Πανεπιστήμιο Πειραιώς – Τμήμα Ψηφιακών Συστημάτων
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Ασφάλεια Ψηφιακών Συστημάτων»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	ΥΛΟΠΟΙΗΣΗ ΕΝΟΣ ΠΡΑΚΤΟΡΑ ΓΙΑ ΕΛΕΓΧΟ ΛΕΙΤΟΥΡΓΙΚΟΥ ΣΥΣΤΗΜΑΤΟΣ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΚΑΝΟΝΕΣ YARA ΚΑΙ ΠΛΗΡΟΦΟΡΙΕΣ ΠΟΥ ΑΝΤΛΟΥΝΤΑΙ ΑΠΟ ΤΟ MALWARE INFORMATION SHARING PLATFORM
Ονοματεπώνυμο Φοιτητή	Αναγνώστου Ιωάννης
Πατρώνυμο	Αθανάσιος
Αριθμός Μητρώου	MTE/1704
Επιβλέπων	Κωσταντίνος Λαμπρινουδάκης

Ημερομηνία Παράδοσης 9 Απριλίου 2019

Τριμελής Εξεταστική Επιτροπή

Υπογραφή	Υπογραφή	Υπογραφή

Λαμπρινουδάκης Κων/νος Ξενάκης Χρήστος Νταντογίαν Χριστόφορος

Περίληψη

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη ενός agent ο οποίος θα συνδέεται με τη διαδικτυακή πλατφόρμα Malware Information Sharing Platform and Threat Sharing (MISP) προκειμένου να αντλεί ενδείκτες παραβίασης (Indicators of Compromise). Μέσω αυτών θα προβαίνει σε μια στοχευμένη αναζήτηση κακόβουλων λογισμικών που δύναται να τρέχει το λειτουργικό σύστημα.

Συγκεκριμένα στην πλατφόρμα MISP αναλυτές κακόβουλων λογισμικών και κακόβουλης δραστηριότητας εισάγουν και διαμοιράζονται πληροφορίες που αντλούνται κατόπιν αυτής της ανάλυσης. Οι αναλυτές που χειρίζονται την πλατφόρμα, μπορούν να εισάγουν ενδείκτες παραβίασης αυτών των λογισμικών. Παράλληλα, οι χρήστες της πλατφόρμας μπορούν να συνδέονται και να ενημερώνονται για νέα ευρήματα αλλά και να αντλούν τους ενδείκτες παραβίασης προκειμένου να ενημερώνουν τα συστήματα ασφαλείας που χρησιμοποιούν.

Η εφαρμογή που υλοποιήθηκε αυτοματοποιεί αυτήν την διαδικασία αντλώντας ενδείκτες παραβίασης τύπου Yara και IP προορισμού. Προβαίνει σε έλεγχο των τελευταίων εκτελέσιμων προγραμμάτων που έχουν τρέξει στο λειτουργικό σύστημα, κάνει έλεγχο των διεργασιών που εκτελούνται την κάθε δεδομένη στιγμή καθώς και επιβλέπει τις συνδέσεις που το λειτουργικό σύστημα προσπαθεί να υλοποιήσει. Σε περίπτωση που κάποιος ενδείκτης ενεργοποιηθεί είτε διαγράφεται το αρχείο είτε σταματάει η διεργασία είτε τερματίζεται η σύνδεση.

Abstract

Main purpose of the current thesis is the development of an agent which will be connected with the Online platform Malware Information Sharing Platform and Threat Sharing (MISP) in order to retrieve Indicators of Compromise. Through them, it will perform a targeted research of malicious software that could be running in the operating system.

More specifically, in the MISP platform, analysts of malicious software activity, have the opportunity to both import and share information that they retrieve from the analysis. The analysts that handle the platform can import indicators related to software violation. On the other hand, users are able to connect and in parallel get informed on the new findings but also retrieve the indicators of violation so as to inform the security systems that they are using.

The application which has been developed, automates the process by retrieving violence indicators Yara and IP destination. It is performing a test of the latest executed programs that have been run in the operating system, checks the processes that are executed the particular moment and overviews the connections that the operating system is trying to perform. In case an indicator is activated or deleted, the files either ends the procedure or the connection is terminated.

Πίνακας Περιεχομένων

1. Εισαγωγή

Οι αναλυτές κακόβουλης δραστηριότητας , για να μπορέσουν να διαμοιράσουν πληροφορίες για τις δραστηριότητές τους και τις νέες τάσεις επιθέσεων σε τυχόν ενδιαφερόμενους, υλοποιήθηκε το Malware Information Sharing Platform and Threat Sharing (MISP). Σκοπός της συγκεκριμένης πλατφόρμας, είναι οι χρήστες της να εισάγουν ευρήματα κακόβουλης δραστηριότητας, τρόπους αντιμετώπισης αλλά και οποιαδήποτε πληροφορία που ένας αναλυτής θεωρεί χρήσιμη να αναφερθεί. Ο στόχος των παραπάνω είναι η θωράκιση των πληροφοριακών συστημάτων ενάντια των κακόβουλων δραστηριοτήτων που έχουν δηλωθεί στο MISP. Επιπρόσθετα, στην πλατφόρμα μπορούν να εισαχθούν και ενδείκτες παραβίασης, τους οποίους μπορεί ο χρήστης να τους εξάγει και να τους χρησιμοποιήσει σε αρκετά συστήματα ασφάλειας. Ενδεικτικά μπορούν να εισαχθούν κανόνες YARA, Suricata, Bro, κακόβουλες IP και άλλα. Για να αποκτήσει πρόσβαση στη πλατφόρμα ο ενδιαφερόμενος, χρειάζεται να αποκτήσει διαπιστευτήρια (user name και password) που θα τα ορίσει ο διαχειριστής του MISP.

Κάθε πλατφόρμα MISP έχει τη δυνατότητα να συγχρονίζεται με άλλες πλατφόρμες MISP καθώς υπάρχει προ εγκατεστημένος μηχανισμός ο οποίος φροντίζει για την ανταλλαγή των καταχωρημένων γεγονότων (Events) καθώς και χαρακτηριστικών (Attributes) αυτών. Με αυτόν τον τρόπο επιτυγχάνεται η δημιουργία ενός ισχυρού δικτύου ανταλλαγής πληροφοριών ανάμεσα σε διαφορετικούς οργανισμούς.

Στόχος της παρούσας διπλωματικής είναι η υλοποίηση ενός λογισμικού που θα εξάγει στοχευμένες πληροφορίες από το MISP και οι οποίες θα χρησιμοποιούνται για εκτεταμένο έλεγχο του συστήματος. Συγκεκριμένα το λογισμικό θα αντλεί κανόνες YARA και κακόβουλες IP, θα ελέγχει τα τελευταία προγράμματα που εκτελέστηκαν, τις διεργασίες που εκτελούνται την κάθε δεδομένη στιγμή αλλά και τις συνδέσεις που τυχόν υπάρχουν. Η εφαρμογή θα ενημερώνεται συνέχεια για νέα γεγονότα που εισάγονται στην πλατφόρμα MISP και θα προβαίνει σε απαραίτητες ενέργειες στην περίπτωση που βρεθεί κάποιο εύρημα. Οι ενέργειες που πραγματοποιεί θα είναι οι εξής :

- Ελέγχει τα τελευταία εκτελέσιμα αρχεία που έτρεξαν στο σύστημα και σε περίπτωση που βρει κακόβουλο αρχείο, τερματίζει τη διεργασίας που το εκτελεί και το διαγράφει από τη θέση που βρίσκεται.
- Παράλληλα αντιγράφει το κακόβουλο αρχείο και το τοποθετεί σε συγκεκριμένη θέση με την μορφή zip.

- Τέλος συντάσσει μία αναφορά για το εν λόγο αρχείο προκειμένου να χρησιμοποιηθεί για περαιτέρω ανάλυση.

Αντίστοιχες λειτουργίες γίνονται και κατά την εύρεση κακόβουλης διαδικτυακής δραστηριότητας.

2. Malware Information Sharing Platform and Threat Sharing (MISP)

Η πλατφόρμα MISP (Malware Information Sharing Platform and Threat Sharing) είναι μια ανοιχτού λογισμικού πλατφόρμα η οποία συλλέγει, αποθηκεύει, διαμοιράζεται και συσχετίζει πληροφορίες, ενδείκτες ασφαλείας (Indicators of Compromise) που αφορούν σε περιστατικά κυβερνοεπιθέσεων καθώς και περιστατικά ανάλυσης κακόβουλου λογισμικού.

2.1 Βασικές λειτουργίες του MISP

Η πλατφόρμα MISP αποτελεί ένα χρήσιμο εργαλείο για τις ομάδες αντιμετώπισης κυβερνοπεριστατικών αλλά και για όσους ενδιαφέρονται ή τους αφορά να γνωρίζουν κακόβουλες ενέργειες που εντοπίστηκαν στον οργανισμό που έχει εγκατασταθεί η εν λόγο πλατφόρμα. Αποτελεί μία αποτελεσματική βάση δεδομένων για την αποθήκευση τεχνικών και μη τεχνικών πληροφοριών ενός κακόβουλου λογισμικού, δείγματα αυτού, πληροφορίες για τις κακόβουλες απειλές, καθώς και γενικές πληροφορίες της κακόβουλης δραστηριότητας. Παρέχει συσχετίσεις ανάμεσα στα γνωρίσματα (attributes) και στους ενδείκτες (indicators) μιας καταχώρησης των χρηστών, με σκοπό τον εύκολο εντοπισμό πληροφοριών που συσχετίζονται και την αποφυγή διπλοεγγραφών. Συγχρόνως, αποθηκεύει πληροφορίες ως αποδεκτή δραστηριότητα (Whitelisting) προκειμένου να μην καταχωρηθεί εσφαλμένα ως κακόβουλη.

Πέραν των παραπάνω, διαθέτει προ-εγκατεστημένες λειτουργίες διαμοιρασμού για την ευκολότερη ανταλλαγή γεγονότων (events) και γνωρισμάτων (attributes) ανάμεσα σε διαφορετικά στιγμιότυπα του MISP και κατ' επέκταση ανάμεσα σε διαφορετικούς οργανισμούς. Επιπλέον παρέχει μηχανισμούς επιλογής και καθορισμού των συνδεόμενων στιγμιοτύπων, όπως και των καταχωρημένων γεγονότων που θα αποστέλλονται ή θα λαμβάνονται. Το γραφικό περιβάλλον είναι φιλικό προς τον χρήστη δίνοντας του την δυνατότητα για εύκολη πλοήγηση στην πλατφόρμα, καταγραφή γεγονότων και αξιοποίηση όλων των λειτουργιών που παρέχει. Ακόμη, δίνει στο χρήστη τη δυνατότητα εξαγωγής αποθηκευμένων γεγονότων, ως πληροφορίες για συστήματα εντοπισμού κακόβουλης δραστηριότητας (IDS(Suricata, Snort, Bro), OpenIOC, YARA , κ.α.), ώστε να μπορούν να αλληλεπιδράσουν και να χρησιμοποιηθούν από άλλα συστήματα.

Τα γεγονότα μπορούν να εισαχθούν μαζικά (JSON ή XML), ως απλό κείμενο, είτε απο άλλες πηγές (OpenIOC, GFI sandbox, ThreatConnect CSV). Στην περίπτωση που ο χρήστης εισάγει ένα γεγονός ως απλό κείμενο το MISP παρέχει αυτοματοποιημένες διαδικασίες για την μετατροπή του σε ένα γεγονός του MISP. Χαρακτηρίζεται από αυτοματοποιημένο, εύκολο και διακριτό τρόπο με σκοπό τη συνεργασία των χρηστών στα καταχωρημένα γεγονότα επιτρέποντας τους να προτείνουν αλλαγές σε αυτά. Εκτός των παραπάνω, παρέχει και τη δυνατότητα ψευδό–ανώνυμης ανάθεσης γεγονότων σε άλλους οργανισμούς

Παράλληλα, η ταξινόμηση των γεγονότων με ετικέτες που είναι προ εγκατεστημένες στο MISP είναι εφικτή. Επεκτασιμότητα της λειτουργίας αυτής είτε εισάγοντας νέες ετικέτες με χρήση χρωματικών διακρίσεων και ονομασιών που εξυπηρετούν την εκάστοτε ανάγκη. Το MISP παρέχει βιβλιοθήκες με επιπλέον πληροφορίες οι οποίες ονομάζονται MISP galaxies. Είναι άμεσα συνδεόμενες με ήδη υπάρχουσες βιβλιοθήκες γνώριμων κακόβουλων λογισμικών, κακόβουλων δραστηριοτήτων, κακόβουλων αντιπάλων (threat actor) κ.α. όπου οι καταχωρήσεις των ευρημάτων ενός χρήστη μπορούν εύκολα να συσχετισθούν με αυτές. Σε περίπτωση που η σύνδεση στην διεπαφή του MISP μπορεί να χρησιμοποιηθεί ένα SSL πιστοποιητικού, το οποίο δύναται να είναι είτε υπογεγραμένο από τον φορέα δημιουργίας του στιγμιοτύπου (Self-signed Certificate) είτε από έναν έγκυρο πάροχο.

Προκειμένου να αξιοποιηθεί η δυνατότητα αποστολής ηλεκτρονικών μηνυμάτων μέσω της πλατφόρμα, απαιτείται η σύνδεση του MISP με έναν mail Server. Το MISP κατά την εγκατάσταση του εγκαθιστά το Postfix mail server. Η αποστολή και λήψη των μηνυμάτων γίνεται σε απλό κείμενο. Ωστόσο, υπάρχει η δυνατότητα χρήσης ενός pgp κλειδιού προκειμένου να κρυπτογραφηθεί το εν λόγο μήνυμα. Το κλειδί αυτό είναι ξεχωριστό σε κάθε χρήστη ο οποίος είναι υπεύθυνος για την δημιουργία του και την γνωστοποίηση του στον διαχειριστή του MISP. Το pgp κλειδί μπορεί να αποθηκευτεί σε ένα repository όπως αυτό του MIT και να αντληθεί αυτόματα από την πλατφόρμα.

Υπάρχει επιπλέον η δυνατότητα χρήσης ετικετών (tags) για την εύκολη αναγνώριση ενός γεγονότος, είτε από τις προ εγκατεστημένες επιλογές, είτε δημιουργώντας νέες από τον διαχειριστή της καθώς και ανάθεσή αυτών στους οργανισμούς οι οποίοι εχου δημιουργηθεί στο MISP με μορφή **οργανισμός:κατηγορία=”γεγονός”**. Οι ετικέτες μπορούν να διαθέτουν

χρωματικούς προσδιορισμούς και να ανατεθούν σε συγκεκριμένους χρήστες για την χρήση τους.

Μία από τις σημαντικότερες δυνατότητες του MISP είναι η συνεργασία δύο η περισσοτέρων στιγμιότυπων. Με αυτόν τον τρόπο τα στιγμιότυπα μπορούν να ανταλλάσσουν πληροφορίες και διαφορετικοί οργανισμοί δύναται να ενημερώνονται μεταξύ τους. Οι οργανισμοί πρέπει να δημιουργήσουν αντίστοιχους χρήστες υπαγόμενοι σε αντίστοιχους οργανισμούς προκειμένου να προβαίνουν στον εν λόγο αυτοματοποιημένο συγχρονισμό. Σημαντική είναι επίσης και η ανταλλαγή του **universally unique identifier** (**UUID**) προκειμένου να αναγνωρίζεται η μοναδικότητα του απομακρυσμένου χρήστη με τον οποίο γίνεται η ανταλλαγή πληροφοριών.

Το MISP παρέχει την δυνατότητα να δημιουργίας Modules προσωπικής επιλογής του καθενός με σκοπό την αυτοματοποίηση των λειτουργιών, σύμφωνα πάντοτε με τις ανα΄γκες του χρήστη. Παρέχει ακόμη τη βιβλιοθήκη PyMISP η οποία χρησιμοποιείται από την γλώσσα προγραμματισμού Python. Η εν λόγω βιβλιοθήκη δίνει σημαντικές δυνατότητες αυτοματοποίησης όλων των διεργασιών στις οποίες οι προβαίνουν οι διαχειριστές μέσα στο γραφικό περιβάλλον.

3. YARA Rules

Ο σκοπός των κανόνων Yara είναι να βοηθήσουν τους αναλυτές να πιστοποιήσουν και να κατηγοριοποιήσουν δείγματα κακόβουλου λογισμικού. Με τους κανόνες YARA μπορούμε να περιγράψουμε και να πιστοποιήσουμε κακόβουλα λογισμικά βάση κειμένων (strings) που βρίσκονται στο σώμα τους ή δυαδικά μοτίβα (binary pattern). Κάθε κανόνας (rule) αποτελείται από ένα σετ κειμένων και λογικών εκφράσεων.

Ενδεικτικά παρατίθεται το ακόλουθο παράδειγμα:

```
rule example
{
   meta:
      description = "This is just an example"
      thread_level = 3
      in_the_wild = true
   strings:
      $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
      $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
      $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
   condition:
      $a or $b or $c
}
```

Παρατηρούμε ότι χωρίζεται σε τρία μέρη. Τα meta (metadata), strings και condition ενώ το όνομα του κανόνα ορίζεται στην αρχή και έξω από τα άγκιστρα (rule example). Κατά την ανάλυση του κακόβουλου λογισμικού λαμβάνουν χώρα μόνο το πεδίο strings και το πεδίο conditions, ενώ τα meta προσδιορίζονται μόνο για πληροφορίες του αναλυτή.

Μια τυπική δομή ενός YARA απαιτεί να σημειώνονται κάτω από την επιλογή strings, τα μοτίβα που επιθυμεί κανείς να αναζητήσει μέσα σε ένα αρχείο. Τα κριτήρια ελέγχου πρέπει να ξεκινούν με το σύμβολο του δολαρίου “$” και στην συνέχεια το όνομα του κριτηρίου. Για την αναζήτηση ενός κειμένου μέσα στο σώμα του κακόβουλου λογισμικού το αναγράφουμε ανάμεσα σε διπλά αποσιωπητικά. Αντίστοιχα, για την αναζήτηση ενός μοτίβου απο bytes αναγράφεται ανάμεσα σε αγκύλες. Ενα παράδειγμα φαίνεται παρακάτω

$my_text_string = "text here" // Δημιουργία ενός κριτηρίου ελέγχου τύπου κειμένου
$my_hex_string = { E2 34 A1 C8 23 FB } // Δημιουργία ενός κριτηρίου ελέγχου τύπου bytes

Στο πεδίο Conditions μπορεί κανείς να αναγράψει τις συνθήκες των κριτηρίων που επιθυμεί να αναζητήσει ενώ έχει τη δυνατότητα να ορίσει και λογικές τιμές (AND, OR). Αντίστοιχα, ο χρήστης μπορεί να χρησιμοποιήσει επιπλέον δεσμευμένες λέξεις ή μόνο το κριτήριο που επιθυμεί να αναζητήσει. Ένα παράδειγμά ενός μοναδικού κριτηρίου είναι το παρακάτω:

```
rule example1
{
	strings:
	  $hex_string = { F4 23 [4-6] 62 B4 }
	condition:
	  $hex_string
}
```

Μερικές δεσμευμένες λέξεις που δύναται να χρησιμοποιηθούν στο πεδίο conditions είναι “any of them”, “or”, “and” “of” “filesize” και άλλες.

Είναι σκόπιμο να γίνει αναφορά σε κάποιες από αυτές και τον τρόπο λειτουργίας τους καθώς χρησιμοποιούνται συνέχεια κατά την δημιουργία των κανόνων.

Η πιο συνηθισμένη είναι η εντολή ***any of them*** όπου ορίζουμε ότι έστω ένα από τα δοθέντα μοτίβα είναι αρκετό για να ισχύσει ο κανόνας.

Για τον ορισμό του μέγεθος του αρχείου που επιθυμεί ο χρήστης να αναζητήσει, γίνεται χρήση της εντολής ***filesize.*** Με την εντολή αυτή ορίζεται αν το αρχείο είναι μικρότερο, μεγαλύτερο ή ενδιάμεσα από δοθείσες τιμές, χρησιμοποιώντας τους αντίστοιχους τελεστές σύγκρισης. Ένα παράδειγμα είναι ***filesize < 1000KB*** που ορίζει ότι το μέγεθος του αρχείου να είναι μικρότερο απο 1000 kilobytes. Αντίστοιχα είναι εφικτός και ο ορισμός της σχετικής

μονάδας μέτρησης σε Megabyte (MB) και Gigabyte (GB) αντικαθιστώντας το «KB» σε «MB» ή «GB».

Με τη χρήση της Εντολή ***of*** όπου ορίζεται το ποσοστό μοτίβων που επιθυμεί ο χρήστης να ικανοποιηθούν. Ένα παράδειγμα είναι ***2 of ($a,$b,$c).*** Σε αυτό το παράδειγμα γίνεται αναζήτηση δύο ή παραπάνω από τα δοθέντα μοτίβα εντός της παρένθεσης ενώ για τον ορισμό της παραμέτρου επιλέγουμε ***2 of them*** ή ***2 of ($*)*** δηλαδή 2 από όλα τα καταγεγραμμένα μοτίβα στο πεδίο strings.

Οι εντολές ***and*** και ***or*** δίνουν τη δυνατότητα λογικής έκφρασης όσον αφορά στο ποια μοτίβα θα ελεγχθούν. Ένα παράδειγμα είναι ***$a and $b*** όπου για να ισχύσει ο κανόνας πρέπει να ισχύουν και τα δύο μοτίβα ή ***$a or $b*** όπου για να ισχύσει ο κανόνας πρέπει να ισχύει είτε το ένα είτε το άλλο μοτίβο.

Σε όλες τις παραπάνω εντολές υπάρχει η δυνατότητα χρήσης παρενθέσεων και η δημιουργία πιο σύνθετες εκφράσεων. Για παράδειγμα έστω ότι έχουμε τέσσερα μοτίβα $a,$b,$c,$d μια έγκυρη λογική έκφραση θα μπορούσε να είναι η παρακάτω:

(($a and $c) or ($c and $d)) and filesize>100MB

Στο παράδειγμα ο κανόνας θα ισχύει αν ισχύ το μοτίβο ***$a*** και ***$c*** ή αν ισχύ το μοτίβο ***$c*** και ***$d***. Σε κάθε περίπτωση το μέγεθος του αρχείου πρέπει να είναι μεγαλύτερο από ***100 megabytes***

Όπως αναφέρθηκε στο πεδίο strings εισάγουμε είτε ελεύθερο κείμενο είτε αλληλουχία από bytes ως μοτίβα για την αναζήτηση κακόβουλου λογισμικού. Προκειμένου να γίνει μια πιο σύνθετη αναζήτηση είναι δυνατόν να χρησιμοποιηθούν επιπλέον εντολές όπως wild Carts. Έτσι αν η αλληλουχία των μοτίβων είναι {***AA BB CC 11 22 33 44 55}*** μπορούμε να χρησιμοποιήσουμε το σύμβολο του αγγλικού ερωτηματικού “?” όταν δεν γνωρίζουμε το byte εκείνου του σημείου που τοποθετείται. Για παράδειγμα, στο προηγούμενο μοτίβο αν αντικαταστήσουμε το 11 με ?? θα έχουμε το παρακάτω μοτίβο ***{AA BB CC ?? 22 33 44 55}*** και θα ισχύει για οποιαδήποτε αλληλουχία απο bytes στο σημείο των ερωτηματικών. Μπορούμε να χρησιμοποιήσουμε από κανένα έως όσα επιθυμούμε και σε όσες θέσεις θέλουμε. Ένα έγκυρο μοτίβο είναι ***{A? BB CC ?? 22 ?3 44 55 ?? ??}.***

Μια άλλη επιλογή που μπορούμε να χρησιμοποιήσουμε είναι η κάθετη μπάρα “|”. Στην περίπτωση αυτή χρησιμοποιούμε το λογικό ***“ή”*** στα μοτίβα των bytes. Παράδειγμα χρήσης αυτής της δυνατότητας από την παραπάνω αλληλουχία είναι η ακόλουθη:

{***AA BB CC (11|22) 22 33 44 55}*** όπου θα ισχύει το μοτίβο {***AA BB CC 11 22 33 44 55}*** αλλά και το μοτίβο {***AA BB CC 22 22 33 44 55}*** .

Παρατηρούμε ότι η επιλογή αυτή πρέπει να είναι σε παρενθέσεις. Επίσης μπορούν να μπουν περισσότερα από ένα λογικό “ή” αλλά και περισσότερα από ένα μοτίβο. Για παράδειγμα το μοτίβο {***AA BB CC (11|22 33 44 | FF) 22 33 44 55}*** αναλύεται στα εξής μοτίβα:

{AA BB CC 11 22 33 44 55}, {***AA BB CC 22 33 44 22 33 44 55},*** {***AA BB CC FF 44 55}***

Συνοψίζοντας μπορούμε να συνδυάσουμε τις δύο παραπάνω εντολές με οποίον τρόπο επιθυμούμε. Για παράδειγμα το μοτίβο {***AA BB CC (11|22 ?? 44 | FF| ??) 22 33 44 55}*** αναλύεται στα εξής μοτίβα

{***AA BB CC 11 22 33 44 55}***
{AA BB CC 22 ?? 44 22 33 44 55}
{AA BB CC FF 22 33 44 55}
{AA BB CC ?? 22 33 44 55}

Τέλος με τις αγκύλες “[ ]” μπορούμε να ορίσουμε περιοχές από bytes. Δηλαδή, αν στο παράδειγμα βάλουμε το [4] θα ισχύει για οποιοιδήποτε αλληλουχία από τέσσερα bytes. Το παράδειγμα θα μπορούσε να γραφτεί ως εξής ***{AA BB CC [4] 22 33 44 55}*** το οποίο ισοδυναμεί με ***{AA BB CC ?? ?? ?? ?? 22 33 44 55}***

Σε αυτήν την λειτουργία μπορούμε να ορίσουμε και πεδίο μεγέθους από οποιαδήποτε bytes δηλαδή αν θέλουμε ένα πεδίο από τέσσερα έως έξι byes το ορίζουμε ως εξής [4-6]. Το μοτίβο που αναλύουμε, θα μπορούσε να παρουσιαστεί και με τον παρακάτω τρόπο {***AA BB CC 11 [4-6] 33 44 55}*** και ισοδυναμεί με τα επόμενα μοτίβα

{AA BB CC 11 ?? ?? ?? ?? 33 44 55}
{AA BB CC 11 ?? ?? ?? ?? ?? 33 44 55}
{AA BB CC 11 ?? ?? ?? ?? ?? ?? 33 44 55}

Ένα έγκυρο Yara rule που περιλαμβάνει όλους τους προαναφερθέντες κανόνες είναι το παρακάτω:

```
rule example2
{
    strings:
        $string = “asde#+”
        $string1 = “while True”
      $hex_string = { 66 23 AA 12 39 62 B4 }
        $hex_string1 = { ?? F4 23 [4-6] 62 B4 }
        $hex_string2 = { FA (12 ?? | AB 12 33 [2] AA | FF ) [4-6] 62 B4 }

    condition:
      3 of them and filesize>12MB}
```

Οι κανόνες Yara αποθηκεύονται σαν ένα απλό αρχείο κειμένου με κατάληξη .yar ή .yara. Οι δεσμευμένες λέξεις των Yara rule παρουσιάζονται στον επόμενο Πίνακα:

all	and	any	ascii	at	condition	contains
entrypoint	false	filesize	fullword	for	global	in
import	include	int8	int16	int32	int8be	int16be
int32be	matches	meta	nocase	not	or	of
private	rule	strings	them	true	uint8	uint16
uint32	uint8be	uint16be	uint32be	wide	xor	

Πίνακας 2.1: Δεσμευμένες λέξεις των κανόνων YARA

3.1 Εγκατάσταση κανόνων YARA

Τα Yara rule μπορούν να χρησιμοποιηθούν και με την γλώσσα προγραμματισμού Python όπου κατόπιν εγκατάστασης καλείται σαν ένα απλό module.

Η εγκατάσταση σε Linux μηχανήματα πραγματοποιείται με τη βοήθεια της εντολής:

-> pip3 install yara-python

Για εγκατάσταση σε Windows κατεβάζουμε από το αποθετήριο των εκτελέσιμων αρχείων εγκατάστασης των κανόνων YARA (σύμφωνα με την έκδοση της python που χρησιμοποιούμε) και το εγκαθιστούμε.

3.2 Χρήση YARA και PYTHON

Για να καλέσουμε την βιβλιοθήκη στην εφαρμογή μας γράφουμε

>> import yara

Για να δημιουργήσουμε ένα αντικείμενο yara με έναν κανόνα που έχουμε δημιουργήσει πληκτρολογούμε

>> rule = yara.compile(filepath="Διαδρομή αρχείου")// Η διαδρομή του yara αρχείου

Στην Επιλογή filepaths μπορούμε να εισάγουμε ένα dictionary το οποίο να έχει το όνομα του κανόνα και την διαδρομή του.

>> rules = yara.compile(filepaths={'namespace1':'Διαδρομή 1ου αρχείου', 'namespace2':'Διαδρομή 2ου αρχείου'})

Για να ελέγξουμε αν ένα αρχείο εμπίπτει στους κανόνες που ορίσαμε πληκτρολογούμε

>> rules.match("Διαδρομή αρχείου")

ή μπορούμε να ανατρέξουμε στα byte ενός αρχείου

>>with open("Διαδρομή αρχείου", 'rb') as f:

>> *matches = rules.match(data=f.read())*

Η βιβλιοθήκη μπορεί να ελέγχει διεργασίες που τρέχει το σύστημα παρέχοντας σαν παράμετρο το αναγνωριστικό της (proceess id ή pid). Τη διεργασία pid 1234 μπορούμε να την ελέγξουμε πληκτρολογώντας

>> ***matches = rules.match(pid=1234)***

Τέλος, για μεταγενέστερη χρήση μπορούμε να αποθηκεύσουμε τους κανόνες που έχουν φορτωθεί στην εφαρμογή μας και να τους καλέσουμε σε δεύτερο χρόνο. Για να το πετύχουμε αυτό πληκτρολογούμε την εντολή

>> rules.save('Διαδρομή αρχείου που θα δημιουργηθεί')

Για να φορτώσουμε στο πρόγραμμά μας τους ήδη αποθηκευμένους κανόνες πράττουμε ως ακολούθως

>> rules = yara.load('***Διαδρομή αρχείου δημιουργήθηκε με την εντολή yara.save***')

Σε περίπτωση που ένας ή περισσότεροι κανόνες ενεργοποιηθούν με ένα αρχείο ή με μια διεργασία που ελέγχεται, μας επιστρέφει μια λίστα με τα ονόματα των κανόνων.

4. Εφαρμογή

Η Εφαρμογή Υλοποιήθηκε και δοκιμάστηκε σε Windows 10 περιβάλλον ενώ το MISP εγκαταστάθηκε σε Ubuntu Server 18.04

Όπως αναφέρθηκε η χρήση του MISP είναι μόνο για τον διαμοιρασμό πληροφοριών μεταξύ χρηστών οργανισμών και διαφορετικών στιγμιότυπων της πλατφόρμας. Η εφαρμογή αποσκοπεί στην εκμετάλλευση αυτής της επικοινωνίας και χρήση των αποθηκευμένων πληροφοριών προκειμένου να γίνει αναγνώριση και αντιμετώπιση των σχετικών απειλών που πιθανόν να έχει το λειτουργικό μας σύστημα.

Η πλατφόρμα σε κάθε λογαριασμό που δημιουργεί ένα authentication key το οποίο επιτρέπει στον χρήστη να μπορεί να αλληλοεπιδρά με την πλατφόρμα μέσω του module PyMISP. Το authentication key είναι μοναδικό σε κάθε χρήστη και χρησιμοποιώντας τα διαπιστευτήρια του το εξάγει από την πλατφόρμα. Σε περίπτωση που ο διαχειριστής δεν επιθυμεί να έχει πρόσβαση ο χρήστης στην πλατφόρμα, μόνο με το authentication key μπορεί να χρησιμοποιήσει την εφαρμογή. Αυτό μπορεί να επιτευχθεί δημιουργώντας έναν λογαριασμό και αποστέλλοντας στον χρήστη μόνο το authentication key της πλατφόρμας και όχι τα διαπιστευτήρια για αυτήν.

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκε μια πληθώρα από modules. Τα σημαντικότερα από αυτά είναι το **yara module** το οποίο χειρίζεται τους κανόνες Yara, τα **modules winreg** και **win32evtlog** που είναι υπεύθυνα για την επικοινωνία με την registry του συστήματος, το **module psutil** το οποίο αλληλοεπιδράει με τις διεργασίες που τρέχουν στο σύστημα, το **module logging** για την δημιουργία και συλλογή logs και τέλος το **PyMISP** για την αλληλεπίδραση της εφαρμογής με το MISP.

Προκειμένου να είναι εύχρηστο το πρόγραμμα και να μην χρειάζεται ο χρήστης να προβεί σε πολύπλοκες και χρονοβόρες ρυθμίσεις, δεν χρησιμοποιήθηκαν βάσεις δεδομένων ενώ για την αποθήκευση των διάφορων πληροφοριών καθώς και των στοιχείων ελέγχων δημιουργήθηκαν απλά αρχεία κειμένου τα οποία στο σώμα τους αποθηκεύουν πληροφορίες σε JSON μορφή.

Η εφαρμογή χωρίζεται σε τέσσερα νήματα threads όπου το καθένα δουλεύει ανεξάρτητο από τα άλλα. Αλληλοεπιδρούν με τις πληροφορίες των προαναφερθέντων

αρχείων και στοιχείων του συστήματος. Το πρώτο νήμα είναι ο περιοδικός Έλεγχος νέο-αποθηκευμένων πληροφοριών, όχι μόνο σε ότι αφορά Yara Rules αλλά και σε χαρακτηρισμένες ως κακόβουλες IP. Το δεύτερο νήμα επικεντρώνεται στην συλλογή διαδρομών από τα τελευταία προγράμματα που ο χρήστης έχει εκτελέσει, ενώ παράλληλα προβαίνει στον έλεγχο αυτών με κανόνες Yara που έχουν ληφθεί από το MISP. Το τρίτο νήμα διαρκώς προβαίνει στην αναζήτηση των διεργασιών που εκτελούνται και τις ελέγχει με τους κανόνες Yara που έχουν ληφθεί από το MISP. Τέλος το τέταρτο νήμα κάνει συνεχόμενο έλεγχο των συνδέσεων που δημιουργούν οι εγκατεστημένες εφαρμογές με ταυτόχρονο έλεγχο αυτών βάση των κακόβουλων IP που έχουν αντληθεί από το MISP.

Η εφαρμογή ξεκινάει ελέγχοντας και δημιουργώντας τα απαραίτητα για την λειτουργία του αρχεία και φακέλους. Πρώτα δημιουργείται το αρχείο **keys.json** στο οποίο εισάγονται οι απαραίτητες πληροφορίες για την σύνδεσή του στο MISP καθώς και διαχειριστικές πληροφορίες για τα αποθηκευμένα Event της πλατφόρμας. Απαραίτητες πληροφορίες για τη σύνδεση της εφαρμογής με το MISP είναι το URL και το Authentication Key. Αυτόματα ορίζεται μία ημερομηνία στο πεδίο **search_from** κατά την οποία υποδηλώνει από ποια ημερομηνία θα αρχίζει το πρόγραμμα να αναζητά νέα event. Κατά την πρώτη εκκίνηση του προγράμματος η ημερομηνία ξεκινάει από το 2000 προκειμένου να συλλέξει όλα τα γεγονότα που είναι αποθηκευμένα.

Στην συνέχεια δημιουργούνται τα αρχεία **paths.json** και το **pids.json.** Στο αρχείο **paths.json** αποθηκεύονται οι έγκυρες αλλά και οι μη έγκυρες οι διαδρομές των εκτελέσιμων αρχείων (περιπτώσεις που η registry έχει κρατήσει ως εγγραφή μια διαδρομή ενός εκτελέσιμου αρχείου ενώ αυτό έχει μεταφερθεί η διαγραφεί). Το αρχείο **pids.json** αποθηκεύει το process id (pid) και την ώρα δημιουργίας όλων των διεργασιών που τρέχει το σύστημα σε μορφή Τούπλας. Τέλος δημιουργούνται τέσσερα αρχεία που αποθηκεύουν logs προκειμένου ο χρήστης να έχει μία εικόνα του συστήματος. Τα αρχεία συλλέγουν πληροφορίες από τον έλεγχο των εκτελέσιμων, των διεργασιών, των συνδέσεων καθώς και γενικές πληροφορίες του συστήματος. Οι φάκελοί που δημιουργούνται είναι ο ioc ο οποίος αποθηκεύει τους ενδείκτες παραβίασης που λαμβάνει από το MISP και ο bad_files στον οποίο μεταφέρονται σε μορφή zip τα κακόβουλα εκτελέσιμα που βρίσκει η εφαρμογή.

Η εφαρμογή ξεκινάει καλώντας την κλάση MISPReceiver η οποία δημιουργεί ένα στιγμιότυπο σύνδεσης με το MISP. Το στιγμιότυπο αυτό λειτουργεί ως ένα νέο thread και εκτελείτε μόνο του και ανεξάρτητα από την βασική ροή του προγράμματος. Η συγκεκριμένη

κλάση, προκειμένου να συνδεθεί στην πλατφόρμα χρησιμοποιεί το authentication key που παρείχαμε και το URL. Αφού δημιουργηθεί η σύνδεση, το πρόγραμμα αντλεί την ημερομηνία από το αρχείο keys.json που του ορίζει από ποια χρονολογία να αρχίσει να αντλεί τα αποθηκευμένα event. Στην αναζήτηση των event ζητείτε να του επιστραφούν όλα τα event που έχουν attributes «YARA» και «ip-dst». Το Attribute «YARA» δηλώνει ότι η πληροφορία που έχει είναι ένας κανόνας YARA ενώ το «ip-dst» δηλώνει ότι η πληροφορία που έχει είναι μία κακόβουλη IP προορισμού.

Κατά την δημιουργία αυτής της κλάσης καλείται η συνάρτηση **get_iocs_last** η οποία αντλεί τους επιθυμητούς ενδείκτες. Αυτή η πληροφορία ανανεώνεται βάση της ημερομηνίας του τελευταίου Event μείον δέκα μέρες. Αφαιρούμε δέκα μέρες ώστε αν εισαχθεί εκ των υστέρων ένα νέο attribute σε κάποιο event η εφαρμογή να μπορέσει να το αντλήσει. Μόλις ολοκληρωθεί η διαδικασία στο αρχείο keys.json αποθηκεύεται το id του Event καθώς και τα id των attributes που αντιστοιχεί στο κάθε event.

Η εφαρμογή δημιουργεί και αποθηκεύει τοπικά αντίστοιχα αρχεία με κανόνες YARA καθώς και ένα αρχείο με πληροφορίες για τις κακόβουλες IP. Για να μπορέσουμε να χρησιμοποιήσουμε τα αρχεία YARA η εφαρμογή καλεί τις διεργασίες repair_yara_rule και check_yara. Η πρώτη εξαλείφει χαρακτήρες που κατόπιν άντλησης των κανόνων επηρεάζουν την σωστή λειτουργία τους και η δεύτερη ελέγχει την ορθότητα αυτών για αποφυγή περιπτώσεων που ο αναλυτής εισάγει εσφαλμένο κανόνα και δεν μπορεί να μεταγλωττιστεί. Αφού ολοκληρωθεί ο έλεγχος, οι κανόνες που μεταγλωττίζονται ορθά συνενώνονται και αποθηκεύονται σε ένα μοναδικό αρχείο yara προκειμένου όλα τα thread να έχουν πρόσβαση σε αυτό. Ένας χρήστης μπορεί να εισάγει και δικούς του κανόνες YARA πέραν από αυτούς που αντλεί από το MISP στον ίδιο φάκελο. Η διαδικασία ελέγχου μεταγλώττισης και συνένωσης θα είναι η ίδια.

4.1 Έλεγχος των εκτελέσιμων αρχείων

Για τον έλεγχο των εκτελέσιμων αρχείων δημιουργείται το 2ο Thread. Καλείται η διεργασία collect() η οποία αντλεί πληροφορίες από το αρχείο paths.json το οποίο διατηρεί τις διαδρομές των αρχείων που έχουν εκτελεστεί τοπικά και έχουν ελεγχθεί. Για να μπορέσουμε να βρούμε τις διαδρομές, ελέγχουμε την registry όπου σε συγκεκριμένα κλειδιά αποθηκεύεται η πληροφορία που ζητάμε. Για να μπορέσουμε να αποκτήσουμε πρόσβαση χρησιμοποιήσαμε το module win32evtlog και το winreg. Τα κλειδιά αυτά είναι:

- HKEY_CURRENT_USER.Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Store
- HKEY_CURRENT_USER.Software\Classes\Local Settings\Software\Microsoft\Windows\Shell\MuiCache
- HKEY_USERS.S-1-5-21-733142866-2450513131-3585932315-1001_Classes\Local Settings\Software\Microsoft\Windows\Shell\MuiCache
- HKEY_CLASSES_ROOT.Local Settings\Software\Microsoft\Windows\Shell\MuiCache
- HKEY _LOCAL_MACHINE.Software\Microsoft\Windows\CurrentVersion\Run
- HKEY_CURRENT_USER.Software\Microsoft\Windows\CurrentVersion\Run
- HKEY_LOCAL_MACHINE,Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKEY_CURRENT_USER.Software\Microsoft\Windows\CurrentVersion\RunOnce

Η εφαρμογή μπορεί να βρει και να αντλήσει διαδρομές από powershell script που πιθανόν να εκτελέστηκαν έχοντας κακόβουλο κώδικα στο περιεχόμενό τους.

Η διεργασία collect που αναφέρθηκε παραπάνω, καλεί τις διεργασίες **get_ps**, **get_paths**, **find_procs_by_name** και δημιουργεί μια λίστα ευρείας χρήσης όπου διατηρεί τις διαδρομές που επρόκειτο να ελεγχθούν. Η πρώτη ελέγχει πιθανά powershell script που έχουν εκτελεστεί και αποθηκεύει την διαδρομή τους. Υπάρχει πιθανότητα το script να έχει διαγράψει τον εαυτό του και σε αυτήν την περίπτωση θα καταγράφεται στο file.log αρχείο. Τα αποτελέσματα εισάγονται στην λίστα με τις διαδρομές που θα ελεγχθούν.

Η δεύτερη συλλέγει όλες τις διαδρομές των τελευταίων εκτελέσιμων αρχείων του συστήματος μέσω τις registry και τα εισάγει στην λίστα με τις διαδρομές που θα ελεγχθούν. Σε περίπτωση που ένα εκτελέσιμο έτρεξε από το δίκτυο ελέγχεται και αναγράφεται σαν event στο αρχείο που file.log ως μια νέα πληροφορία για μελλοντική ανάλυση.

Τέλος η Τρίτη διεργασία καλώντας την **psutil.process_iter()** αντλεί όλες τις διεργασίες που εκτελούνται. Η πληροφορία που μας επιστρέφεται διατηρεί τις διαδρομές από τα εκτελέσιμα αρχεία που τις δημιούργησαν. Και σε αυτή την περίπτωση, οι διαδρομές αποθηκεύονται στην λίστα με τις διαδρομές που θα ελεγχθούν. Με σκοπό την απογυφή των όποιων διπλοεγγραφών, με ταυτόχρονη κατανάλωση πόρων του συστήματος λόγω των

περιττών ελέγχων, δημιουργήθηκε η διεργασία **inser_to_paths** που ελέγχει αν υπάρχει ήδη η διαδρομή στο αρχείο paths.json.

Αφού η εφαρμογή συλλέξει όλες τις διαδρομές που μας ενδιαφέρουν φορτώνει στην μνήμη το αρχείο **saved_yara_file.yara** το οποίο έχει όλους τους κανόνες που αντλήθηκαν από το MISP. Ελέγχει ένα προς ένα τα συλλεχθέντα αρχεία από τις παραπάνω διεργασίες με αυτά που άντλησε από το αρχείο paths.json. Αν συμπίπτουν οι διαδρομές σημαίνει ότι το αρχείο έχει ξανά ελεγχθεί και η εφαρμογή το προσπερνάει, αλλιώς το ελέγχει. Μόλις τελειώσει ο έλεγχος του κάθε αρχείου ενημερώνεται και το αρχείο **paths.json**. Σε περίπτωση που βρεθεί ένα κακόβουλο αρχείο καλείται η διεργασία **get_stats_zip** η οποία δημιουργεί ένα .txt αρχείο με πληροφορίες του προγράμματος (όνομα, διαδρομή md5, sha256, μέγεθος, ημερομηνία δημιουργίας και τροποποίησης κ.α.). Η εφαρμογή διαγράφει το κακόβουλο αρχείο από την θέση που βρίσκεται και μαζί με τις πληροφορίες που άντλησε συμπιέζονται και αποθηκεύονται στον φάκελο **bas_files** που δημιουργήθηκε κατά την πρώτη εκκίνηση του προγράμματος. Ταυτόχρονα ενημερώνεται το file.log για τις παραπάνω ενέργειες.

4.2 Έλεγχος Μνήμης

Μέσω της τρίτου thread που δημιουργείται, ελέγχεται περιοδικά η μνήμη για κακόβουλη δραστηριότητα. Σε αυτό το thread καλείται η διεργασία yaramem η οποία φορτώνει στην μνήμη το αρχείο **saved_yara_file.yara** προκειμένου να προβεί στους ελέγχους. Επιπλέον χρησιμοποιεί το αρχείο **pids.json** όπου αποθηκεύονται το Process ID (pid) καθώς και την ημερομηνία της δημιουργίας του. Η εφαρμογή φορτώνει το αρχείο αυτό ώστε να συγκρίνει τα εν λόγο δεδομένα με αυτά που αντλεί από τους ελέγχους της μνήμης.

Περιοδικά μέσω της **psutil.process_iter()** αντλούνται όλες οι διεργασίες που τρέχει το σύστημα και καλώντας την **as_dict** για κάθε επιστρεφόμενη διεργασία μπορούμε να δούμε ως λεξικό τις πληροφορίες ενδιαφέροντος μας. Στην προκειμένη περίπτωση λαμβάνουμε τις εξής πληροφορίες **['pid', 'name', 'exe', 'cmdline', 'create_time','ppid']**. Η εφαρμογή ελέγχει αν υπάρχει ο συνδυασμός pid και την ώρα δημιουργίας στην λίστα που αντλήθηκε από το αρχείο **pids.json**. Αν υπάρχει η εφαρμογή προσπερνάει τον έλεγχο αλλιώς ελέγχει την διεργασία. Ο συνδυασμός pid και χρόνου δημιουργίας υφίσταται ώστε εάν μία διεργασία σταματήσει να εκτελείται να υπάρχει πιθανότητα το pid να το εκχωρηθεί σε μια νέα διεργασία.

Στην αρχή του προγράμματος το αρχείο pids.json είναι άδειο άρα κατά την έναρξη της εφαρμογής όλες οι διεργασίες που τρέχουν στο σύστημα θα ελεγχθούν. Σημαντικό είναι όχι μόνο να σταματήσει η διεργασία που τρέχει τον κακόβουλο πρόγραμμα αλλά πρέπει να σταματάει και η αρχική διεργασία (αν υπάρχει) που την δημιούργησε. Η συγκεκριμένη ονομάζεται «πατέρας» και το pid του βρίσκεται στο κλειδί του λεξικού με όνομα «ppid». Μέσω του module os αποστέλλουμε ένα σήμα τερματισμού τόσο στην ίδιας της διεργασίας όσο και στον πατέρα που την δημιούργησε. Τέλος ενημερώνεται το **memory.log** αρχείο καθώς και το **pids.json**.

4.3 Έλεγχος Δικτυακών Συνδέσεων

Το τέταρτο thread που δημιουργείτε ελέγχει τις δικτυακές συνδέσεις του υπολογιστή προκειμένου να αποφευχθούν συνδέσεις σε κακόβουλες IP. Η εφαρμογή αποθηκεύει τοπικά στον χρήστη κακόβουλες IP μαζί με πληροφορίες αυτών που αντλήθηκαν από το MISP. Η μορφή τους είναι **100.100.100.12,new_event - UUID: 5bff15c3-d1ec-4746-b8bb-087cc0a8010c** όπου αναγράφεται η κακόβουλη IP το όνομα του Event και το universally unique identifier UUID.

Για τον έλεγχο καλείται η συνάρτηση yaramem() η οποία περιοδικά φορτώνει στην μνήμη τις IP από το αρχείο **misp-c2-iocs.txt**. Στην συνέχεια η βιβλιοθήκη psutil μέσω της εντολής **get_connections(*kind='all'*)** επιστρέφει όλα τα πρωτόκολλα των συνδέσεων (UDP TCP κτλ) τα οποία είναι ανοιχτά καθώς και τις διεργασίες που προβαίνουν στις εν λόγο συνδέσεις.

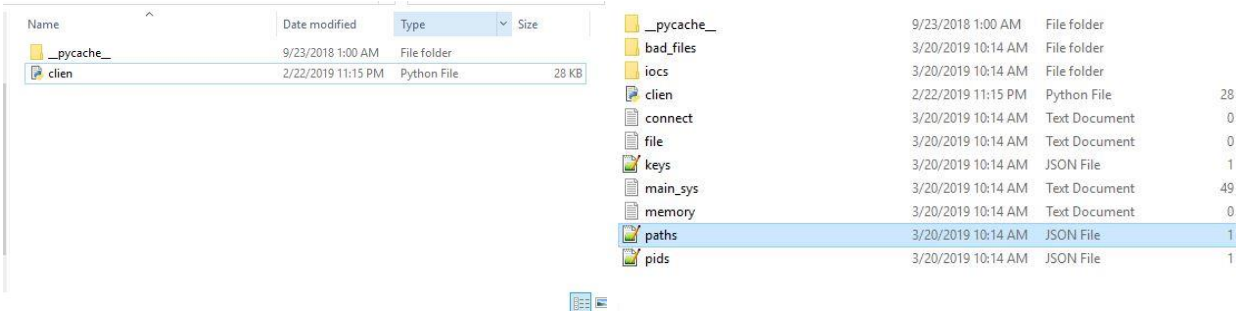
Η μορφή τους είναι μία λίστα από Τουπλες (tuples). Για να μπορέσουμε να διαχωρίσουμε τις απομακρυσμένες συνδέσεις μπορούμε να ανατρέξουμε στην μεταβλητή **raddr** (remote address) της εντολής **get_connections(*kind='all'*).** Η εφαρμογή αναζητά αν η IP που αντλήθηκε από την **raddr** βρίσκεται μέσα στην λίστα με τις κακόβουλες IP. Αν βρεθεί τερματίζουμε την διεργασία που προβαίνει στην σύνδεση στέλνοντας σήμα τερματισμού μέσω του module os. Το γεγονός αυτό επιτυγχάνεται με την εντολή **os.kill(p_to_kill.pid,signal.SIGTERM**) τόσο για την ίδια εφαρμογή όσο και για τον «πατέρα» που την δημιούργησε. Η εφαρμογή ενημερώνει το αρχείο **connect.log** προκειμένου ο χρήστης να έχει μια πλήρη ενημέρωση.

Προκειμένου η εφαρμογή να ελέγχει κάθε φορά όλες τις διεργασίες και τα εκτελέσιμα αρχεία με τους νέους κανόνες που αντλεί από το MISP, μόλις ολοκληρωθεί η λήψη αυτών

διαγράφει όλες τις αποθηκευμένες πληροφορίες στα αρχεία **paths.json** και **pids.json.**

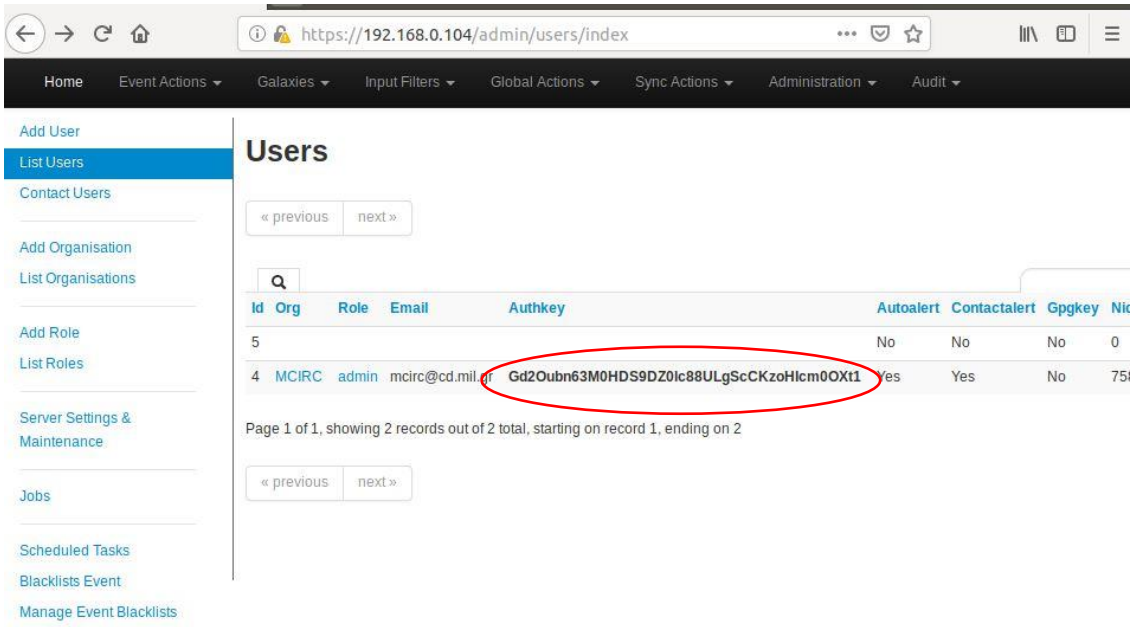
5. Επίδειξη – Εκτέλεση εφαρμογής

Η εφαρμογή μπορεί να τρέξει σε οποιαδήποτε θέση ένας χρήστης επιθυμεί. Κατά την πρώτη εκτέλεση δημιουργούνται τα απαραίτητα αρχεία και φάκελοί για την λειτουργία της.



Εικόνα 1: Δημιουργία απαραίτητων αρχείων και φακέλων

Προκειμένου να συνδεθούμε στο MISP πρέπει να εισάγουμε τα απαραίτητα διαπιστευτήρια στο αρχείο keys.json. Η μορφή του είναι **{"id": "", "url": "", "cert": "False", "search_from": "2000-1-1", "Events": []}** και στο πεδίο id βάζουμε το activation key και στο πεδίο url το url του MISP. Στο MISP οδηγούμαστε στην καρτέλα **List Users** από όπου παίρνουμε το activation key του χρήστη για να το χρησιμοποιήσουμε στην εφαρμογή που υλοποιήθηκε



Εικόνα 2: Άντληση του Authentication Key

Η τελική μορφή του αρχείου keys.json θα είναι η εξής **{"id": "Gd2Oubn63M0HDS9DZ0Ic88ULgScCKzoHIcm0OXt1", "url": "https://192.168.0.104", "cert": "False", "search_from": "2000-1-1", "Events": []}**

Η εφαρμογή είναι έτοιμη να ξεκινήσει. Κατά την εκτέλεση στο main_sys.log αρχείο καταγράφηκαν τι είδους και πόσα IOC αντλήθηκαν από το MISP.

```
2019-03-20 10:27:32,736 : 8 IOCs written to file C:\Users\john\Desktop\workSpace\thesis/iocs\misp-c2-iocs.txt
2019-03-20 10:27:32,738 : 1 IOCs written to file C:\Users\john\Desktop\workSpace\thesis/iocs\misp-filename-iocs.txt
2019-03-20 10:27:32,741 : 1 IOCs written to file C:\Users\john\Desktop\workSpace\thesis/iocs\misp-hash-iocs.txt
2019-03-20 10:27:32,761 : 7 YARA rules written to directory C:\Users\john\Desktop\workSpace\thesis/iocs/yara
```

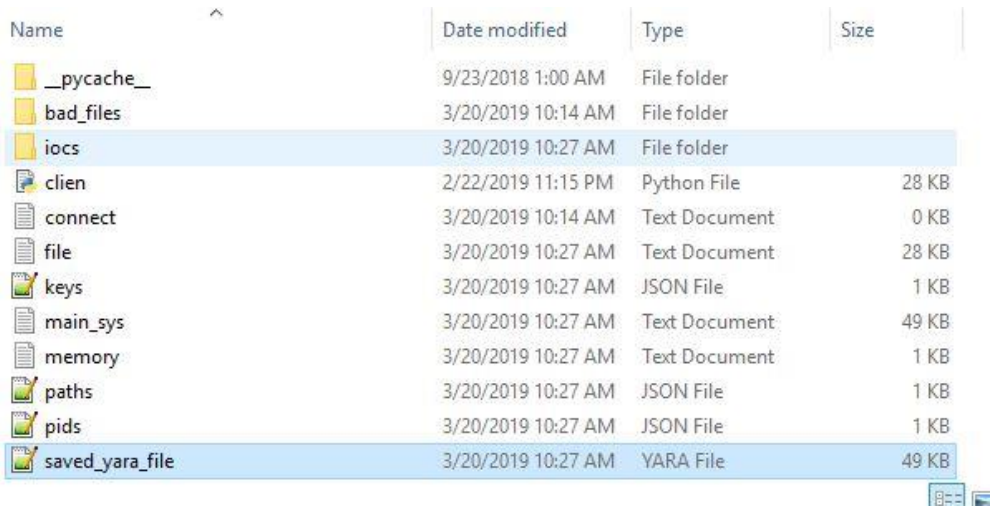
Εικόνα 3: Log αρχείο καταγραφής των IOC που αντλήθηκαν

Επίσης στο αρχείο keys.json καταγράφηκαν τα γεγονότα που προσπελάστηκαν καθώς και τα χαρακτηριστικά τους. Κατά την πρώτη εκτέλεση του προγράμματος, ζητείται να γίνει έλεγχος από την ημερομηνία 2000-1-1 εως και σήμερα. Αφού ολοκληρωθεί η πρώτη αναζήτηση, παρατηρούμε πως στο αρχείο καταγράφτηκε η ημερομηνία αναζήτησης με ημερομηνία δέκα μέρες νωρίτερα (10 μέρες πριν την σημερινή ημερομηνία)

Αποτελέσματα του αρχείου keys.json παρουσιάζονται παρακάτω:

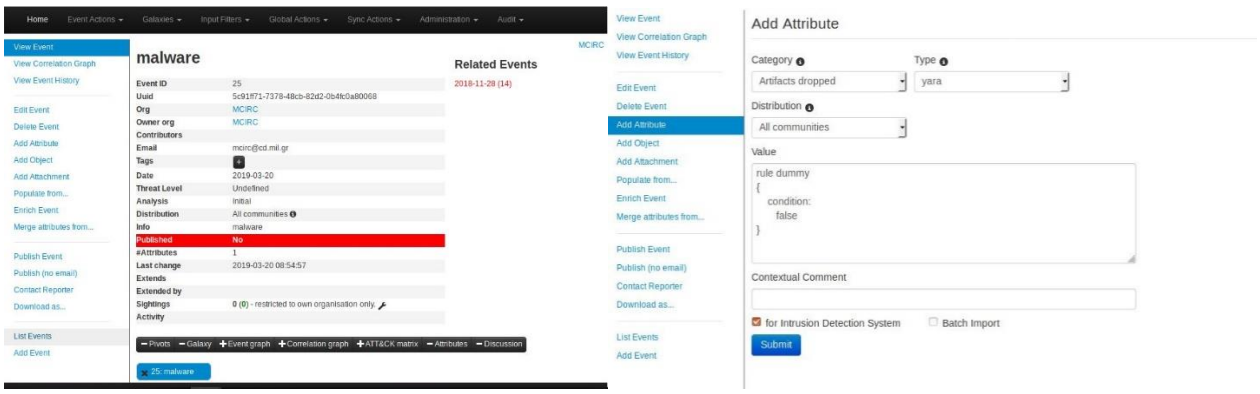
{"id": "Gd2Oubn63M0HDS9DZ0Ic88ULgScCKzoHIcm0OXt1", "url": "https://192.168.0.104", "cert": "False", "search_from": "2019-02-17", "Events": [{"id": "8", "attribs": ["9", "11"]}, {"id": "9", "attribs": ["10"]}, {"id": "10", "attribs": []}, {"id": "11", "attribs": ["18", "19", "20", "21", "22", "23", "24"]}, {"id": "12", "attribs": ["25"]}, {"id": "13", "attribs": ["26"]}, {"id": "14", "attribs": ["27", "28", "29", "32"]}, {"id": "15", "attribs": ["30", "31"]}, {"id": "16", "attribs": ["33"]}, {"id": "17", "attribs": ["34", "35"]}, {"id": "18", "attribs": ["36"]}, {"id": "19", "attribs": ["37"]}, {"id": "20", "attribs": ["38"]}, {"id": "22", "attribs": ["39"]}, {"id": "23", "attribs": ["40"]}, {"id": "24", "attribs": ["41"]}]}

Επιπροσθέτως, αν ανατρέξουμε στους αντίστοιχους φακέλους θα δούμε ότι έχουν αντληθεί οι αντίστοιχοι ενδείκτες παραβίασης. Όλοι οι κανόνες Yara έχουν αποθηκευτεί και συνενωθεί σε ένα αρχείο με το όνομα Saved_yara_file.yara



Εικόνα 4: Δημιουργία Συνολικού αρχείου YARA

Η εφαρμογή ξεκινάει τα 4 threads. Εισάγουμε πλέον ένα νέο γεγονός στο MISP με ένα νέο Yara rule (όπου στην προκειμένη περίπτωση δεν κάνει τίποτα) και παρατηρούμε ότι αντλήθηκε από την εφαρμογή



Εικόνα 5: Εισαγωγή ενός νέου Event

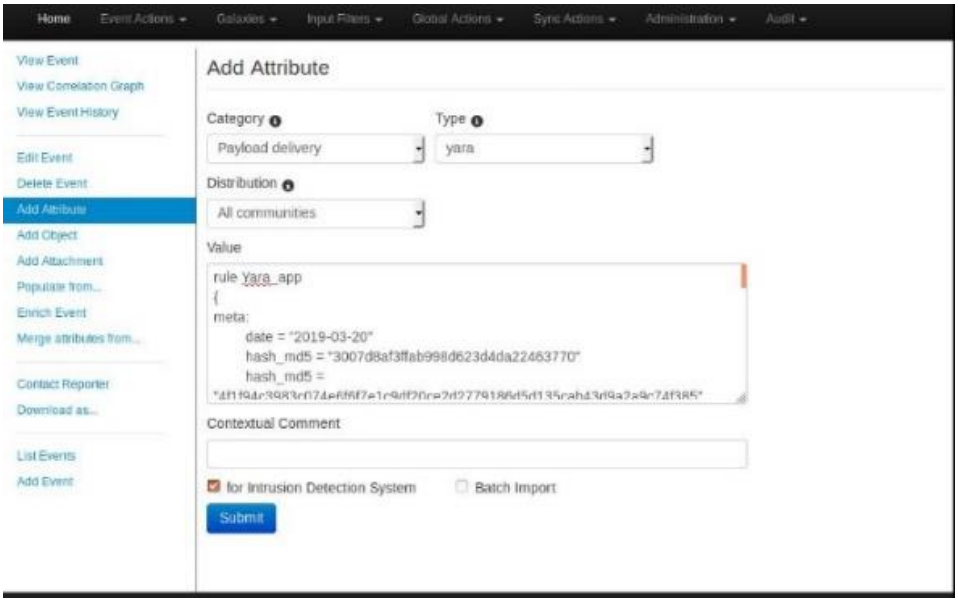
Το γεγονός ότι πραγματοποιήθηκε η άντληση του Yara rule από την εφαρμογή μπορούμε να το παρατηρήσουμε και από τα logs αλλά και από τον φάκελο που κατέβηκε ο εν λόγο κανόνας yara

```
2019-03-20 10:27:32,736 : 8 IOCs written to file C:\Users\john\Desktop\workSpace\thesis/iocs\misp-c2-iocs.txt
2019-03-20 10:27:32,738 : 1 IOCs written to file C:\Users\john\Desktop\workSpace\thesis/iocs\misp-filename-iocs.txt
2019-03-20 10:27:32,741 : 1 IOCs written to file C:\Users\john\Desktop\workSpace\thesis/iocs\misp-hash-iocs.txt
2019-03-20 10:27:32,761 : 7 YARA rules written to directory C:\Users\john\Desktop\workSpace\thesis/iocs/yara
2019-03-20 10:55:00,063 : 1 YARA rules written to directory C:\Users\john\Desktop\workSpace\thesis/iocs/yara
```

Εικόνα 6: Ένδειξη άντλησης νέου IOC

Επίσης στο αρχείο keys.json καταγράφηκε το νέο γεγονός και το χαρακτηριστικό του που προσπελάστηκε **{"id": "25", "attribs": ["42"]}**.

Προκειμένου να έχουμε ένα ρεαλιστικό σενάριο ,δημιουργήσαμε ένα απλό αρχείο το οποίο εμφανίζει διαδοχικά αριθμούς σε ένα τερματικό και θα το ορίσουμε ως κακόβουλο. Αφού το αναλύσαμε και υλοποιήσαμε έναν νέο κανόνα yara το εισαγάγαμε στο γεγονός με Id 25



Εικόνα 7: Εισαγωγή ενός νέου Attribute

```
2019-03-20 10:27:32,736 : 8 IOCs written to file C:\Users\john\Desktop\workSpace\thesis/iocs\misp-c2-iocs.txt
2019-03-20 10:27:32,738 : 1 IOCs written to file C:\Users\john\Desktop\workSpace\thesis/iocs\misp-filename-iocs.txt
2019-03-20 10:27:32,741 : 1 IOCs written to file C:\Users\john\Desktop\workSpace\thesis/iocs\misp-hash-iocs.txt
2019-03-20 10:27:32,761 : 7 YARA rules written to directory C:\Users\john\Desktop\workSpace\thesis/iocs/yara
2019-03-20 10:55:00,063 : 1 YARA rules written to directory C:\Users\john\Desktop\workSpace\thesis/iocs/yara
2019-03-20 11:15:45,685 : 1 YARA rules written to directory C:\Users\john\Desktop\workSpace\thesis/iocs/yara
```

Εικόνα 8: Άντληση του εν λόγο Attribute

Παρατηρούμε ότι αντλήθηκε στις καταγραφές του Log αρχείο και του αρχείου keys.json **{"id": "25", "attribs": ["42", "43"]}]**

Πλέον το κακόβουλο πρόγραμμα που δημιουργήσουμε, μόλις θα ξεκινήσει δραστηριότητα η εφαρμογή μας το τερματίζει. Παράλληλα, το διαγράφει από την θέση εκτέλεσης και το αποθηκεύει στο φάκελο bad_files σε μορφή .zip , ενώ συμπεριλαμβάνει και ένα σχετικό report.

Η παραπάνω διαδικασία, παρατηρείται στο log αρχείο όπου καταγράφονται τα αποτελέσματα του ελέγχου της μνήμης.

```
2019-03-20 11:13:39,756 : Found malicious application name : app.exe path : C:\Users\john\Desktop\app.exe PID 1600
```

Εικόνα 9: Καταγραφή ανεύρεσης κακόβουλου αρχείου

Σε περίπτωση που το πρόγραμμα έχει εκτελεστεί σε παρελθόντα χρόνο από τις διαδρομές που αντλούνται βρίσκεται και γίνεται η αντίστοιχη διαδικασία. Η συγκεκριμένη διαδικασία, παρατηρείται στο log αρχείο όπου καταγράφονται τα ευρήματα από τον έλεγχο των διαδρομών.

Επιπροσθέτως, αφού η εφαρμογή είναι υπεύθυνη για την αναζήτηση και τη ανεύρεση συνδέσεων που προσπαθεί ο υπολογιστής να πετύχει στο διαδίκτυο, υλοποιήσαμε ένα απλό script που προσπαθεί να δημιουργήσει μία σύνδεση στην IP «188.165.79.246». Η εν λόγο IP έχει καταχωρηθεί ως κακόβουλη, όπως αυτό φαίνεται στο αρχείο **misp-c2-iocs.txt** με τις κακόβουλες IP. Το script άνοιξε στιγμιαία, τερματίστηκε και ακολουθήθηκε η αντίστοιχη διαδικασία.

6. Επίλογος - Συμπεράσματα

Αναφορικά με την υλοποίηση της εφαρμογής αντιμετωπίστηκαν προβλήματα όπως η συνδεσιμότητα της εφαρμογής με το MISP και χρήση των πόρων του συστήματος.

Η πρώτη προσέγγιση ήταν η δημιουργία ένας client και ένας Server προκειμένου ο Server να αποστέλλει συγκεκριμένες και κρυπτογραφημένες πληροφορίες στον client. Χρησιμοποιήθηκε η δημιουργία κλειδιού με την μέθοδο Diffie Hellman και χρήση αυτού για την κρυπτογράφηση των πληροφοριών με την κρυπτογραφική μέθοδο AES. Τα προβλήματα που αντιμετωπίστηκαν ήταν ότι η δημιουργία κλειδιού με την μέθοδο Diffie Hellman υπόκειτο σε ευπαθείς όπως Man In The Middle και Pass The Hash. Προκειμένου να εξασφαλιστεί η αυθεντικότητα και η ακεραιότητα της πληροφορίας χρησιμοποιήθηκε η μέθοδος hash-based message authentication code (hmac) με επιπλέον πληροφορία την ώρα σύνδεσης το χρήστη στον server. Η ίδια ώρα χρησιμοποιήθηκε επίσης για την κρυπτογράφηση του μηνύματος. Έτσι, για την αποκρυπτογράφηση θα έπρεπε και ο server και ο client να γνωρίζουν την ώρα που υλοποιήθηκε η σύνδεση για να μπορέσουν να έχουν πρόσβαση στην πληροφορία. Παρ' όλα αυτά, και αυτή η μέθοδος είχε ευπάθειες και έτσι απορρίφθηκε ενώ σημειώθηκαν πολλές καθυστερήσεις και σφάλματα κατά την εκτέλεση της εφαρμογής.

Στη συνέχεια, για την εφαρμογή του μοντέλου επιχειρήθηκα η μέθοδος αυθεντικοποίησης με RSA και SSL sockets η οποία φάνηκε να είναι εξαιρετικά αποτελεσματική και ασφαλής. Ωστόσο, απορρίφθηκε διότι η γλώσσα προγραμματισμού Python είχε δυσκολίες για την υλοποίηση πολλών thread όπου το καθένα από αυτά προοριζόταν για μια ξεχωριστή σύνδεση ενός client.

Τέλος, υλοποιήθηκε το πρόγραμμα όπως παρουσιάστηκε παραπάνω, όπου ο κάθε χρήστης με το authentication key συνδεόταν κατευθείαν στο MISP και αντλούσε τις επιθυμητές πληροφορίες. Η σύνδεση ελέγχθηκε με την εφαρμογή Wireshark και παρατηρήθηκε ότι δημιουργεί ένα κανάλι TLS1.2 προκειμένου να αποστείλει τους κανόνες YARA και τις κακόβουλες IP. Η σύνδεση TLS θεωρείτε ασφαλή και επιθέσεις στο δίκτυο (όπως μια επίθεση τύπου man in the Middle) δύναται να αποτύχουν. Τόσο ο τρόπος αναζήτησης όσο και ο τρόπος επεξεργασίας υλοποιήθηκαν έτσι, ώστε να χρησιμοποιούνται οι ελάχιστοι πόροι. Συγκεκριμένα, η κατανάλωση για το κάθε thread ήταν περίπου 13 MB RAM ενώ για το σύνολο της εφαρμογής, περίπου 100MB RAM. Δεν υλοποιήθηκαν

μηχανισμοί whitelisting προκειμένου τυχόν καλόβουλές εφαρμογές - που ενδέχεται να ενεργοποιήσουν κάποιον κανόνα – να αγνοούνται από την εφαρμογή.

Σε μελλοντικές επεκτάσεις θα υλοποιηθεί ένα GUI για την αλληλεπίδραση του χρήστη με την εφαρμογή, καθώς και μηχανισμοί whitelisting με σκοπό την αποφυγή λανθασμένου ελέγχου στα εκτελέσιμα αρχεία.