénario 1 - Le pont

Ce scénario est composé d'un pont à 5 voies au-dessus desquelles se trouve un panneau d'affichage électronique. Ces panneaux permettent d'indiquer le sens de circulation autorisé - vers le nord ou vers le sud. De manière standard, les panneaux des deux voies d'une extrémité et ceux des deux voies de l'extrémité opposée, indiquent respectivement une circulation vers le nord et vers le sud. La voie du milieu va changer de sens en fonction du moment de la

journée afin d'accommoder la circulation et de diminuer la saturation du réseau, comme cela pourrait se voir dans les périphéries d'une grande ville à la congestion significative.

5.1.2. Scénario 2 - L'intersection

Ce scénario est la réplique d'une intersection avec deux voies à sens double qui se coupent en angle droit. L'intersection comporte quatre feux de signalisation tricolore indiquant le droit de passage sur chacune des voies. Il est à noter que notre simulation ne supporte pas les virages aux intersections - cela est discuté en détail dans la section 7. Ainsi, les véhicules se déplacent toujours en ligne droite.

5.1.3. Scénario 3 - Le compétiteur de l'épicerie

Ce scénario capture une situation plus spécifique que les deux premiers. Il comporte une voie à sens unique vers le nord depuis laquelle il est possible de tourner vers la droite afin de rejoindre une seconde voie aussi à sens unique. Nous sommes dans un périmètre citadin réduit où se trouvent deux épiceries en compétition - l'épicerie A et l'épicerie B. Chacune est située à l'extrémité de l'une des voies décrite précédemment.

Au niveau de l'intersection se trouve un panneau d'affichage électronique qui indique à l'automobiliste la possibilité de continuer sur la voie vers le nord ou l'interdiction de le faire. Dans le dernier cas, cela oblige l'automobiliste à prendre un détour vers la voie de droite.

Le but de chaque conducteur est d'aller faire son épicerie. À noter que l'épicerie A est plus proche que l'épicerie B, donc on comprend que dans ce scénario, l'épicier B agissant comme attaquant a pour motif d'augmenter ses profits en contraignant les automobilistes à faire leur épicerie chez lui, car ces derniers ne peuvent plus accéder à l'épicier A à cause de la route maintenant bloquée.

5.2. Exploitation

L'exploitation des vulnérabilités est faite en utilisant les outils WireShark [4] et Modbus Poll [5]. WireShark est un outil gratuit qui permet, entre autres, d'écouter les communications sur le réseau. Modbus Poll, dans sa version d'évalution limitée, permet de simuler un maître Modbus sur le réseau afin de lire les états des esclaves Modbus présents et/ou d'envoyer des requêtes d'écriture afin de commander leurs états. Une alternative gratuite serait l'utilitaire en ligne de commande mbpoll [6].

Dans un premier temps, pour un scénario choisi, nous lançons les scripts Python (maître Modbus et client Modbus) ainsi que la simulation Unity contenant un serveur TCP qui recevra les états des PLCs modélisés depuis le serveur Modbus. À partir de ce moment-là, le banc d'essai est en marche, l'attaque peut commencer.

Pour rappel, nous assumons que l'attaquant a déjà accès au réseau à distance, ce qui est

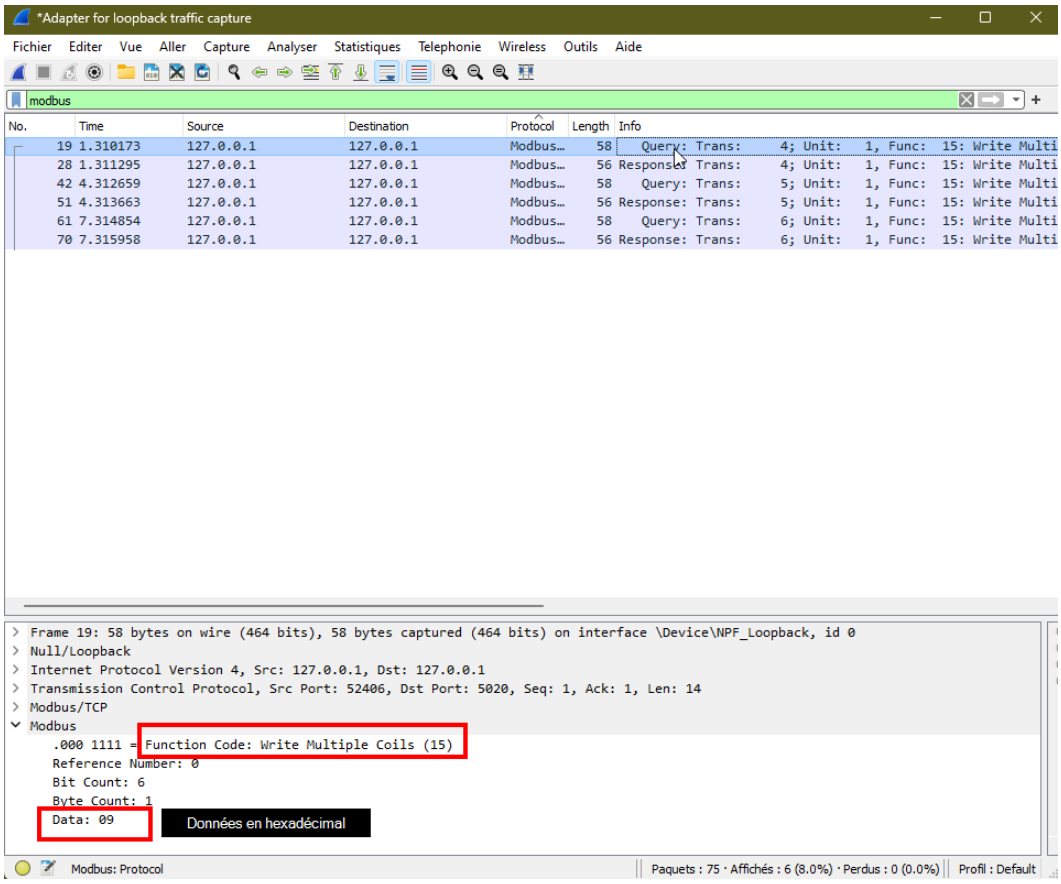

FIGURE 3 – Visualisation du trafic en clair avec Wireshark

facilité par le fait que la communication Modbus TCP est sans-fil, tel que précédemment indiqué à la section 4.

Après avoir obtenu l'accès au réseau, on utilise WireShark afin de voir les communications Modbus TCP et de les analyser - ce qui est possible car le protocole n'est pas chiffré, comme le montre la Figure 3. En faisant la comparaison avec l'état du système routier, l'attaquant est ainsi capable de déterminer le format du paquet nécessaire pour envoyer une commande personnalisée. On utilise alors Modbus Poll pour s'interfacer sur le réseau et envoyer le paquet Modbus TCP malveillant, en répliquant la structure désirée. En l'absence d'un système d'authentification, le PLC contrôlé par le client Modbus va interpréter la commande sans se poser de question, conduisant au succès de l'attaque.

Les attaques testées sont les suivantes :

— Le pont : l'indication du sens nord et du sens sud autorisés simultanément sur la voie centrale ;
— L'intersection : des feux rouges dans toutes les directions ;
— Le compétiteur de l'épicerie : l'épicier B mandate un pirate informatique afin que le panneau annonce l'interdiction de continuer sur la voie vers le nord (vers l'épicerie A).

La Figure 4 montre l'exécution réussie d'une attaque lors du premier scénario.

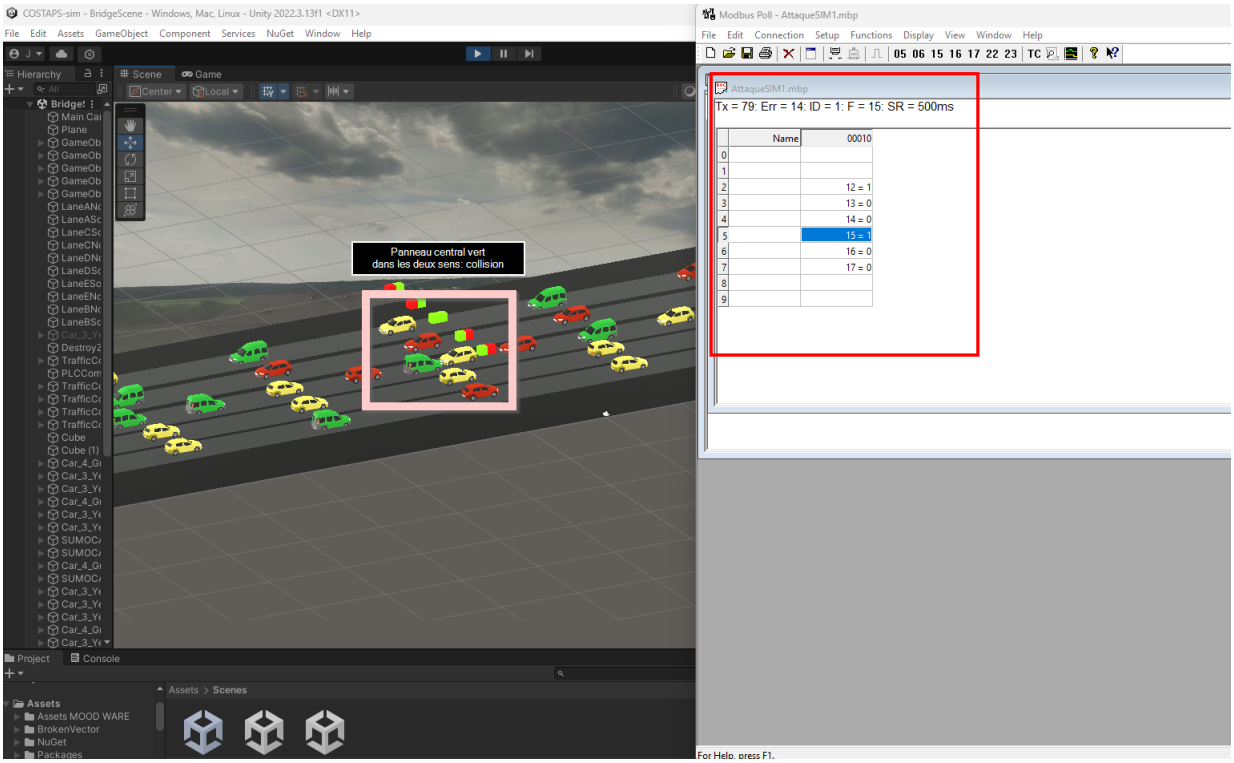

FIGURE 4 – Attaque réussie sur le pont du scénario 1

5.3. Résultats

La simulation des attaques sur ces scénarios distincts nous permet de constater différents impacts sur la circulation routière.

D'abord, il y a une dégradation générale du système de gestion du trafic à cause des interférences crées par les commandes malveillantes qui diminue la qualité du service si l'opérateur n'est pas vigilant, et qui augmente la latence à cause du trafic généré sur le réseau sans-fil. Ensuite, des délais, des collisions et des accidents sont aussi résultants possibles de ces cyberattaques, comme visible à nouveau dans la Figure 4.

Spécifiquement, pour le scénario du compétiteur de l'épicerie, les impacts sur le réseau routier s'accompagnent d'impacts financiers. En effet, l'épicerie B attaquante voit ainsi ses revenus augmenter, et leurs clients vont dépenser davantage en essence pour s'y rendre, car le voyage est plus grand (l'épicerie est plus loin). Nous avons implémenté dans Unity une manière de calculer le temps de parcours des voitures pour comparer le temps mis à aller à l'épicerie A (9 secondes) versus l'épicerie B (16 secondes). Bien que ces temps ne soient pas réalistes, le delta observé est suffisant pour conclure aux effets de délai sur une chronologie réelle.

6. Propositions de rémédiation

6.1. Encapsulation TLS

L'encapsulation TLS, Transport Layer Security, a comme objectif principal de chiffrer les données échangées entre les dispositifs. Cette mesure de sécurité essentielle garantit l'intégrité et la confidentialité des données transitant via le réseau. En utilisant TLS, les informations sensibles deviennent inintelligibles pour toute entité non autorisée, renforçant ainsi la protection des données contre les interceptions malveillantes.

Dans la remédiation, nous avons utilisé l'encapsulation TLS disponible dans la librairie PyModbus [3]. PyModbus, étant une bibliothèque Python populaire pour la mise en oeuvre du protocole Modbus, intègre le support TLS pour renforcer la confidentialité et l'intégrité des données échangées.

L'un des principaux objectifs de l'encapsulation TLS dans PyModbus [3] est de chiffrer les données transitant entre le client et le serveur Modbus. En activant cette fonctionnalité, toutes les informations sensibles, telles que les valeurs des registres Modbus, deviennent illisibles pour toute entité non autorisée. Cela garantit une communication sécurisée, en particulier dans des environnements où la confidentialité des données est essentielle.

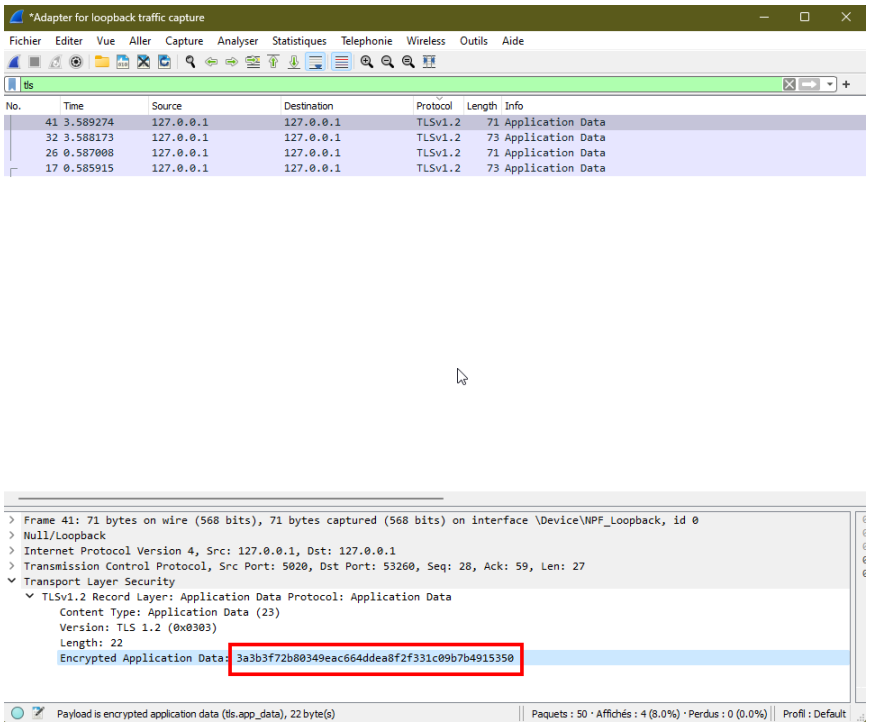

FIGURE 5 – Visualisation du trafic chiffré avec Wireshark

PyModbus permet également la mise en place de certificats TLS pour l'authentification des parties prenantes, ce que nous avons mis en place. Les certificats assurent l'identification

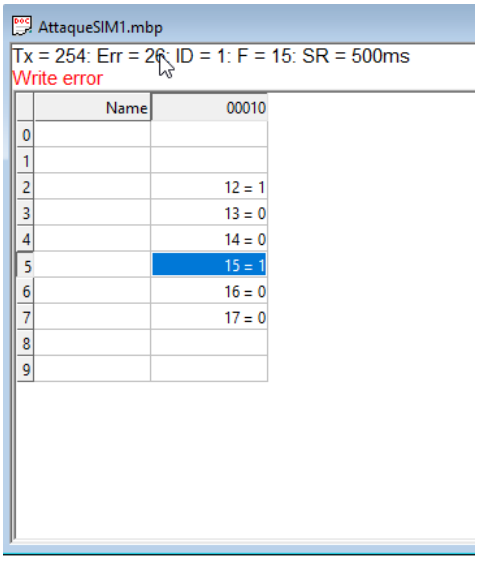

FIGURE 6 – Erreur d'exécution de Modbus Poll

mutuelle entre le client et le serveur, renforçant ainsi la confiance dans l'échange de données. En combinant le chiffrement des données et l'authentification, l'encapsulation TLS dans PyModbus contribue à prévenir les attaques telles que l'interception de données sensibles et les intrusions non autorisées.

6.1.1. Mise en place de la remédiation

Nous avons relancé nos simulations cette fois avec le mode de communication TLS, que nous avons configuré comme argument dans nos scripts de simulation. Nous avons effectué les mêmes cyberattaques. Cependant, cette fois-ci, l'attaquant n'a pas réussi à injecter ses commandes dans le système Modbus, comme nous pouvons le constater dans Wireshark sur la Figure 5 montrant le trafic TLS et sur la Figure 6 indiquant les erreurs dans l'interface de Modbus Poll.

6.2. Modbus Security

Pour renforcer la sécurité dans le contexte du protocole Modbus, des mécanismes spécifiques sont mis en place. Cela inclut l'implémentation d'un ensemble de mesures telles que l'authentification des dispositifs, le contrôle d'accès et d'autres stratégies pour prévenir l'accès non autorisé aux périphériques et aux données Modbus. Ces protocoles de sécurité contribuent à garantir l'intégrité des communications et à protéger les systèmes contre les potentielles intrusions [7].

Nous n'avons pas implémenté le protocole MBAPS (ModBus Application Protocol Security) pour plusieurs raisons. La principale justification réside dans la complexité de MBAPS, qui exige la réimplémentation d'un nouveau protocole.

7. Limites et suggestions d'amélioration

Le modèle de circulation actuel repose sur des scripts de comportement déterministes pour gérer le mouvement des véhicules. Bien qu'il existe d'autres outils dans Unity pour simuler le comportement des voitures, les options les plus prometteuses sont généralement payantes, ou alors nécessitent un temps de développement considérable pour nos compétences.

Actuellement, ni Unity ni SUMO ne sont spécialisés dans la modélisation de cyberattaque, et les facteurs humains ou physiques ne sont pas pris en considération dans la simulation existante. Cela rend par exemple difficile les tests par déni de service - car ils sont basés sur la performance des contrôleurs réels, ce dont nous ne disposions pas. La pertinence d'une attaque DoS dans une co-simulation de la circulation routière est limitée en raison des différences entre les PLC et les scripts, chacun ayant ses propres limitations et caractéristiques. La nature spécifique de la simulation doit être prise en compte pour évaluer l'impact potentiel d'une telle attaque.

Il est important de noter que la simulation construite n'est pas bi-directionnelle. Pour des considérations futures, il est envisagé d'implémenter la communication de Unity vers Modbus TCP, ce qui permettra de modéliser des situations où des capteurs physiques réagissent à la présence de voitures.

Une autre considération future consiste à prendre en compte le support de TLS avec ScadaBR, ce qui permettrait de surveiller même avec Modbus TCP encapsulé dans TLS. Ces ajustements potentiels visent à améliorer la représentation du système en prenant en compte des aspects tels que la sécurité, la bi-directionnalité et les réactions aux cyberattaques.

8. Bilan et conclusion

Le banc d'essai implémenté dans ce travail réussit à simuler l'infrastructure d'un réseau de contrôle routier minimal. Il intègre la modélisation du trafic avec Unity et la simulation du contrôle de la signalisation utilisant le protocole Modbus TCP par l'intermédiaire de scripts Python.

D'après nos expérimentations, ce banc d'essai est adapté pour l'évaluation des impacts de cyberattaques sur la circulation routière. Il ne permet pas encore de quantifier les impacts par exemple sur la durée du trajet (sauf dans certains cas spécifiques), sur le nombre d'accidents ou encore sur le nombre de victimes impliquées dans ces accidents. Toutefois, on peut clairement voir avec la simulation du trafic que le résultat des cyberattaques sur les différentes sections routières étudiées se propagent rapidement aux intersections alentours, c'est donc un constat qui peut être fait visuellement à l'aide du banc d'essai.

Les résultats soulignent qu'il est important de comprendre ces impacts qui deviennent facilement et rapidement significatifs sur le réseau routier. Cela touche donc directement aux activités sociétales, à l'économie et surtout à la sécurité des personnes, comme on le prévoyait au début de ce travail.

8.1. Travaux futurs

Des travaux futurs pourraient envisager de concevoir le banc d'essai en tenant en compte le concept de réseaux véhiculaires ou de véhicules intelligents. En effet, une mesure de prévention ou de mitigation contre les cyberattaques présentées, au niveau des usagers, pourrait être un système d'alerte entre les différents véhicules - cela est aisément simulables avec les scripts appropriés dans Unity, et les données résultantes pourraient éventuellement être exportées.

De plus, il serait pertinent de mettre en place une communication bi-directionnelle comme mentionné en section 7, ce qui ajouterait du réalisme au banc d'essai. Investiguer les manières de simuler Modbus Security en tenant compte des contraintes réelles du marché, ou encore l'utilisation d'un protocole de communication complètement différent pourrait aussi être envisagé.

Finalement, faciliter la paramétrisation du banc d'essai et en revoir l'efficacité du code ne pourrait qu'améliorer les perspectives de notre simulation pour utilisation par d'autres chercheurs et scientifiques curieux d'en comprendre le fonctionnement et de développer davantage de fonctionnalités en lien avec celle-ci.

Références

[1] Marielba Urdaneta, Antoine Lemay, Nicolas Saunier, Jose Fernandez (2018). A Cyber-Physical Testbed for Measuring the Impacts of Cyber Attacks on Urban Road Networks. 12th International Conference on Critical Infrastructure Protection (ICCIP), Mar 2018, Arlington, VA, United States. pp.177-196, https ://dx.doi.org/10.1007/978-3-030-04537-1_10

[2] Technologies, Unity. “Unity User Manual 2022.3 (LTS).” Unity, docs.unity3d.com/Manual/index.html.

[3] PyModbus. “Welcome to pyModbus’s documentation - pyModbus 0.2.1 documentation” pyModbus, pymodbus.readthedocs.io/en/latest/.

[4] Wirehark. "Wireshark User’s Guide" wireshark, https ://www.wireshark.org/docs/wsug/html/chunked,

[5] Mbpoll. “Modbus poll” Modbus poll, modbustools.com/download.html.

[6] Mbpoll Command Line. “Modbus poll command line” Modbus poll command line, github.com/epsilonrt/mbpoll.

[7] C. Palmer and S. Shenoi (Eds.) : Critical Infrastructure Protection III, IFIP AICT 311, pp. 83–96, 2009.

A. Captures des interfaces développées pour ScadaBR

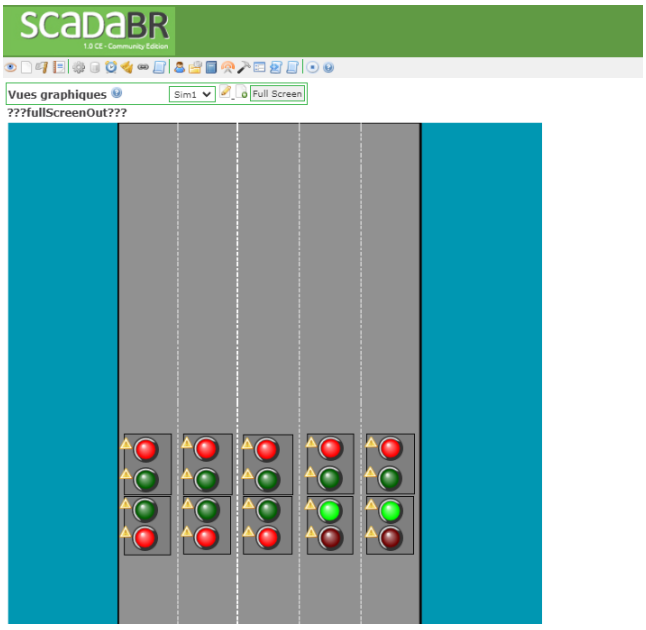

FIGURE 7 – Visualisation du premier scénario avec ScadaBR

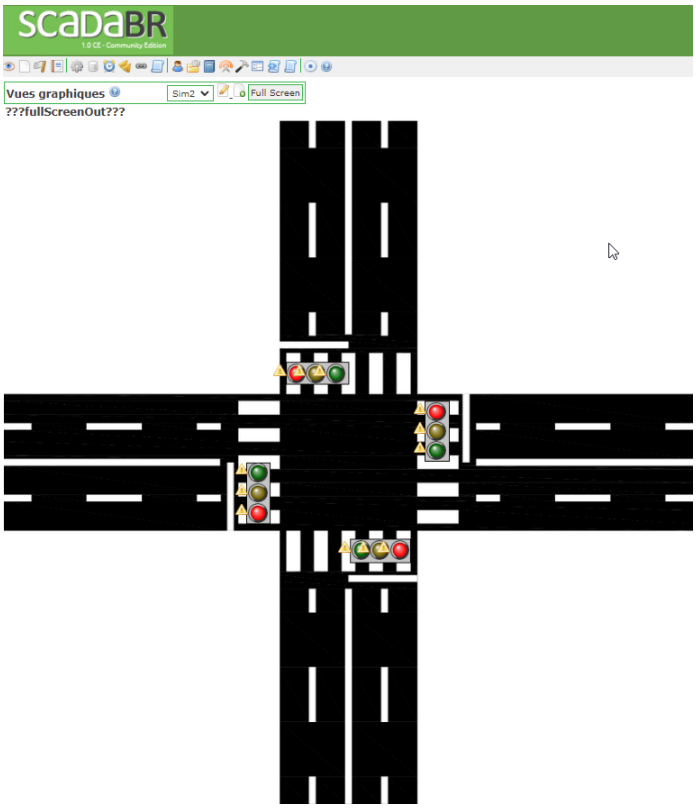

FIGURE 8 – Visualisation du second scénario avec ScadaBR

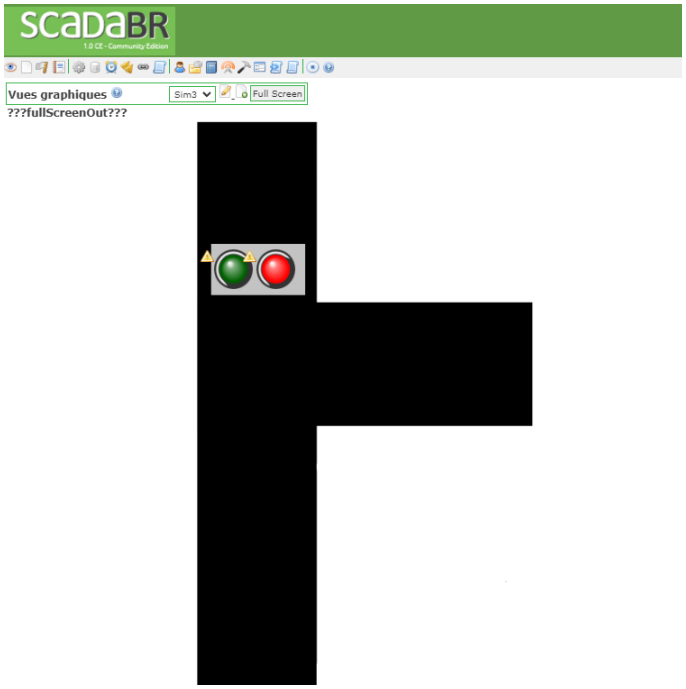

FIGURE 9 – Visualisation du troisième scénario avec ScadaBR