**Τμήμα Ψηφιακών Συστημάτων**


# Διπλωματική Εργασία

Χρήση εργαλείων αποσφαλμάτωσης εφαρμογών web 2.0 με σκοπό την παράκαμψη μέτρων προστασίας περιεχομένου προστατευμένης πνευματικής ιδιοκτησίας.

Βελισσάρης Γεώργιος

Δεκέμβριος 2015

## Επιβλέπων Καθηγητής

Κάτσικας Σωκράτης, Καθηγητής, Πανεπιστήμιο Πειραιώς

## Εξεταστική Επιτροπή

Κάτσικας Σωκράτης, Καθηγητής, Πανεπιστήμιο Πειραιώς
Λαμπρινουδάκης Κωνσταντίνος, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Πειραιώς
Ξενάκης Χρήστος, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Πειραιώς

**Περίληψη**

Τα τελευταία χρόνα, η πρόοδος της τεχνολογίας σε επίπεδο υπολογιστών και παγκόσμιας δικτύωσης έχει επιτρέψει στις διάφορες ηλεκτρονικές συσκευές που χρησιμοποιούν οι άνθρωποι να διειδύσει σε μεγάλο βαθμό μέσα στις ζωές τους. Ένα από τα πιο καίρια σημεία διείσδυσης είναι η διασκέδαση και η ψυχαγωγία. Όπως είναι αναμενόμενο λοιπόν, πολυμεσικό υλικό που παλαιότερα επενδύονταν τεράστια ποσά για να αποτραπεί η αντιγραφή του, πλέον είναι διαθέσιμο σε οποιοδήποτε χρήστη ανα τον κόσμο, για αναπαραγωγή μέσω συνεχούς ροής (streaming) από το διαδίκτυο. Υπάρχει όμως μία παράμετρος η οποία έμεινε σταθερη, και αυτή είναι η ανάγκη προστασίας της πνευματικής ιδιοκτησίας. Η οποία, απο τη στιγμή που τα μέσα παραδίδονται στις συσκευές των χρηστών προς αναπαραγωγή, είναι ευπαθής σε ένα μεγάλο εύρος επιθέσεων με σκοπό την παράκαμψη των μηχανισμών ασφαλείας και την ελεύθερη διαμοίραση τους.

Στα πλαίσια αυτής της διπλωματικής εργασίας παρουσιάζονται μεθοδολογίες που μπορούν να ακολουθηθούν από κακόβουλους χρήστες για την απόκτηση αντιγράφων προστατευμένων μέσων που διαμοιράζονται μέσω streaming με σκοπό την σφαιρικότερη κατανόηση των ευπαθειών και της φύσης του προβλήματος. Τέλος παρουσιάζονται πιθανά αντίμετρα.

**Περιεχόμενα**

# Μέρος Α

# Θεωρητική Προσέγγιση

# 1. Εισαγωγή

## 1.1. Εισαγωγή

Τα τελευταία χρόνα, η πρόοδος της τεχνολογίας σε επίπεδο υπολογιστών και παγκόσμιας δικτύωσης έχει επιτρέψει στις διάφορες ηλεκτρονικές συσκευές που χρησιμοποιούν οι άνθρωποι να διειδύσουν σε μεγάλο βαθμό μέσα στις ζωές τους. Ένα από τα πιο καίρια σημεία διείσδυσης είναι η διασκέδαση και η ψυχαγωγία. Όπως είναι αναμενόμενο λοιπόν, πολυμεσικό υλικό που παλαιότερα επενδύονταν τεράστια ποσά για να αποτραπεί η αντιγραφή του, πλέον είναι διαθέσιμο σε οποιοδήποτε χρήστη ανα τον κόσμο, για αναπαραγωγή μέσω συνεχούς ροής (streaming) από το διαδίκτυο.

## 1.2. Παραδοσιακοί τρόποι διαμοιρασμού έργων πνευματικής ιδιοκτησίας

Έως και την περασμένη δεκαετία, πριν δηλαδή την εξάπλωση των φορητών συσκευών και τη ραγδαία αύξηση της προσβασιμότητας στο ασύρματο διαδίκτυο, τα έργα πολυμεσικής φύσης διαμοιράζονταν με χρήση οπτικών (ή και μαγνητικών πιο παλιά) μέσων. Για να αποκτήσει κάποιος ενδιαφερόμενος πρόσβαση σε ένα τέτοιο προιόν, θα έπρεπε να καταβάλλει το αντίτιμο της αγοράς του, ποσοστά από το οποίο θα αποδίδονταν στους κατόχους της πνευματικής ιδιοκτησίας.

Για να επιτευχθεί η αποτροπή μη-εξουσιοδοτημένης αντιγραφής των προϊόντων αυτών, σε μερικές περιπτώσεις εισάγονταν διάφορες τεχνικές προστασίας σε ηλεκτρονικό επίπεδο.

Ακόμα και στην περίπτωση όμως που μπορούσε να επιτευχθεί η δημιουργία πιστού αντιγράφου, παρακάμπτοντας τους μηχανισμούς προστασίας αυτούς, το κόστος που απαιτείτο για την παράνομη διακίνηση μεγάλου δείγματος τέτοιων έργων ήταν αρκετά μεγάλο, καθώς θα έπρεπε να υπάρξει πρόσβαση στο αυθεντικό πρωτότυπο (και άρα αγορά του) από τους κακόβουλους παράγοντες.

## 1.3. Σύγχρονοι τρόποι παροχής πρόσβασης σε έργα πνευματικής ιδιοκτησίας

Τα τελευταία χρόνια το μοντέλο παροχής πρόσβασης σε έργα πνευματικής ιδιοκτησίας έχει αλλάξει. Υπάρχουν υπηρεσίες στο διαδίκτυο οι οποίες παρέχουν απεριόριστο εύρος από τέτοια έργα κάθε είδους (ταινίες, μουσική, τηλεοπτικές σειρές κλπ) και η χρέωση στο χρήστη δεν γίνεται με τη “μονάδα” που αναπαράγει. Μπορεί να προσπελάσει όλα τα προϊόντα που προσφέρει η υπηρεσία, το ένα μετά το άλλο, αρκεί να πληρώνει μία μηνιαία συνδρομή, συνήθως μικρότερη από το κόστος ένος και μόνο αντίστοιχου έργου σε μορφή οπτικού μέσου. Το μόνο μειονέκτημα για το χρήστη είναι ότι η πρόσβαση του σε όλα αυτά τα

μέσα σταματάει μόλις λήξει και η συνδρομή. Συνολικά όμως, είναι μία αρκετά συμφέρουσα προσφορά.

Η προστασία της πνευματικης ιδιοκτησίας στην ουσία επιτυγχάνεται με την στέρηση απο το χρήστη της δυνατότητας αποθήκευσης του μεσου σε μορφή χρησιμοποιήσιμη πέραν της συνδρομής.

Σε περίπτωση όμως που ο χρήστης καταφέρει να “σπάσει” την αλυσίδα των αντιμέτρων αυτών και να ανακαλύψει ένα τρόπο να αποθηκεύει σε δικό του χώρο τα μέσα αυτά, το δυνητικό κόστος είναι πολύ μεγαλύτερο από αυτό των παραδοσιακών πολυμέσων. Γιατί με την πληρωμή της ελάχιστης περιόδου συνδρομής, μπορεί - με τα κατάλληλα εργαλεία - να ξεκινήσει μία ακολουθιακή προσπέλαση όλης της βάσης των μέσων με ταυτόχρονη λήψη και αποθήκευση και εν συνεχεία να προχωρήσει στο διαμοιρασμό τους.

Γίνεται αμέσως αντιληπτό ότι ο κίνδυνος αυτός είναι τελικά αδιαμφισβήτητα μη αμελητέος και το μέγεθος του εξαιρετικά επικίνδυνο.

## 1.4. Τι είναι Web 2.0

Ο όρος Web 2.0 (Ιστός 2.0), χρησιμοποιείται για να περιγράψει τη νέα γενιά του Παγκόσμιου Ιστού η οποία βασίζεται στην όλο και μεγαλύτερη δυνατότητα των χρηστών του Διαδικτύου να μοιράζονται πληροφορίες και να συνεργάζονται online. Αυτή η νέα γενιά είναι μια δυναμική διαδικτυακή πλατφόρμα στην οποία μπορούν να αλληλεπιδρούν χρήστες χωρίς εξειδικευμένες γνώσεις σε θέματα υπολογιστών και δικτύων.[1]

Στα πλαίσια της εργασίας, όταν αναφέρεται ο όρος Web 2.0, στην ουσία αναφερόμαστε στο μοντέλο σύγχρονων διαδικτυακών υπηρεσιών οι οποίες όχι μονο δεν προσφέρουν στατικό περιεχόμενο, αλλά υψηλά δυναμικό με μεγάλο ποσοστό μεταφοράς της επιχειρησιακής λογικής (business logic) στο φυλλομετρητή (browser) του χρήστη μέσω της εκτέλεσης εντολών JavaScript.

Από τεχνολογικής και τεχνικής απόψεως, μόνο μια Web 2.0 διαδικτυακή υπηρεσία μπορεί να προσφέρει πρόσβαση σε έργα πνευματικής ιδιοκτησίας, κατά τη βούληση του χρήστη.

## 1.5. Υπηρεσίες παροχής πολυμέσων μέσω streaming

Οι υπηρεσίες παροχής πολυμέσων μέσω streaming που είναι ευρέως γνωστές τη στιγμή της συγγραφής της εργασίας είναι, χωρισμένες ανά κατηγορία μέσων:

1. Μουσική
    1. Spotify[2]
    2. Google Play Music[3]
    3. Deezer[4]
    4. Rdio[5]
    5. Apple Music[6]
    6. Amazon Prime Music[7]
2. Ταινίες

    1. Netflix[8]
    2. Hulu[9]
3. Τηλεοπτικές Σειρές
    1. Netflix
    2. Hulu

Όλες οι υπηρεσίες αυτές χρησιμοποιούν το συνδρομητικό μοντέλο χρηστών που περιγράφηκε στην ενότητα 1.3.

# 2. Θεμελίωση της Επίθεσης

## 2.1. Βασική φιλοσοφία της επίθεσης

Όλες οι υπηρεσίες αναπαραγωγής πολυμέσων μέσω streaming βασιζονται στην εξής φιλοσοφία: Ο χρήστης αυθεντικοποιείται στην υπηρεσία μέσω των συνθηματικών του, πλοηγείται μέσα στην διαδικτυακή εφαρμογή μέχρι να βρει το μέσο που επιθυμεί και ξεκινάει την αναπαραγωγή. Από τεχνικής απόψεως η αναπαραγωγή γίνεται με αποστολή των πολυμεσικών δεδομένων, κρυπτογραφημένων ή μη, στον υπολογιστή του χρήστη, όπου ο φυλλομετρητής του αναλαμβάνει την αναπαραγωγή. Για να μπορέσει να επιτευχθεί η αναπαραγωγή, ακόμα και στην περίπτωση που τα δεδομένα αποστέλλονται κρυπτογραφημένα από την υπηρεσία, είναι απαραίτητο να υπάρξει το μέσο διαθέσιμο στη μνήμη του υπολογιστή του τελικού χρήστη στην αυθεντική του μορφή.

Συνεπώς, ανάλογα με τη φύση της υπηρεσίας, μπορεί να είναι δυνατή η απόκτηση πρόσβασης στους ενιαίους εντοπιστές πόρων (URLs) των πολυμέσων, ή η παράλληλη αποθήκευση των πολυμέσων σε ένα ξεχωριστό σημείο όπως γίνεται η λήψη τους απο την επίσημη εφαρμογή της υπηρεσίας και, σε περίπτωση κρυπτογραφημένων δεδομένων, η αποκρυπτογράφησή τους σε δεύτερο χρόνο. Το κλειδί κρυπτογράφησης μπορεί να αποκαλυφθεί με αντίστροφη μηχανική στις εντολές JavaScript που εκτελούνται στις ιστοσελίδες της υπηρεσίας, στο browser του χρήστη.

## 2.2 Πιθανά κίνητρα για τον κακόβουλο χρήστη

Το ενδεχόμενο ένας κακόβουλος χρήστης να καταφέρει να αποκτήσει στην κατοχή του πολυμεσικά αρχεία με μη-εξουσιοδοτημένο τρόπο, ενέχει πολλά εν δυνάμει κίνητρα. Το πιο απλό και αθώο είναι η ηθική ικανοποίηση για την παραβίαση. Ένα άλλο κίνητρο μπορεί να είναι η απόκτηση πρόσβασης σε συγκεριμένες πολυμεσικές συλλογές “δωρεάν” και εφάπαξ, καθώς αν γίνει λήψη των αρχείων πληρώνοντας την ελάχιστη περίοδο συνδρομής, μπορεί να έχει πρόσβαση σε αυτά εφόρου ζωής.

Ακόμα παραπέρα όμως, ένας κακόβουλος χρήστης μπορεί να διακινήσει τα πολυμεσικά αυτά αρχεία που θα έχει στην κατοχή του, καταφέρνοντας έτσι να επωφεληθεί και οικονομικά πουλώντας τα. Είναι λοιπόν φανερό ότι υπάρχουν μεγάλα και πολύ ισχυρά κίνητρα για έναν κακόβουλο χρήστη ώστε να παραβιάσει και να παρακάμψει το οικονομικό μοντέλο (Business Model) παροχής επί πληρωμή υπηρεσιών προστατευόμενου περιεχομένου.

## 2.3 Κόστη και κίνδυνοι που ελλοχεύουν για τις εταιρείες που βάλλονται.

Φυσικά, οποιαδήποτε “παράκαμψη” του επιχειρησιακού/εμπορικού μοντέλου των εταιρειών που προσφέρουν τις πολυμεσικές αυτές υπηρεσίες, επηρεάζει άμεσα την οικονομική ισορροπία την οποία η ίδια έχει προγραμματίσει.

Στην απλή περίπτωση που ελάχιστοι χρήστες καταφέρνουν να εκμεταλλευτούν μια ευπάθεια και να αποκτήσουν πρόσβαση στα πηγαία αρχεία των πολυμέσων και τα κρατούν αποκλειστικά και μόνο για δική τους χρήση, το κόστος και η ζημία είναι σχεδόν αμελητέα, καθώς αυτοί οι λίγοι χρήστες είναι ένα πολύ μικρό ποσοστό από όλη την αγορά παγκοσμίως.

Οι συνέπειες αυξάνονται, καθώς αυξάνεται και ο αριθμός των χρηστών που αποκτούν πρόσβαση σε κάποιο εργαλείο “εκμετάλλευσης” της ευπάθειας αυτής και το χρησιμοποιούν για να αποκτήσουν μόνιμη πρόσβαση στα μέσα που επιθυμούν, καταργώντας έτσι τις συνδρομές τους και άρα σταματώντας να συνεισφέρουν στα έσοδα της εταιρείας.

Το ανωτέρο σενάριο μπορεί να εξελιχθεί σε ένα σενάριο “εφιάλτη”, αν ορισμένοι χρήστες διατηρούν μόνιμες συνδρομές, παράλληλα “σκανάρουν” όλη τη βάση των πολυμέσων της υπηρεσίας και διατηρούν αντίγραφα από όλα τα αρχεία όσα βρίσκουν και τα προωθούν σε μορφή πειρατείας. Σε αυτή την περίπτωση, η ευπάθεια της εταιρείας που προσφέρει την πολυμεσική υπηρεσία στρέφεται ολοκληρωτικά εναντίον της, καθώς ενδυναμώνει και ισχυροποιεί την πειρατεία και παρέχει ένα εύκολο τρόπο προσφοράς μεγάλου εύρους μέσων στα πειρατικά κανάλια.

Η ενδυνάμωση της πειρατείας θα προκαλέσει μια αλυσιδωτή αντίδραση μείωσης πωλήσεων σε επίπεδο παγκόσμιας αγοράς των συγκεκριμένων μέσων κάτι το οποίο φυσικά θα σημαίνει και λιγότερα έσοδα για τους κατόχους των πνευματικών δικαιωμάτων (καλλιτέχνες και κυρίως εταιρείες παραγωγής). Επόμενο αυτού, θα είναι δικαστικές διενέξεις μεταξύ κατόχων των πνευματικών δικαιωμάτων και των εταιριών παροχής υπηρεσιών πολυμέσων που είναι ευπαθείς, κάτι το οποιο δύναται να καταλήξει σε ακόμα μεγαλύτερες οικονομικές ζημίες με αποζημιώσεις της τάξης των εκατομμυρίων, αναλογικά πάντα με το μέγεθος της παραβίασης που έχει προκληθεί.

# 3. Θεωρητικό υπόβαθρο

Για να κατανοηθεί καλύτερα η μεθοδολογία της επίθεσης σε τέτοιου είδους υπηρεσίες πρέπει να γίνει πρώτα μία ανάλυση των βασικών εννοιών και μηχανισμών πάνω στους οποίους δουλεύουν.

## 3.1 Τα πρωτόκολλα HTTP και HTTPS

Το Πρωτόκολλο Μεταφοράς Υπερκειμένου (HyperText Transfer Protocol, HTTP) αποτελεί το κύριο πρωτόκολλο που χρησιμοποιείται στους φυλλομετρητές του Παγκοσμίου Ιστού για να μεταφέρει δεδομένα ανάμεσα σε έναν διακομιστή (server) και έναν πελάτη (client).

Όλες οι ιστοσελίδες που επισκεπτόμαστε καθημερινά από τον παγκόσμιο ιστό χρησιμοποιούν αυτό το πρωτόκολλο για τη μεταφορά της πληροφορίας στον υπολογιστή του τελικού χρήστη.

Το Ασφαλές Πρωτόκολλο Μεταφοράς Υπερκειμένου (HyperText Transfer Protocol Secure, HTTPS) αποτελεί μία μετεξέλιξη του πρωτοκόλλου HTTP με την προσθήκη μηχανισμών ασφαλείας, οι οποίοι ως στόχο έχουν την κατα το δυνατό εγγύηση της Αξιοπιστίας και Εμπιστευτικότητας των υπό ανταλλαγή δεδομένων. Μία από τις συνέπειες υιοθέτησης του HTTPS από διαδικτυακές υπηρεσίες, η οποία θα μας απασχολήσει άμεσα για την έρευνά μας, είναι ότι οποιοσδήποτε μεσάζων ανάμεσα στον φυλλομετρητή-πελάτη και στον εξυπηρετητή της υπηρεσίας δεν είναι σε θέση να ανακαλύψει την πληροφορία που ανταλλάσσεται, σε όποιο στάδιο της μεταγωγής κι αν παρεμβληθεί.

## 3.2 Πυλώνες υλοποίησης του Web 2.0

Οι μοντέρνες διαδικτυακές εφαρμογές ιστού, αποσκοπώντας στο να προσφέρουν την καλύτερη και πιο ολοκληρωμένη εμπειρία στο χρήστη, έχουν ένα εξελιγμένο μοντέλο λειτουργίας. Αυτό το μοντέλο, βασίζεται στην μεταφορά μεγάλου ποσοστού της επιχειρησιακής λογικής και τον υπολογισμό ολόκληρης σχεδόν της παρουσιαστικής λογικής στο πρόγραμμα-πελάτη (browser του χρήστη στη συγκεκριμένη περίπτωση).

Αυτομάτως, αυτό το μοντέλο λειτουργίας χωρίζει της εφαρμογές αυτές σε δύο εξ'ίσου απαραίτητους αλλά διακριτούς πυλώνες υλοποίησης, τον πυλώνα του εξυπηρετητή και τον πυλώνα του browser.

Ιστορικά, πάντα ήταν απαραίτητη η ύπαρξη και των δύο πυλώνων, αλλά είναι τα τελευταία χρόνια που έχει αρχίσει και μετατίθεται όλο και μεγαλύτερο βάρος στον πυλώνα του browser. Συγκεκριμένα, ακολουθείται ένας διαχωρισμός όπου λειτουργίες που αφορούν τα δεδομένα, την αποθήκευση και τις παραμέτρους ασφάλειας εκτελούνται στην πλευρά του εξυπηρετητή, ενώ όσα αφορούν την παρουσίαση και το περιεχόμενο εκτελούνται στην πλευρά του browser.

Όπως γίνεται αμέσως αντιληπτό, ένας κακόβουλος χρήστης μπορεί να έχει πολύ πιο εύκολα πρόσβαση στον κώδικα που εκτελείται εκ μέρους του browser (ο οποίος "διατίθεται" προς κάθε υπολογιστή που συνδέεται σε μια ιστοσελίδα ανοιχτά και χωρίς περιορισμούς, για να μπορέσει να τον εκτελέσει) από ό,τι στον κώδικα του εξυπηρετητή.

Γι'αυτό το λόγο (μαζί με κάποιους άλλους λόγους όπως η οικονομία εύρους ζώνης, η βελτιστοποίηση της διερμηνείας του κ.α.) ο κώδικας JavaScript αποστέλεται στον browser σε μία ελαχιστοποιημένη (obfuscated) μορφή, όπου, έχοντας περάσει από ένα προεπεξεργαστή (preprocessor), εμφανίζεται με παραλλαγμένο συντακτικό και ονόματα μεταβλητών κατά τέτοιο τρόπο έτσι ώστε να είναι δυσνόητο για ανθρώπινους αναγνώστες, να προκαλεί όμως το ίδιο αποτέλεσμα η εκτέλεσή του από τη μηχανή.

## 3.3 Obfuscated Javascript

Η προεπεξεργασία στην οποία υποβάλλεται ο Javascript κώδικας μιας εφαρμογής για να γίνει obfuscated μπορεί να αναλυθεί στα εξής βήματα:

1. Concatenation. Όλα τα JS αρχεία που χρησιμοποιούνται σε μια εφαρμογή (ίσως χωρισμένα σε υποενότητες) ενώνονται το ένα μετά το άλλο σε ένα ενιαίο προσωρινό Javascript αρχείο.
2. Minification. Όλοι οι χαρακτήρες που προσδιορίζουν "κενό" (διάστημα, αλλαγή γραμμής, tab, κ.α.) αφαιρούνται και αντικαθιστούνται από ένα σκέτο κενό και επίσης γίνεται αντικατάσταση σε οποιεσδήποτε εντολές μπορούν να αντικατασταθούν με συντομότερες. Έχει υπολογιστεί ότι σε αυτό το στάδιο το συνολικό μέγεθος του ενιαίου αρχείου αυτού μειώνεται κατά 15-25%.
3. Obfuscation. Υπάρχουν διάφοροι μετασχηματισμοί που γίνονται σε αυτό το βήμα, και διαφέρουν από obfuscator σε obfuscator καθώς και εξαρτώνται από τις ρυθμίσεις του χρήστη-προγραμματιστή κάθε φορά. Οι πιο τυπικές διεργασίες είναι η αντικατάσταση των ονομάτων μεταβλητών με "τυχαίες" συμβολοσειρές (συνήθως ακολουθιακά όλα τα γράμματα του αλφαβήτου), και η παραλλαγή μεγάλων ακολουθιών εντολών σε πιο πολύπλοκα switch statements.

Για καλύτερη κατανόηση της έννοιας του obfuscated Javascript κώδικα, παρατίθεται παράδειγμα της διαδικασίας βηματικά:

Ας υποθέσουμε ότι η εφαρμογή μας αποτελείται από 6 αρχεία (στις μεγάλες εμπορικές εφαρμογές μπορεί να είναι και εκατονταπλάσιος ο αριθμός αυτός), τα εξής:

```
// constants.js

var userRolesWebService = '/users/getRole/';
```

```
// helpers.js

function httpGet(url) {
  return $.ajax(url);
}
```

```
// roles.js

var ROLES = {
  ADMIN: 1,
  SIMPLE_MEMBER: 2
};

function isAdmin(roleId) {
  return roleId === ROLES.ADMIN;
}
```

```
// userStore.js

function getUserRole(userId) {
  return httpGet(userRolesWebService + userId);
}
```

```
// viewRelated.js

function enableExtendedActions() {
  $('[data-ui="extended"]').attr('disable', null);
}
```

```
// main.js

var userId;

if (isAdmin(getUserRole(userId))) {
  enableExtendedActions();
}
```

Μετά από το πρώτο βήμα, το Concatenation δηλαδή, θα έχουν ενωθεί όλα σε ένα αρχείο ως εξής:

```
var userRolesWebService = '/users/getRole/';

function httpGet(url) {
  return $.ajax(url);
}

var ROLES = {
  ADMIN: 1,
  SIMPLE_MEMBER: 2
};

function isAdmin(roleId) {
  return roleId === ROLES.ADMIN;
}

function getUserRole(userId) {
  return httpGet(userRolesWebService + userId);
}

function enableExtendedActions() {
  $('[data-ui="extended"]').attr('disable', null);
}

var userId;

if (isAdmin(getUserRole(userId))) {
  enableExtendedActions();
}
```

Ήδη γίνεται αντιληπτό ότι αν είχαμε να κάνουμε με μία βάση εκατοντάδων αρχείων, τα οποία είναι ~100 γραμμές το καθένα, θα ήταν **δύσκολο** να πλοηγηθούμε μέσα στον κώδικα και να εντοπίσουμε το κάθε σημείο, κατά την αντίστροφη μηχανική.

Έπειτα, μετά την επεξεργασία που γίνεται στο minification, καταλήγουμε με το εξής αρχείο:

```
function httpGet(url){return $.ajax(url)}function isAdmin(roleId){
  return roleId==ROLES.ADMIN}function getUserRole(userId){return
  httpGet(userRolesWebService+userId)}function enableExtendedActions(
  ){$('[data-ui="extended"]').attr("disable",null)}var
  userRolesWebService="/users/getRole/",ROLES={ADMIN:1,SIMPLE_MEMBER:
  2},userId;isAdmin(getUserRole(userId))&&enableExtendedActions();
```

Η μείωση στο συνολικό μέγεθος του αρχείου είναι αισθητή, καθώς και η δυσκολία ανάγνωσής του, **παρ'ότι ακόμα διατηρείται η εννοιολογικότητα του κώδικα** μέσω της διατήρησης των ονομάτων μεταβλητών και μεθόδων.

Μετά και το τελευταίο βήμα, οπότε έχει ολοκληρωθεί η διαδικασία του obfuscation, το τελικό αρχείο έχει ως εξής:

```
!function(){function n(n){return $.ajax(n)}function t(n){return n===i
  .ADMIN}function u(t){return n(a+t)}function e(){$('[data-
  ui="extended"]').attr("disable",null)}var r,a="/users/getRole/",i={
  ADMIN:1,SIMPLE_MEMBER:2};t(u(r))&&e()}();
```

Είναι προφανές πλέον ότι ο κώδικας δεν έχει **καμία εννοιολογική σύνδεση** με την αρχική του μορφή και οποιαδήποτε προσπάθεια ανάγνωσής του δυσχαιρένεται ακόμα περισσότερο από το γεγονός ότι ακόμα και η δομή του έχει παραλλαχθεί και αποτελείται από **μία και μόνο γραμμή**. Σε περιπτώσεις εμπορικών εφαρμογών όπου ο πηγαίος κώδικας θα είναι της τάξης των 600 αρχείων και συνολικού μεγέθους 30.000 γραμμών,μπορεί να γίνει

εύκολα αντιληπτό ότι θα είναι ασύγκριτα πιο δυσνόητος από το παράδειγμα μας το οποίο προέρχεται από κώδικα 20 γραμμών και 6 αρχείων.

# 4. Διάγραμμα της Επίθεσης

Για να μπορέσουμε να δούμε τα ευπαθή σημεία στην όλη υποδομή και να προτείνουμε αντίμετρα, θα είναι χρήσιμο να αναλύσουμε το διάγραμμα μιας επίθεσης. Στο υποθετικό σενάριο όπου ένας χρήστης θέλει να εκτελέσει επίθεση και να αποσπάσει τα αρχεία μέσων από μία τέτοια υπηρεσία, θα πρέπει να ακολουθήσει την εξής διαδικασια:

## 4.1 Παρακολούθηση της επικοινωνίας

Πρώτο βήμα θα πρέπει να είναι η παρακολούθηση της επικοινωνίας του browser με τον εξυπηρετητή της υπηρεσίας, με σκοπό την ανάλυση του πρωτοκόλλου επικοινωνίας που ακολουθείται και έτσι την εξερεύνηση και μίμηση του από ένα πρόγραμμα-πελάτη κατασκευασμένο για τις ανάγκες του επιτιθέμενου.

### 4.1.1 Παράκαμψη του HTTPS

Εμπόδιο στην παρακολούθηση της επικοινωνίας θα αποτελέσει η υιοθέτηση και χρήση HTTPS από πλευράς της υπηρεσίας. Όταν χρησιμοποιείται αυτό το σχήμα κρυπτογραφίας, κάθε ένα από τα δύο άκρα της επικοινωνίας (browser - εξυπηρετητής) κρυπτογραφεί με ένα γνωστό μεταξύ τους κλειδί την πληροφορία και την αποστέλει κρυπτογραφημένη. Ο παραλήπτης, χρησιμοποιώντας το ίδιο προσυμφωνημένο κλειδί, αποκρυπτογραφεί τα δεδομένα και τα επεξεργάζεται.

Μόνο τρόπο επιτυχούς παρεμβολής ενός τρίτου στο δίαυλο αυτό της επικοινωνίας, αποτελεί η συνθήκη κατά την οποία η παρεμβολή στο κανάλι επικοινωνίας έχει εγκαθιδρυθεί πριν την έναρξη της επικοινωνίας και ο επιτιθέμενος λειτουργεί σαν διαμεσολαβητής (proxy) που οδηγεί στον εξυπηρετητή, κι έτσι διατηρεί δύο διαύλους επικοινωνίας, έναν με το browser και έναν με τον εξυπηρετητή, επεξεργαζόμενος την πληροφορία στο ενδιάμεσο στάδιο κατά βούληση.

## 4.2 Αντίστροφη μηχανική των εντολών που εκτελούνται στη συσκευή του χρήστη

Επόμενο βήμα από την παρακολούθηση της επικοινωνίας αποτελεί η αντίστροφη μηχανική των εντολών που εκτελούνται στο browser. Αυτό επιτυγχάνεται με προσεκτική μελέτη των αρχείων HTML και Javascript που αποστέλονται από τον εξυπηρετητή. Επειδή, όπως αναφέρθηκε, στη μεγάλη πλειοψηφία των περιπτώσεων ο Javascript κώδικας παραδίδεται σε obfuscated μορφή, είναι χρήσιμη η χρήση ενός Javascript Beautifier. Ενός, δηλαδή, εργαλείου αυτόματης επεξεργασίας του κώδικα, το οποίο αναιρεί τα αποτελέσματα του βήματος 2 (και μόνο) της διαδικασίας του obfuscation (Minification). Στην ουσία, αναδιαφορφώνει την διάταξη του κώδικα στο πηγαίο αρχείο, **χωρίς όμως** να επαναφέρει την χαμένη εννοιολογικότητα του. Τα ονόματα των μεταβλητών και των μεθόδων παραμένουν τυχαιοποιημένα, αλλά υπάρχει πλέον ένα πιο κατανοητό συντακτικό.

## 4.3 Μίμηση του πρωτοκόλλου επικοινωνίας και πειραματισμός

Έχοντας “αποκρυπτογραφήσει” τη σημασιολογία και τη λειτουργικότητα του πηγαίου κώδικα και του πρωτοκόλλου επικοινωνίας, και έχοντας ανιχνεύσει την ακολουθία των ενεργειών / κλήσεων που απαιτείται για να αποκτήσει ένας χρήστης πρόσβαση στο πηγαίο αρχείο του μέσου που επιθυμεί, επόμενο βήμα για τον κακόβουλο χρήστη θα είναι να κατασκευάσει ένα πρόγραμμα πελάτη ή ενα σενάριο εντολών σε αντίστοιχη γλώσσα (scripting language) που θα εκτελεί τις απαραίτητες ενέργειες και θα δημιουργεί ένα αντίγραφο του αρχείου στον τοπικό δίσκο.

# Μέρος Β

# Επίθεση Απόκτησης Πρόσβασης στα Πηγαία Αρχεία των Μέσων, Στην Πραξη

# 5. Παρουσίαση των εργαλείων

Κατα την σειρά των βημάτων που θα ακολουθήσουμε, σύμφωνα με το προηγούμενο κεφάλαιο, τα εργαλεία που θα χρησιμοποιήσουμε είναι τα ακόλουθα:

## 5.1 Fiddler

Το Fiddler[10] είναι μία εφαρμογή διαμεσολαβητής κατασκευασμένη με σκοπό την αποσφαλμάτωση διαδικτυακών εφαρμογών. Καταγράφει όλη την κίνηση μεταξύ του υπολογιστή στον οποίον εκτελείται και του διαδικτύου, και την παρουσιάζει σε μία λίστα με δυνατότητες αναζήτησης, φιλτραρίσματος αλλά και επεξεργασίας / επανάληψης. Τέλος η συμπεριφορά του είναι παραμετροποιήσιμη με χρήση σεναρίων εντολών, οπότε μπορεί αυτοματοποιημένα να διαχειρίζεται με διαφορετικό τρόπο κάθε περίπτωση.

Μαζί με το Fiddler, θα χρειαστούμε και το FiddlerScript Editor[11] που διατίθεται ξεχωριστά, οπότε θα πρέπει στα πλαίσια αυτής της εργασίας να το έχουμε εγκατεστημένο.

## 5.2 Google Chrome Developer Tools

Τον τελευταίο χρόνο, υπολογίζεται ότι περισσότερο από το 99% των χρηστών χρησιμοποιούν έναν εκ των 5 πιο δημοφιλών browsers[12]:

- Google Chrome[13]
- Internet Explorer[15] / Edge[16]
- Mozilla Firefox[14]
- Safari[17]
- Opera[18]

Όλοι αυτοί οι φυλλομετρητές παρέχουν εργαλεία αποσφαλμάτωσης, ελέγχου και ανάλυσης των ιστοσελίδων, τα οποία είναι γνωστά ως Developer Tools. Στην παρούσα εργασία χρησιμοποιήθηκε ο Google Chrome και τα Developer Tools που τον συνοδεύουν, αλλά θα μπορούσε να χρησιμοποιηθεί οποιοσδήποτε browser.

Η χρήση των Developer Tools εξυπηρέτησε τον σκοπό της αντίστροφης μηχανικής του πηγαίου κώδικα Javascript και παρακολούθησης των επικοινωνιών με τον εξυπηρετητή μέσω WebSockets.

## 5.3 Simple HTTP File Server[19]

Το Simple HTTP File Server είναι ένα εργαλείο που απλοποιεί την διαδικασία έκθεσης ενός ή περισσοτέρων αρχείων προς προσπέλαση μέσω του πρωτοκόλλου HTTP, χωρίς την ανάγκη για ιδιαίτερες ρυθμίσεις. Στα πλαίσια της εργασίας αυτής χρησιμοποιήθηκε για την εξυπηρέτηση των “τροποποιημένων” από εμάς πηγαίων αρχείων Javascript, κατά την διαδικασία της αντίστροφης μηχανικής.

## 5.4 JS Beautifier

Το εργαλείο που χρησιμοποιήθηκε στα πλαίσια αυτής της εργασίας για Javascript Beautification είναι το http://www.danstools.com/javascript-beautify/

# 6. Βηματική παρουσίαση της επίθεσης

Στα πλαίσια της εργασίας αυτής εκτελέσαμε μια επίθεση με σκοπό την απόσπαση αντιγράφων των αρχείων μουσικής της διαδικτυακής υπηρεσίας Spotify. Στόχος της επίθεσης ήταν η διαδικτυακή πλατφόρμα αναπαραγωγής μουσικής του Spotify, http://play.spotify.com.

## 6.1 Βήμα 1 - Αυθεντικοποίηση

Το πρώτο βήμα που πρέπει να κάνει ένας χρήστης για να αποκτήσει πρόσβαση στην εφαρμογή του Spotify, είναι να αυθεντικοποιηθεί. Αυτό γίνεται από τη φόρμα εισόδου που περιλαμβάνει στην κεντρική της σελίδα η ιστοσελίδα του Spotify Web Player.

Για να αναλύσουμε τι αποστέλλεται από τη φόρμα αυθεντικοποίησης, ανοίγουμε τα Developer Tools του browser πατώντας Ctrl+Alt+I και αφού συμπληρώσουμε τη φόρμα πατάμε “Login”.

Όπως βλέπουμε και στην εικόνα 6.1.1, η αυθεντικοποίηση γίνεται με Αίτηση POST στη διεύθυνση https://play.spotify.com/xhr/json/auth.php και οι παράμετροι που αποστέλλονται είναι οι εξής:

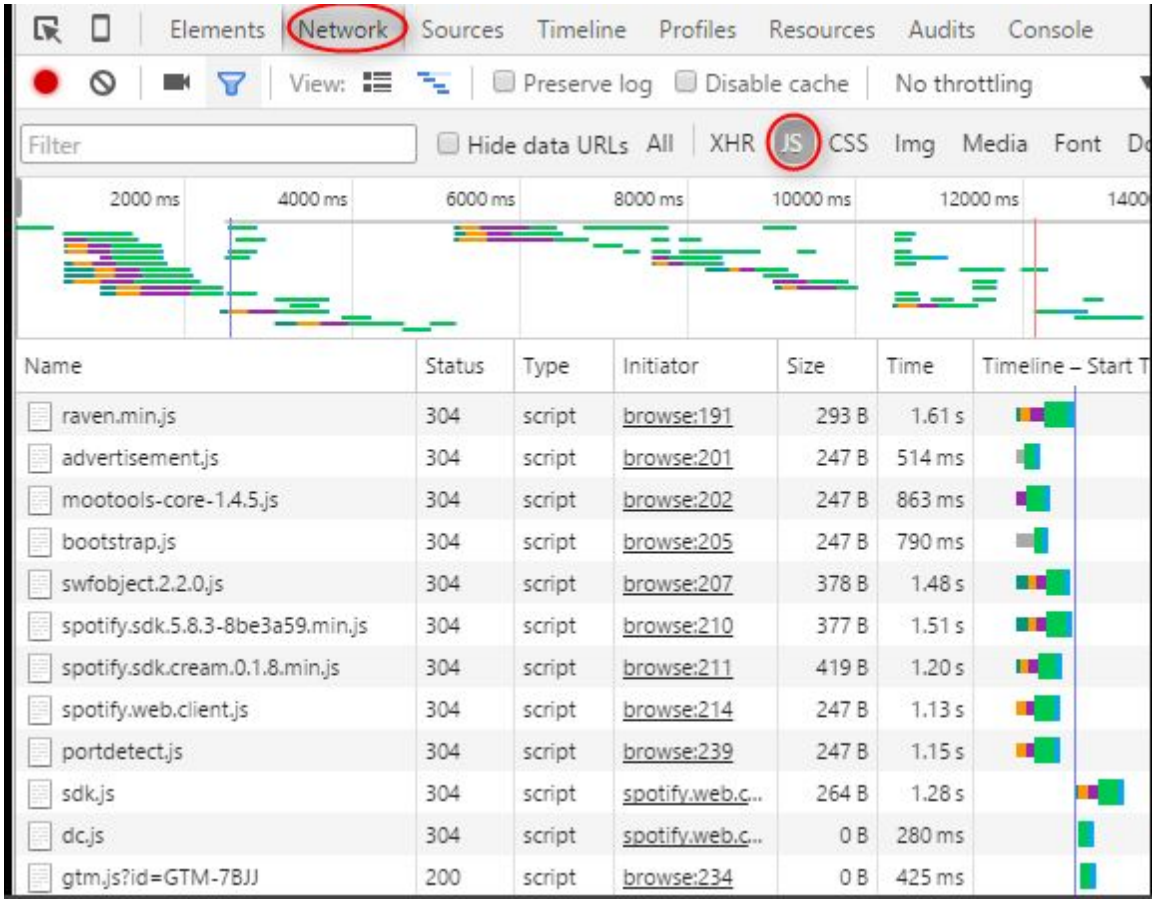
type, username, password, secret, trackingId, referrer, landingURL, cf, f, s, landingURL


*Εικόνα 6.1.1*

Όπως γίνεται αμέσως αντιληπτό, τα πεδία username και password περιέχουν τα στοιχεία αυθεντικοποίησης του χρήστη και το πεδίο secret ένα CSRF protection token. Αυτές τις τιμές θα χρειαστεί να τροποποιήσει κατά βούληση κάποιος χρήστης για να μπορέσει να αυθεντικοποιηθεί μέσω ενός τρίτου, προσωπικού, προγράμματος πελάτη.

## 6.2 Επιθεώρηση των λειτουργιών της υπηρεσίας μέσα από τα Developer Tools

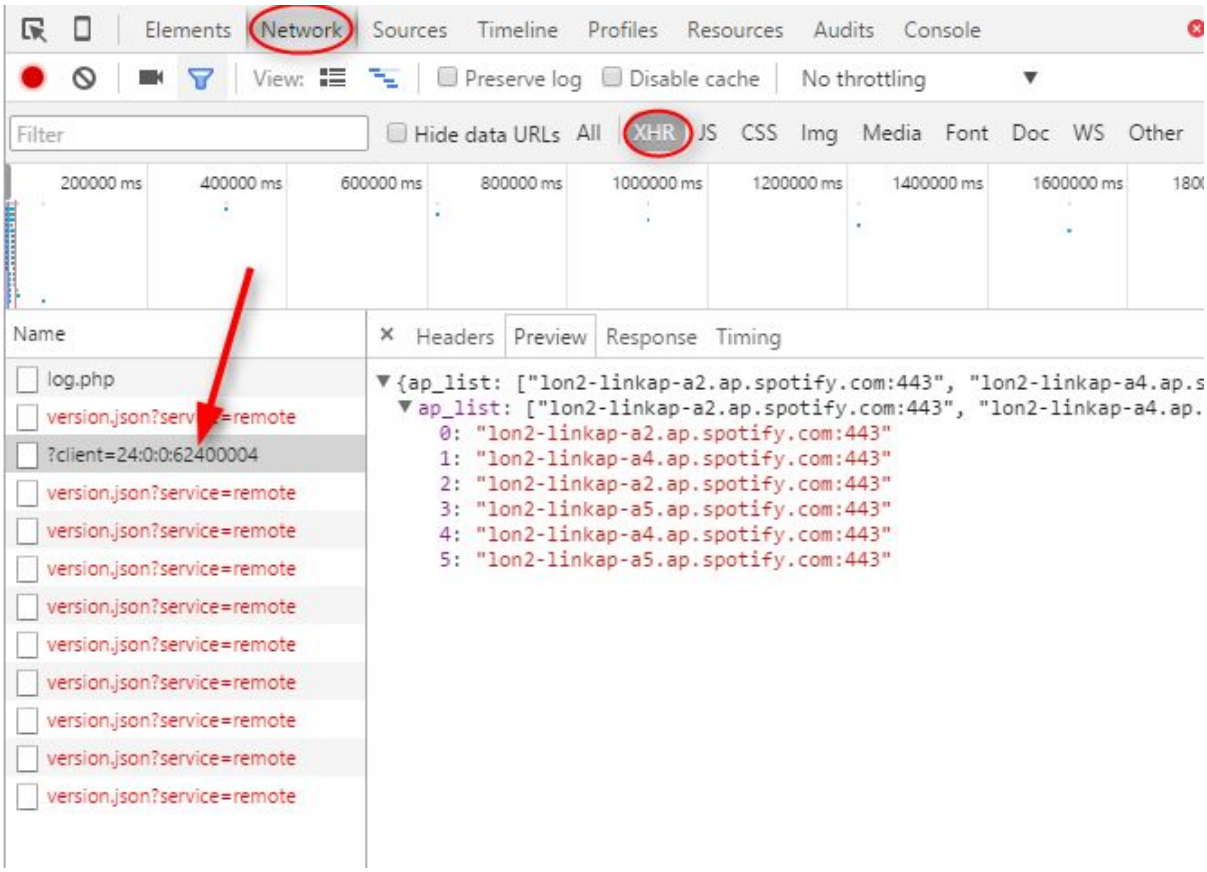
Αφού συνδεθούμε στην υπηρεσία αν πατήσουμε στο tab **Network** των Developer Tools και στη συνέχεια στο φίλτρο **JS και ανανεώσουμε τη σελίδα** μπορούμε να δούμε όλα τα εξωτερικά Javascript αρχεία που φορτώθηκαν:




Από τα αρχεία που βλέπουμε, θα ξεκινήσουμε την ανάλυση / επιθεώρηση από αυτά των οποίων το όνομα τα καθιστά πιο πιθανά να αφορούν τη γενική λειτουργικότητα της εφαρμογής και της αναπαραγωγής μουσικής. Αυτό είναι μία αυθαίρετη επιλογή και δεν μας εγγυάται ότι θα βρούμε τα κατάλληλα αρχεία άμεσα, γι αυτό και αν χρειαστεί θα πρέπει να τα αναλύσουμε όλα.

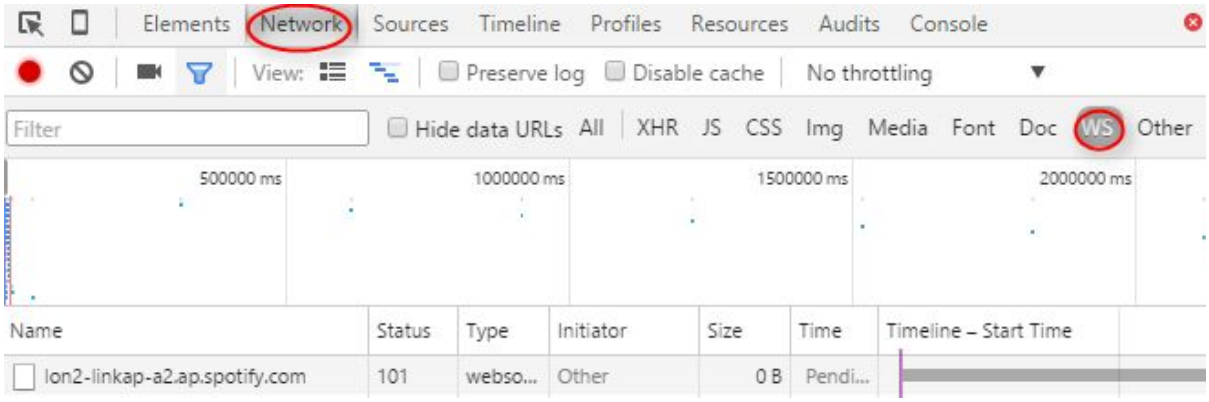
Ας δούμε όμως πρώτα την ευρύτερη δικτυακή συμπεριφορά της υπηρεσίας κατά τη χρήση. Διατηρώντας τα Developer Tools ανοιχτά, προχωράμε στην υπηρεσία στην αναζήτηση μουσικών κομματιών και στην ακρόασή τους. Αφού το κάνουμε αυτό για λίγο, επιστρέφουμε στα Developer Tools και αναζητούμε για ίχνη επικοινωνίας.

Δύο είναι τα κυριότερα σημεία στα οποία θα κοιτάξουμε για ίχνη δικτυακής επικοινωνίας, και αυτά είναι τα AJAX requests[20] και οι WebSockets[21]. Για να επιθεωρήσουμε αυτές τις δύο διαστάσεις του δικτυακού αποτυπώματος της εφαρμογής, πατάμε στα αντίστοιχα φίλτρα στα Developer Tools. Ξεκινάμε με τις αιτήσεις AJAX όπου και πατάμε στο φίλτρο **XHR**:

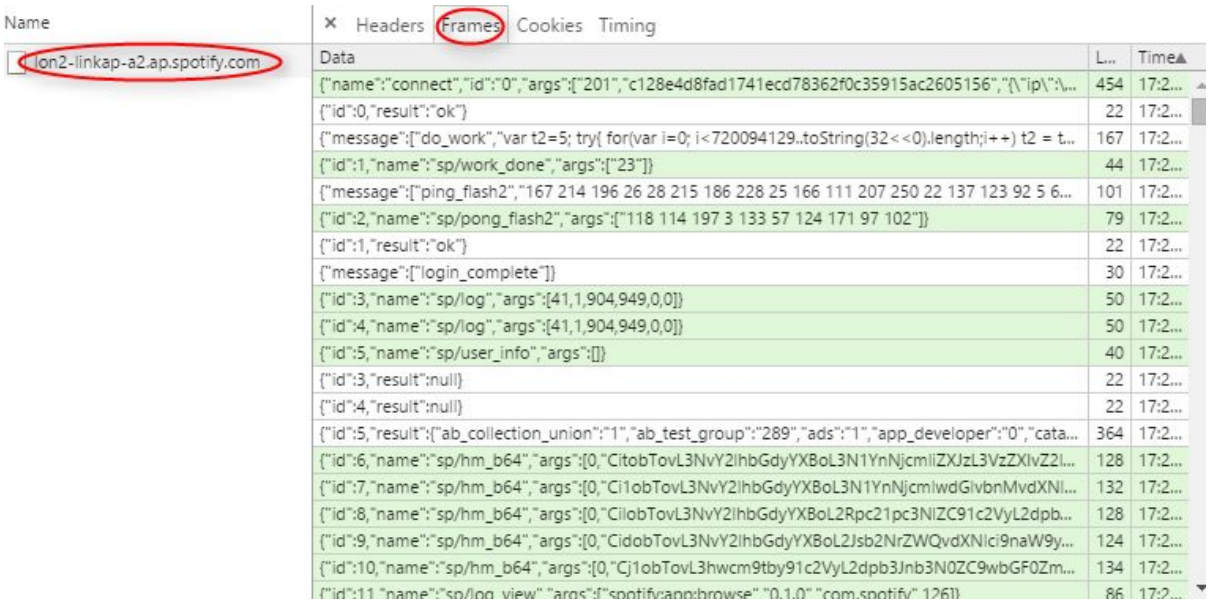



Η ανάλυσή μας σε αυτό το πεδίο δεν έδωσε και ιδιαίτερα αποτελέσματα καθώς απο ό,τι φαίνεται δεν έγιναν σχεδον καθόλου AJAX requests, και πιο συγκεκριμένα μόνο μία έγινε που θα μπορούσε να μας φανεί ενδιαφέρουσα. Αυτή η αίτηση έλαβε από τη διαδικτυακή τοποθεσία [https://apresolve.spotify.com/?client=24:0:0:62400004](https://apresolve.spotify.com/?client=24:0:0:62400004) μία λίστα με **πιθανά** σημεία σύνδεσης.

Προχωράμε την έρευνά μας ελέγχοντας και για επικοινωνίες μέσω WebSockets πατώντας στο αντίστοιχο φίλτρο:




Βλέπουμε ότι υπάρχει μία σύνδεση μέσω WebSocket η οποία εξακολουθεί να είναι ενεργή. Τα Developer Tools μας δίνουν τη δυνατότητα να δούμε τα δεδομένα που έχουν αποσταλεί από τις δύο πλευρές. Για να το πετύχουμε αυτό, πατάμε πάνω στην εγγραφή που μας ενδιαφέρει στη λίστα με τις WebSockets, και έπειτα στην επιλογή Frames:




Εδώ βλέπουμε ότι υπάρχει μια έντονη επικοινωνία μεταξύ του browser και του εξυπηρετητή που ήδη έχει καταγράψει πάνω από 300 μηνύματα και καθώς συνεχίζουμε να χρησιμοποιούμε την εφαρμογή διαρκώς αυξάνονται.

Πρώτη παρατήρηση που μπορούμε να κάνουμε είναι ότι η διεύθυνση με την οποία είναι συνδεδεμένη η WebSocket (φαίνεται στην αριστερή στήλη) δείχνει να είναι μία από τις διευθύνσεις που εξετάσαμε όταν εξετάζαμε τα AJAX Requests. Άρα η υπόθεση μας ότι ήταν πιθανά σημεία σύνδεσης ήταν σωστή.

Επόμενο γεγονός που συμπεραίνουμε παρατηρώντας τα ίχνη της επικοινωνίας αυτής είναι ότι τα δεδομένα μεταφέρονται κωδικοποιημένα στη μορφή JSON[22].

Τρίτον, βλέπουμε ότι μετά το πρώτο μήνυμα που φέρει το διακριτικό *connect*, όλα φέρουν ένα σειριακό αριθμό, πιθανώς για ταυτοποίηση των απαντήσεων όπως αντίστοιχα γίνεται στο πρωτόκολλο TCP[23].

Τέταρτον, παρατηρούμε ένα διαχωρισμό, όπου μερικά μηνύματα φέρουν ιδιότητες *name* και *args* και άλλα φέρουν απλά την ιδιότητα *result.* Επίσης, **για κάθε μήνυμα της μίας ενότητας, υπάρχει και ένα της άλλης**. Είναι κατανοητό λοιπόν ότι τα μεν πρώτα αφορούν αιτήσεις/κλησεις απομακρυσμένων μεθόδων και λειτουργιών, ενώ τα δεύτερα τις αντίστοιχες απαντήσεις. Πέραν αυτών υπάρχουν και μηνύματα που δεν εμπίπτουν σε καμία από τις δύο κατηγορίες και φέρουν μόνο μία ιδιότητα, τη *message*. Αυτά πιθανώς είναι εντολές του εξυπηρετητή προς το browser, καθώς τα άλλα πάντοτε ξεκινάνε από τον υπολογιστή του χρήστη.

Ψάχνοντας ανάμεσα στα μηνύματα αυτά, μπορούμε να διακρίνουμε τις εξής τιμές για την παράμετρο *name*:

- connect
- sp/work_done (απαντώντας στο do_work message)
- sp/pong_flash2 (απαντώντας στο ping_flash2 message)
- sp/log
- sp/user_info
- sp/hm_b64
- sp/log_view
- sp/set_ua
- sp/echo

- sp/track_uri
- sp/track_event
- sp/track_progress

Η *sp/track_uri* εντολή φαίνεται να είναι αρκετά ενδιαφέρουσα. Αν δούμε το μήνυμα που απαντάει στην αίτηση αυτή, μπορούμε να διακρίνουμε μία δομή δεδομένων που περιέχει τις τιμές *lid, tid, type* και *uri.* Όπου η τιμή του *uri* είναι μία διαδικτυακή διεύθυνση του πρωτοκόλλου **RTMP[24]**, και **κατά πάσα πιθανότητα**, αποτελεί τη διεύθυνση του αρχείου μουσικής που ζητούμε.

Επιπλέον, η κλήση της *sp/track_uri* συνοδεύεται από τρεις παραμέτρους:

- το λεκτικό **mp3160**
- ένα δεκαεξαδικό αριθμό 16 bytes **ca6733cd745344279e67a4a48bc9f594**
- το λεκτικό **rtmp**

όπου πιθανώς η πρώτη παράμετρος είναι η ποιότητα του μέσου που ζητούμε, η δεύτερη το αναγνωριστικό του μέσου αυτού, και η τρίτη η μορφή στην οποία θέλουμε να έχουμε πρόσβαση.

Ο στόχος μας λοιπόν γίνεται λίγο πιο συγκεκριμένος, και μάλιστα περιφέρεται γύρω από την αποκωδικοποίηση του πρωτοκόλλου επικοινωνίας έως του να φτάσει το πρόγραμμα-πελάτης στην *sp/track_uri* εντολή.

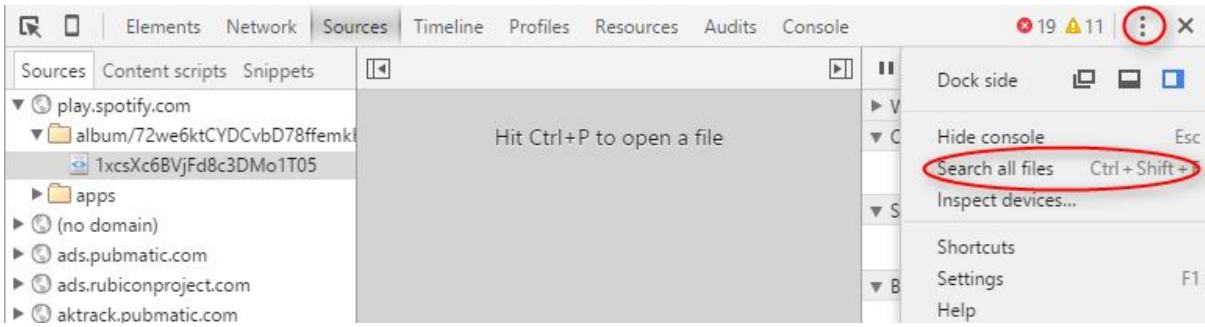
Επόμενη παρατήρηση που μπορούμε να κάνουμε είναι ότι το πεδίο *args* τις περισσότερες φορές περιέχει μια συμβολοσειρά η οποία θα μπορούσε να είναι κωδικοποιημένη σε αριθμό του 64δικού συστήματος (Base64 encoded). Μπορούμε να δοκιμάσουμε να αποκωδικοποιήσουμε μερικά (ή και όλα) με σκοπό να κατανοήσουμε τι έχει σταλεί, αλλά δεν θα βοηθήσει τόσο πολύ η διαδικασία και θα είναι πολύ χρονοβόρα. Καλύτερα να επεξεργαστούμε τα πηγαία αρχεία Javascript να εκτυπώνουν αυτόματα την αποκωδικοποιημένη μορφή των παραμέτρων αυτών, ενώ εμείς παράλληλα επιθεωρούμε τον κώδικα βήμα βήμα.

## 6.3 Εξέταση των αρχείων Javascript

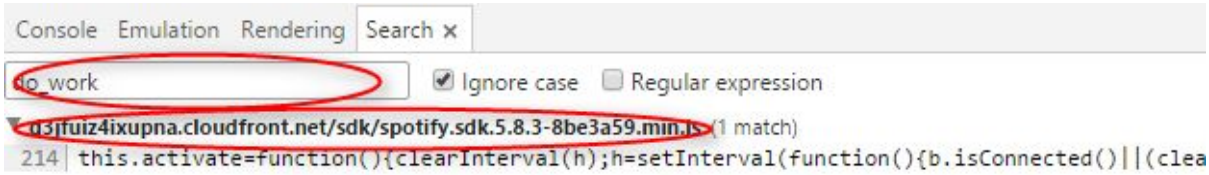
### 6.3.1 Προετοιμασία του πρώτου αρχείου προς επιθεώρηση

Εφόσον πλέον έχουμε μια πιο ολοκληρωμένη εικόνα για την επικοινωνία την οποία θέλουμε να αποδομήσουμε και να μελετήσουμε, δεν υπάρχει ανάγκη να ερευνήσουμε στα τυφλά όλα τα αρχεία Javascript που εντοπίσαμε στο προηγούμενο βήμα.

Ένας έξυπνος τρόπος να ξεκινήσουμε είναι να κάνουμε μια αναζήτηση μέσα σε όλα τα αρχεία για τις εντολές που είδαμε πριν π.χ. την *do_work*. Για να εκτελέσουμε την αναζήτηση αυτή, αρκεί μέσα από τα Developer Tools να πατήσουμε στο μενού πάνω δεξιά και μετά στην εντολή “Search all files”.




Και στη συνέχεια να εισάγουμε “do_work” στο πεδίο αναζήτησης και να πατήσουμε Enter:




Βλέπουμε ότι η εντολή αυτή βρέθηκε σε ένα μόνο αρχείο, το **spotify.sdk.5.8.3-8be3a59.min.js** οπότε μπορούμε να ξεκινήσουμε από αυτό την ανάλυσή μας.

Για να ανοίξουμε το αρχείο αυτό, το βρίσκουμε στη λίστα **JS** του **Network** tab, και κάνουμε διπλό κλικ επάνω του. Το αρχείο ανοίγει σε μία νέα καρτέλα, από όπου επιλέγουμε όλο το περιεχόμενο, ανοίγουμε σε νέα καρτέλα το **Javascript Beautifier[25]** και επικολλούμε το περιεχόμενο στο αντίστοιχο πεδίο.

Στη δεξιά στήλη υπάρχουν διάφορες ρυθμίσεις με τις οποίες μπορούμε να επιλέξουμε την επιθυμητή τελική μορφοποίηση του κώδικα. Πατώντας στο κουμπί **Format Code**, μετασχήμαζεται ο κώδικας στην επιθυμητή μορφή. Επιλέγουμε τα περιεχόμενά του, τα αντιγράφουμε σε ένα νέο αρχείο και είμαστε έτοιμοι να ξεκινήσουμε.

Για ευκολία στη χρήση και στη συνέχεια, ας αποθηκεύσουμε αυτό το αρχείο ως **spotify.sdk.js**.

Πρώτο βήμα στην ανάλυση, θα είναι να παραλλάξουμε το αρχείο και όταν στέλνει ένα αίτημα μέσω WebSocket να εκτυπώνει και στην κονσόλα το αντίστοιχο μήνυμα μη-κωδικοποιημένο. Για να ξεκινήσουμε αυτή τη διαδικασία, ψάχνουμε στο πηγαίο αρχείο για τη συμβολοσειρά **hm_b64** που είναι η τιμή του πεδίου *name* για τα μηνύματα που δείχνουν κωδικοποιημένα. Ψάχνοντας για τη συμβολοσειρά αυτή τη βρίσκουμε σε τρία σημεία μέσα στο αρχείο, ένα από τα οποία είναι το εξής:

```
D = function() {
  if (0 == A.length) c([{
    reply: C
  }], 200);
  else {
    var a = (new Date).getTime(),
      f = [0, window.btoa(n.serializeToStringSync(k))];
    this.setRequestFrameData(0, {
      request: A
    });
    f.push(window.btoa(this.getRequestFrame(0)));
    d.log("Spotify.Hermes.Request", ["It took", (new Date).getTime() -
      a, "ms to serialize the request"
    ], "parsing_times");
    b.rpc("hm_b64", f, E, g, this, h || !1, 2, p)
  }
},
```

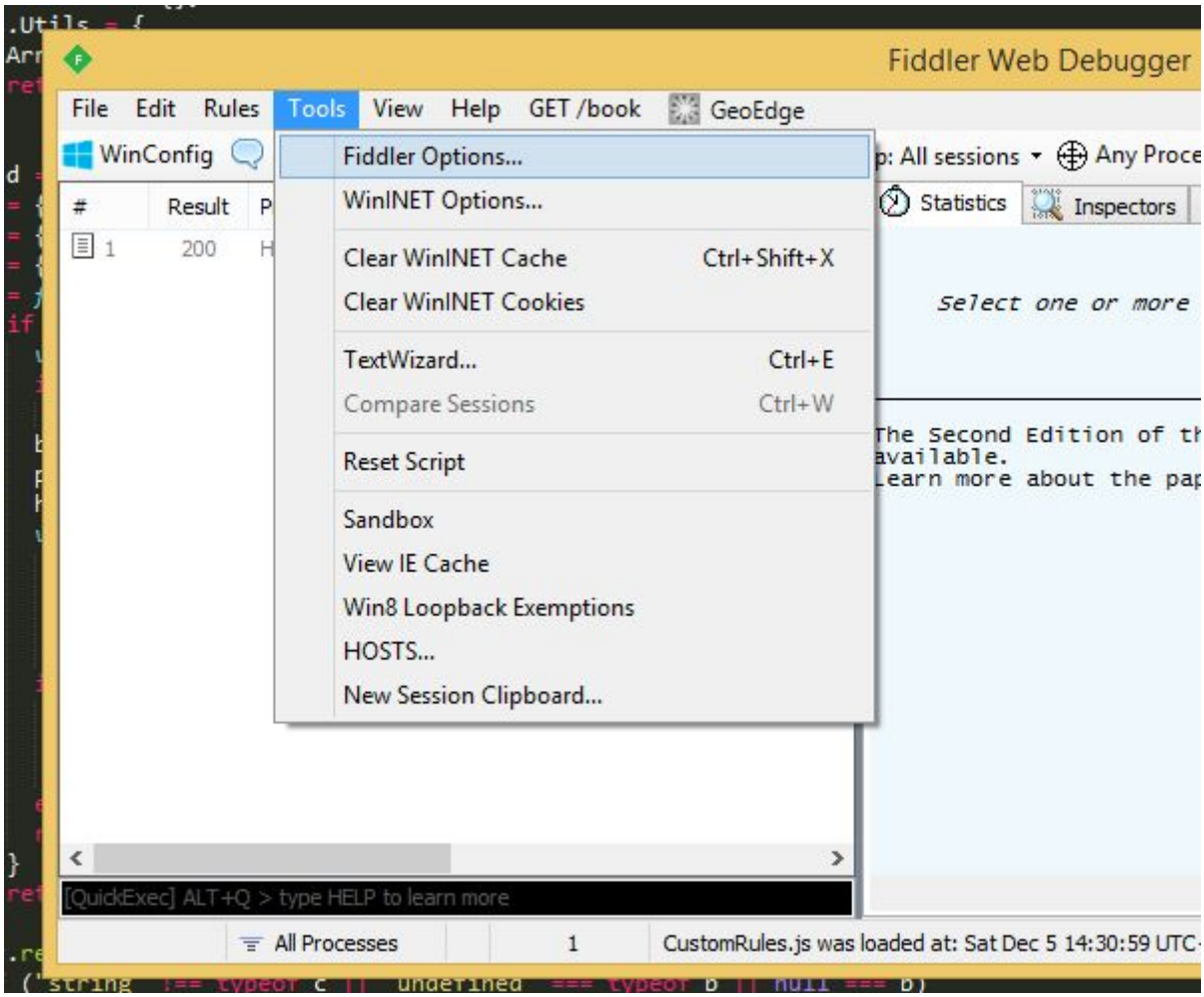
Γίνεται αμέσως αντιληπτό ότι η μέθοδος *rpc* είναι που στέλνει τα αιτήματα, οπότε θα μπορούσαμε να προσθέσουμε τον κώδικά μας εκεί μέσα. Παρ'όλα αυτά όμως, προσέχουμε και στην προηγούμενη γραμμή, όπου χρησιμοποιείται κάποια εσωτρική *log* μέθοδος για να εκτυπώσει διάφορα δεδομένα αποσφαλμάτωσης. Συμπεραίνουμε ότι, για να μην υπάρχουν δεδομένα στην κονσόλα, μάλλον η μέθοδος αυτή δεν εκτυπώνει τίποτα σε περιβάλλοντα παραγωγής (production environment). Οπότε πρώτος στόχος μας θα είναι να την ενεργοποιήσουμε, καθώς τα δεδομένα αποσφαλμάτωσης είναι δυνατόν να μας παράσχουν πολύ χρήσιμες πληροφορίες.

Εκτελούμε μια αναζήτηση με τον ορο **"log = "** και αφού περιηγηθούμε λίγο στα αποτελέσματα καταλήγουμε στην αρχή του αρχείου, στις πρώτες γραμμές, όπου ορίζεται η μέθοδος αυτή. Τοποθετούμε εκεί ένα breakpoint γράφοντας την εντολή **debugger;** σε μια κενή γραμμή και είμαστε έτοιμοι να δούμε το αρχείο να εκτελείται:

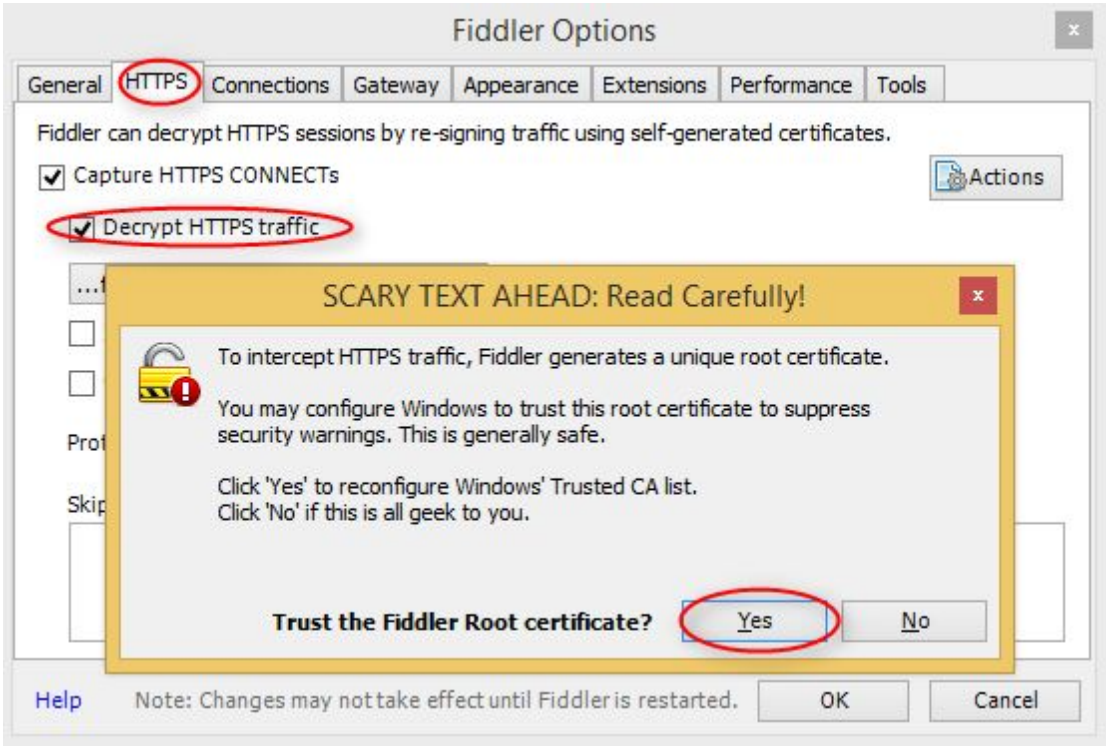
```
78   this.log = function(a, b, g) {
79     debugger;
80     return c("log", a, b, g)
81   };
```

### 6.3.2 Χρήση του Fiddler ως proxy για αντικατάσταση των αυθεντικών Javascript αρχείων με τα τροποποιημένα

Ξεκινάμε την εκτέλεση του Fiddler, και πληγούμαστε στο μενού **Tools -> Fiddler Options**:



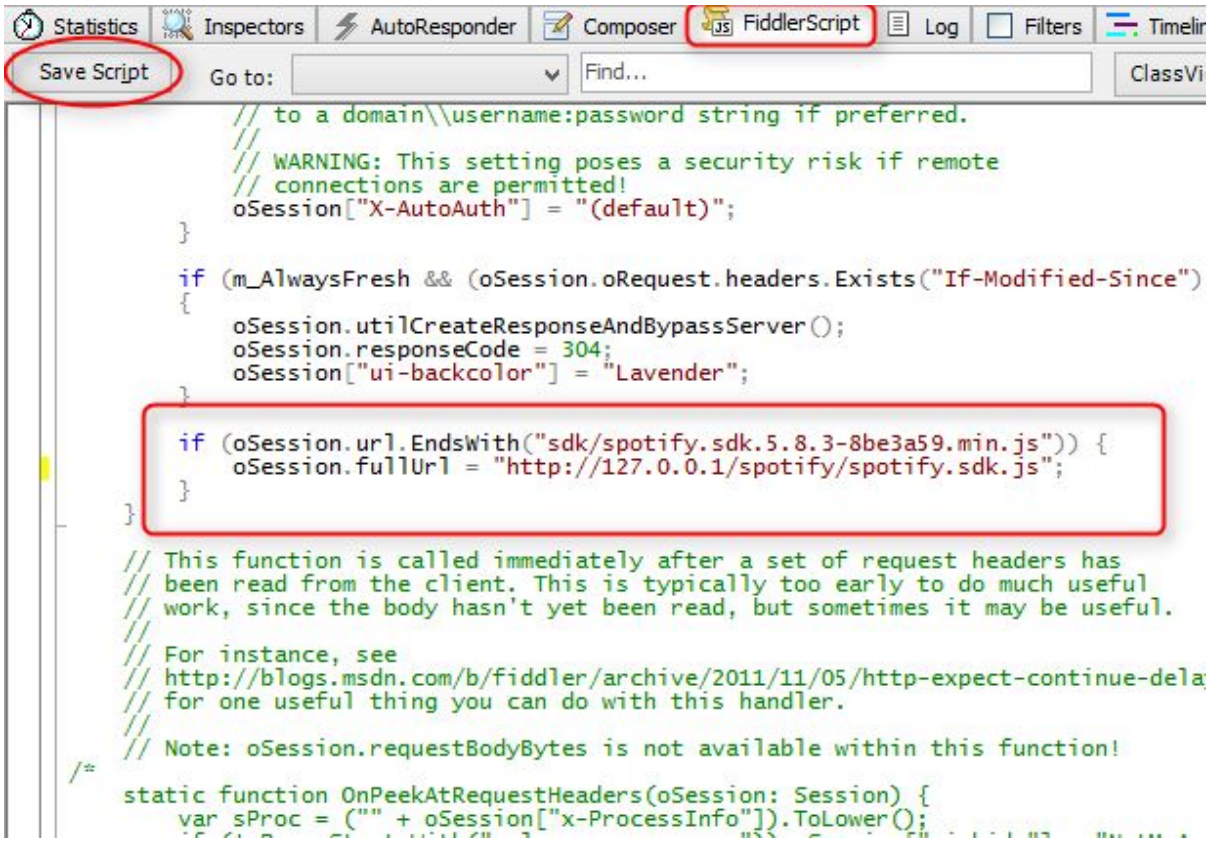

Στο παράθυρο που αναδύεται, επιλέγουμε την καρτέλα **HTTPS** και την επιλογή **Decrypt HTTPS Traffic** και στο πλαίσιο διαλόγου που θα εμφανιστεί επιλέγουμε **Yes** (και σε όποια άλλα διαδοχικά πλαίσια διαλόγου του λειτουργικού συστήματος) έτσι ώστε εγκατασταθεί το πιστοποιητικό του Fiddler ως Εμπιστευόμενη Αρχή Πιστοποίησης, κι έτσι να μπορούμε να συλλαμβάνουμε κίνηση μέσω HTTPS:




Εν συνεχεία επιλέγουμε **OK** στο παράθυρο Fiddler Options και επανεκκινούμε το Fiddler για να εφαρμοστούν οι αλλαγές μας.

Επόμενο βήμα είναι να ορίσουμε έναν τοπικό εξυπηρετητή που θα διανέμει το τροποιημένο μας αρχείο. Όπως προαναφέραμε, για αυτό το σκοπό θα χρησιμοποιήσουμε το Simple HTTP File Server. Το εκκινούμε, κάνουμε δεξί κλικ στο μενού στα αριστερά και προσθέτουμε τα τροποιημένα αρχεία μας.

Ανοίγουμε το Fiddler, επιλέγουμε το tab **FiddlerScript** από τη δεξιά στήλη, και προσθέτουμε τον ακόλουθο κώδικα στο τέλος της μεθόδου **OnBeforeRequest**:




Φυσικά, η δεύτερη γραμμή πρέπει να αντιστοιχεί στη σωστή διεύθυνση του αρχείου όπως διανέμεται από το HTTP File Server. Κατόπιν, πατάμε **Save Script** και είμαστε έτοιμοι.

Επιστρέφουμε στο browser μας και κάνουμε ανανέσωση της σελίδας **με παράλληλη διαγραφή της κρυφής μνήμης (cache memory)**. Μόλις η σελίδα ανανεωθεί βλέπουμε ότι κατευθείαν μπαίνει μέσα στη log μέθοδο, και σταματάει η εκτέλεση όταν φτάσει στο breakpoint που θέσαμε.

Προχωράμε περνώντας πάνω από την γραμμή που βρισκόμαστε (**step over**) και στην επόμενη γραμμή εισερχόμαστε μέσα (**step in**) και καταλήγουμε σε έναν έλεγχο, **όπου η εκτύπωση θα γίνει μόνο αν επιστρέψει αληθή τιμή**:

```
36 DebuggerJS = Spotify.DebuggerJS = new function() {
37   this.Parsers = {};
38   this.Loggers = {};
39   this.Parsers = {};
40   this.Utils = {
41     isArray: Array.isArray || function(a) {
42       return "[object Array]" == Object.prototype.toString.call(a)
43     }
44   };
45   var d = !1,
46     n = {},
47     k = {},
48     a = {},
49     c = function(c, b, g, p) {  c = "log", b = "Spotify.Managers.ProtobufS
50       if (d) {
51         var h;  h = undefined
52         if (!DebuggerJS.Utils.isArray(g)) throw Error(  g = ["Registering
53           "The message argument should be an array");
54         b = "string" === typeof b ? b : "";  b = "Spotify.Managers.Protobu
55         p = "string" === typeof p ? p : "";  p = "corejs"
56         h = b;  h = undefined, b = "Spotify.Managers.ProtobufSchemasManage
57         var f = !1,  f = undefined
58           m = !1,  m = undefined
59           f = 0 === n.tags.length ? !0 : 0 <= n.tags.indexOf(p) ? !0 : !1,
60           m = 0 === n.modules.length ? !0 : 0 <= n.modules.indexOf(h) ?  m
61           !0 : !1;
62         if (f && m)  f = undefined, m = undefined
63           for (var q in k) h = a[q].parse(b, g, p), "log" === c ? k[q].log.
64               this, h) : "warn" === c ? k[q].warn.apply(this, h) : "error"
65             c && k[q].error.apply(this, h);
66         else return !1;
67         return !0
68       }
69       return !1
```

Όπως βλέπουμε η μεταβλητή *d* έχει **προκαθορισμένη τιμή false**, για να μην εκτελείται ποτέ η αποσφαλμάτωση σε production environment. Αλλάζουμε την τιμή της μεταβλητής σε **true** στο αρχείο spotify.sdk.js, αφαιρούμε τη debugger; εντολή που εισαγάγαμε πριν, αποθηκεύουμε το αρχείο και ανανεώνουμε τη σελίδα καθαρίζοντας τη μνήμη cache.

Εκτελώντας τον κώδικα στην σελίδα βλέπουμε ότι προκύπτει σφάλμα εκτέλεσης, καθώς δεν έχουν οριστεί καλά οι ετικέτες για τις οποίες θα λειτουργεί η εκτύπωση. Αν αντικαταστήσουμε τις γραμμές 59 και 60 με το ακόλουθο:

```
56          h = b;
57          var f = !1,
58            m = !1,
59            f = true,
60            m = true;
61          if (f && m)
62            for (var q in
63                this, h) :
```

και ανανεώσουμε τη σελίδα αδειάζοντας την κρυφή μνήμη, βλέπουμε ότι πλέον όλα τα δεδομένα αποσφαλμάτωσης εκτυπώνονται στην κονσόλα:


Παρατηρώντας τα δεδομένα αποσφαλμάτωσης προκύπτουν διάφορες χρήσιμες πληροφορίες, όπως το ότι τα μηνύματα είναι κωδικοποιημένα με το πρωτόκολλο Protocol Buffers[26] της Google και κατόπιν Base64 encoded για αποστολή μέσα σε ένα JSON αντικείμενο. Και όλες οι αιτήσεις που κωδικοποιούνται με αυτό τον τρόπο, περνάνε από τη μονάδα **Spotify.Hermes.Request** του αρχείου που αναλύουμε.

## 6.4 Παρακολούθηση της επικοινωνίας από WebSockets

Τώρα λοιπόν που έχουμε στη διάθεση μας πληροφορίες αποσφαλμάτωσης για όλες τις αιτήσεις, είμαστε σε θέση να εποπτεύσουμε όλο το πρωτόκολλο επικοινωνίας. Για να γίνει αυτό καλύτερα όμως, θα πρέπει να απαλλαγούμε από το θόρυβο που υπάρχει στην επικοινωνία λόγω των διαφόρων ενεργειών μας μέσα στη συνεδρία αυτή αλλά και λόγω των διαδικασιών καταγραφής που απλώς μεγαλώνουν τον όγκο της επικοινωνίας. Γι’αυτό το λόγο, κάνουμε μια ανανέωση στο παράθυρο του browser, και κατευθείαν ξεκινάμε την αναπαραγωγή ενός τραγουδιού.

Στο σημείο αυτό καταγράφουμε τις πληροφορίες που έχουμε από την ενότητα Frames της κονσόλας δικτύου αλλά και από τα δεδομένα αποσφαλμάτωσης:

1. Η εφαρμογή πελάτη ξεκινάει με ένα μήνυμα connect το οποίο φέρει ως παραμέτρους τρεις τιμές, εκ των οποίων:
    - μία σταθερή τιμή, **201**

- μία μυστική ποσότητα που παρέχεται από τη σελίδα, σχετική με τη συνεδρία του χρήστη
- ένα JSON αντικείμενο που περιέχει πληροφορίες όπως η διεύθυνση IP του χρήστη, η ώρα που γίνεται το αίτημα, άλλη μία μυστική ποσότητα που παρέχεται από τη σελίδα, ο τύπος του browser και η έκδοση του λογισμικού πελάτη που βρίκεται σε εκτέλεση.

2. Ο εξυπηρετητής απαντά με ένα μήνυμα **ok**
3. Ο εξυπηρετητής στέλνει ένα μήνυμα με *message*=***do_work***
4. Ο browser απαντάει με ένα μήνυμα του τύπου **sp/work_done** και μία αριθμητική τιμή. Το συγκεκριμένο βήμα αποτελεί ένα μέρος της αυθεντικοποίησης του πελάτη, με χρήση του σχήματος Challenge-Response[27].
5. Ο εξυπηρετητής στέλνει ένα μήνυμα με *message*=**ping_flash2**
6. Ο browser απαντάει με ένα μήνυμα τύπου **sp/pong_flash2** και μια ακολουθία από αριθμητικές τιμές. Το συγκεκριμένο βήμα αποτελεί και αυτό μέρος της αυθεντικοποίησης του πελάτη, με χρήση του σχήματος Challenge-Response.
7. Ο εξυπηρετητής απαντάει με μήνυμα *message*=**login_complete**

Συνεπώς είναι ασφαλές να υποθέσουμε ότι μέχρι αυτό το σημείο εκτελούνται ενέργειες αυθεντικοποίησης. Έπειτα παρατηρούμε ότι:

1. Ο browser στέλνει ένα μήνυμα τύπου **sp/user_info**
2. Ο εξυπηρετητής απαντάει με διάφορες πληροφορίες σχετικές με τον υπο σύνδεση χρήστη, όπως η χώρα, οι περιορισμοί που υπάρχουν στο λογαριασμό, κ.α.
3. Ο browser αιτείται διάφορες πληροφορίες περιεχομένου του χρήστη, όπως
   - τις “αγαπημένες” του λίστες μουσικής
   - άλλους χρήστες που ακολουθεί
     hm://socialgraph/subscriptions/user/<username>
   - χρήστες που τον ακολουθούν
     hm://socialgraph/subscribers/user/<username>
   - τα αναγνωριστικά των λιστών μουσικής του
     hm://playlist/user/giorgostd/rootlist
   - τα περιεχόμενα των λιστών αυτών
     hm://playlist/

Και τέλος γίνεται η αίτηση του **sp/track_uri** στην οποία η απάντηση περιέχει την τελική διεύθυνση του μέσου που θα αναπαραχθεί. Επίσης πρέπει να σημειωθεί ότι μαζί με όλες τις αιτήσεις που αναφέρθηκαν έχουν σταλεί και αιτήσεις **sp/log** οι οποίες επειδή είναι φύσης καταγραφής στατιστικών της επιχειρησιακής λογικής και δεν επηρεάζουν την εξέλιξη της λειτουργίας, παραλήφθηκαν.

## 6.5 Ανάλυση πρωτοκόλλου επικοινωνίας

Επόμενο βήμα για την πλήρη κατανόηση του πρωτοκόλλου επικοινωνίας, είναι η αντίστροφη μηχανική των συγκεκριμένων ενεργειών που καταγράψαμε στο προηγούμενο κεφάλαιο. Για να το κάνουμε αυτό, ξεκινάμε και τις αναζητούμε στο πηγαίο αρχείο Javascript και παρακολουθούμε τις ακολουθίες των κλήσεων που τις περιβάλλουν.

### 6.5.1 sp/connect

Ξεκινάμε την αναζήτησή μας με την εντολή **sp/connect**. Επειδή είναι η πρώτη κλήση στην ακολουθία των ενεργειών, θα ξεκινήσουμε αντίστροφα, ψάχνοντας στον **HTML πηγαίο κώδικα της σελίδας** για αναφορές των δεδομένων που αποστέλλονται σαν παράμετροι της connect. Αυτό το βήμα εκτελείται με το σκεπτικό ότι, αφού δεν έχει προηγηθεί κάποια αλλεπίδραση του χρήστη ή άλλο AJAX Request που να παρέχει τα δεδομένα αυτά, θα πρέπει να υπάρχουν στην ιστοσελίδα ή σε κάποια άλλη μορφή προσωρινής αποθήκευσης συνεδρίας όπως τα cookies[28] ή το localStorage[29].

Η αναζήτηση φέρνει αποτελέσματα, και συγκεκριμένα βρίσκουμε στην σελίδα τον εξής κώδικα:

```
<script type="text/javascript" charset="utf-8">
    var login = new Spotify.Web.Login(
        document,
        {"ads_core_source":false,"aps":{"resolver":{"hostname":"apresolve.spotify.com",
        62400004,
        Spotify.Web.App.initialize.bind(Spotify.Web.App)
    );
    login.init();
</script>
```

ο οποίος εν τέλει περιέχει όλες τις παραμέτρους που αποστέλλονται στην **connect**.

### 6.5.2 do_work, sp/work_done

Αναζητούμε για τη συμβολοσειρά “do_work” στο πηγαίο αρχείο (μαζί με τα εισαγωγικά, καθώς θέλουμε να βρούμε αναφορές σε αυτή μέσα στον κώδικα) και καταλήγουμε στο εξής εύρημα:

```
}
b = f.message ? f.message[0] : null;
"token_lost" === b ? a.trigger(c.TOKEN_LOST, {}) : "do_work" === b ?
  a.trigger(c.WORK, f.message[1]) : "disconnect" === b ?
  "password-changed" ===
  f.message[1] && a.trigger(c.PASSWORD_CHANGED, f.message[1]) :
  "ping_flash" === b ? a.trigger(c.PING_FLASH, f.message[1]) :
  "ping_flash2" === b ? a.trigger(c.PING_FLASH2, f.message[1]) :
```

βλέπουμε δηλαδή ότι γίνεται trigger μια callback μέθοδος που έχει αντιστοιχιστεί στη σταθερά c.WORK οπότε αναζητούμε πάλι για “.WORK” (χωρίς το c γιατί λόγω του obfuscation θα έχει άλλο όνομα στα υπόλοιπα σημεία αυτή η μεταβλητή)

```
n.prototype.initialize = function(a, c) {
  this._gateway = a;
  this._audioManager = c;
  this._gateway.bind(k.WORK, this._onWork, this);
  this._gateway.bind(k.PING_FLASH,
    this._onFlashWork, this);
  this._gateway.bind(k.PING_FLASH2, this._onFlashWork, this)
}
)(Spotify);
```

οπότε οδηγούμαστε στην _onWork μέθοδο:

```
n.prototype._onWork = function(a) {
  eval(a.params)
};
```

η οποία στην ουσία αυτό που κάνει είναι να καλέσει την native Javascript μέθοδο eval[30] με παράμετρο τον κώδικα που έχει σταλεί από τον εξυπηρετητή. Η eval μέθοδος το μόνο που κάνει είναι να εκτελεί τον κώδικα που της δίνεται σε συμβολοσειρά σαν παράμετρος και να επιστρέφει το αποτέλεσμα που παράγεται (αν παράγεται κάποιο).

Αυτό λοιπόν που επιτυγχάνει αυτό το βήμα, είναι να επαληθεύσει ότι το πρόγραμμα πελάτης είναι όντως ένας browser (βασιζόμενο στην υπόθεση ότι είναι λιγότερο πιθανό για ένα άλλου είδους πρόγραμμα να μπορεί να εκτελεί Javascript κώδικα). Το αποτέλεσμα που παράγεται μέσω της eval αποστέλλεται πίσω στον εξυπηρετητή με την εντολή sp/work_done.

### 6.5.3 ping_flash2, sp/pong_flash2

Κινούμενοι αντιστοίχως με την προηγούμενη περίπτωση, αναζητούμε το “ping_flash2” και βρίσκουμε πάλι event αρχιτεκτονική με triggers, όπου καταλήγουμε στην εξής μέθοδο:

```
n.prototype._onFlashWork = function(a) {
  this.run(a.params)
};
n.prototype._onWorkNotDone = function(a) {
  this._codeValidationPassed = !1
};
n.prototype._onWorkDone = function(a) {
  this._codeValidationPassed = !0
};
n.prototype.reply = function() {
  var a = Array.prototype.slice.call(arguments);
  this._gateway.rpc("work_done", a, this._onWorkDone.bind(this), this._onWorkNotDone
    .bind(this), this, !1, 0, "work_done")
};
n.prototype.run = function(a) {
  var c = "undefined 0";
  this._interface || (this._interface = this._audioManager.getInterface());
  try {
    c = this._interface.run(a)
  } catch (l) {}
  this._gateway.rpc("pong_flash2", [c], function() {}, function() {},
    this, !1, 0, "pong_flash2")
};
```

Ψάχνοντας την _interface.run (με τον όρο αναζήτησης run παλι γιατι το _interface είναι τοπική μεταβλητή) έχουμε το εξής:

```
Spotify.Flash.PlayerInterface = function(d, n) {
  Spotify.EventTarget.call(this);
  var k = this,
    a = new Spotify.Events,
    c;
  this.hasSoundCapabilities = !0;
  var l = function(a) {
    this.trigger(a.type, a.params)
  };
  this.run = function(a) {
    return "undefined" !== typeof c && c.isLoaded ? c.getSWF().sp_run(a) :
      !1
  };
  this.playpause = function(a) {
    return "undefined" !== typeof c && c.isLoaded && this.hasSoundCapabilities ?
      c.getSWF().sp_playpause(a) : !1
  };
  this.position = function(a) {
    try {
      if ("undefined" !== typeof c && c.isLoaded && this.hasSoundCapabilities)
        return c.getSWF().sp_time(a)
    } catch (g) {
      return 0
    }
  };
  this.getPlayerState = function(a) {
    "undefined" !== typeof c && c.isLoaded && this.hasSoundCapabilities &&
      (response = c.getSWF().sp_playerState(a));
    return response
  };
  this.seek = function(a, g) {
    if ("undefined" !== typeof c && c.isLoaded && this.hasSoundCapabilities)
      return c.getSWF().sp_seek(a, g)
  };
  this.pause = function(a) {
    "undefined" !== typeof c && c.isLoaded && this.hasSoundCapabilities &&
      c.getSWF().sp_pause(a)
  };
```

Το συγκεκριμένο σημείο του κώδικα είναι ο κώδικας ελέγχου του εργαλείου αναπαραγωγής μουσικής της εφαρμογής, το οποίο, σύμφωνα με τα ευρύματα μας, είναι ένα αρχείο Adobe Flash[31]. Όλες οι εντολές αναπαραγωγής, σταματήματος, αλλαγής τραγουδιού περνάνε από εκεί, και πέρα από αυτές υπάρχει και η sp_run, η οποία υπολογίζει την τιμή απάντησης που θα σταλεί με το sp/pong_flash2.

Αυτό το βήμα διασφαλίζει ακόμα περισσότερο την εφαρμογή, καθώς ελέγχεται και η φύση του εργαλείου αναπαραγωγής της μουσικής, προσθέτει μία ακόμα διάσταση που πρέπει να γίνει αντίστροφη μηχανική και εν τέλει ένα επιπλέον επίπεδο ασφάλειας.

Στα πλαίσια της εργασίας αυτής, δεν έγινε αντίστροφη μηχανική του εν λόγω SWF αρχείου, γιατί εντοπίστηκε ένας παράπλευρος τρόπος αποφυγής του προβλήματος. Καθώς το αρχείο αυτό δεν ζητάει αυθεντικοποίηση από την HTML σελίδα που το φιλοξενεί, απαντάει ανεξαιρέτως σε όλες τις αιτήσεις Challenge-Response που θα περαστούν σε αυτό.

Συνεπώς, είναι δυνατή η λήψη του απο το διαδίκτυο, ενσωμάτωση του σε μια εσωτερική σελίδα κατασκευασμένη από το χρήστη, και προώθηση σε αυτό των δεδομένων του **ping_flash2**. Τα δεδομένα που απαντάει μπορούν να προωθούνται κατευθείαν και αυτομάτως στον εξυπηρετητή με την **sp/pong_flash2** από το τρίτης φύσης πρόγραμμα-πελάτη. Περισσότερες πληροφορίες παρατίθενται στο Μέρος Γ.

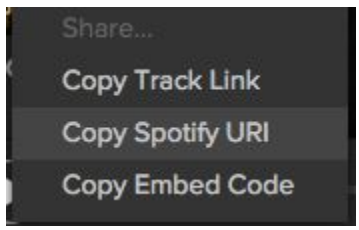
### 6.5.4 sp/user_info

Η sp/user_info εντολή δεν στέλνει παραμέτρους στον εξυπηρετητή παρα μόνο αιτεί πληροφορίες απο αυτόν, και γι αυτο τον σκοπό δεν θα μελετηθεί περαιτέρω, εκτός αν αποδειχτεί ότι οι υπόλοιπες διαδικασίες εξαρτώνται από αυτήν.

### 6.5.5 sp/track_uri

Από τις υπόλοιπες κλήσεις, αυτή που μας ενδιαφέρει περισσότερο είναι η sp/track_uri καθώς είναι λογικό να υποθέσουμε ότι μόλις ολοκληρωθεί η αυθεντικοποίηση μπορούμε κατευθείαν να αιτηθούμε να αναπαράγουμε ένα τραγούδι (εφ'όσον ξέρουμε το αναγνωριστικό του).

Η διεπαφή χρήστη του spotify μας δίνει πρόσβαση σε ένα είδος Ενιαίου Εντοπιστή Πόρου, το οποίο είναι προσαρμοσμένο στα πλαίσια της εφαρμογής αυτής και χρησιμοποιείται για το διαμοιρασμό συνδέσμων με λίστες αναπαραγωγής, τραγούδια, καλλιτέχνες, κ.α. περιεχόμενο της εφαρμογής μεταξύ των χρηστών. Για να αποκτήσει κάποιος πρόσβαση σε ένα τέτοιο σύνδεσμο, αρκεί να κάνει μέσα στην εφαρμογή δεξί κλικ πάνω στο υλικό για το οποίο επιθυμεί να δημιουργήσει το σύνδεσμο και να πατήσει "Copy Spotify Uri".




Φυσικά αυτό το uri σε καμία περίπτωση δεν δίνει στο χρήστη πρόσβαση στον πόρο αυτόν καθαυτόν, αλλά μόνο πλοηγεί την εφαρμογή μέχρι τον πόρο αυτό. Οι σύνδεσμοι αυτοί είναι της μορφής *spotify:track:<resource_id>* οπότε ασφαλώς μπορούμε να υποθέσουμε ότι ένας χρήστης πολύ εύκολα μπορεί να μάθει το αναγνωριστικό ενός τραγουδιού.

Παρ'όλα αυτά, εκτελώντας στην εφαρμογή αναπαραγωγή τραγουδιών και παρακολουθώντας τις παραμέτρους που παρέχονται στην **sp/track_uri**, μπορεί κανείς να διακρίνει ότι οι τιμές αυτές διαφέρουν. Παραδείγματος χάρη, για ένα συγκεκριμένο τραγούδι, στο πρώτο σημείο έχουμε **spotify:track:3pH3x0sJdAZIK1ODdKuSRs** ενώ στο δεύτερο **702ff4f580e74e3d95ff4e5681e333da**.

Σύμφωνα με τη θεωρητική προσέγγιση που ακολουθούμε λοιπόν, αν αναλύσουμε και διερευνήσουμε και τον μετασχηματισμό αυτό ανάμεσα στα δύο αναγνωριστικά, θα είμαστε έτοιμοι να εφαρμόσουμε το πρωτόκολλο με σκοπό την απόκτηση πρόσβασης στα φυσικά μέσα.

Αναζητώντας λοιπόν στο αρχείο Javascript για "track_uri", φτάνουμε στο εξής σημείο του κώδικα:

```
} else q = new Spotify.Calls.Simple({
  method: "track_uri",
  payload: ["mp3160", q.id, g],
  context: k,
  persistent: !0,
  retries: 2,
  type: "track_uri"
}, function(c) {
  if (null !== c.response) {
```

Βλέπουμε ότι σαν παράμετρος στην κλήση αυτή περνιέται το id του τραγουδιού, το οποίο βρίσκεται μέσα στο αντικείμενο *q*. Πιο ψηλά στον κώδικα, στην ίδια μέθοδο, βλέπουμε πώς κατασκευάζεται το αντικείμενο *q*:

```
this.list = function(b, g, f, m) {
  var q = Spotify.Link.fromString(b);
```

Θα έχει πολύ ενδιαφέρον να δούμε τα περιεχόμενα των μεταβλητών *b* και *q*. Γι'αυτό το λόγο, τοποθετούμε ένα breakpoint αμέσως μετά από αυτή τη γραμμή και ανανεώνουμε τη σελίδα μας. Η εκτέλεση σταματάει και από την κονσόλα των DevTools βλέπουμε τις εξής τιμές για τις μεταβλητές μας:

```
> q
◂ ▼S…y.Link {type: "track", id: "702ff4f580e74e3d95ff4e5681e333da"}
    id: "702ff4f580e74e3d95ff4e5681e333da"
    type: "track"
  ▶ __proto__: Spotify.Link
> b
◂ "spotify:track:3pH3x0sJdAZIK1ODdKuSRs"
```

Που σημαίνει ότι η εκτέλεση της Spotify.Link.fromString μετασχημάτισε το ένα αναγνωριστικό σε αυτό που απαιτεί η track_uri.

Ερευνώντας την **fromString** μέθοδο καταλήγουμε στο εξής κομμάτι του κώδικα:

```
a.fromString = function(a) {
  var b = 1;
  if (0 == a.indexOf("spotify:")) a = a.slice(8).split(":"),
    b = 0;
  else if (0 == a.indexOf("http://play.spotify.com/")) a = a.slice(
    24).split("/");
  else if (0 == a.indexOf("https://play.spotify.com/")) a = a.slice(
    25).split("/");
  else if (0 == a.indexOf("http://open.spotify.com/")) a = a.slice(
    24).split("/");
  else if (0 == a.indexOf("https://open.spotify.com/")) a = a.slice(
    25).split("/");
  else throw 1;
  return h(a, b)
};
```

Το οποίο στην ουσία αφαιρεί τα κομμάτια του συνδέσμου πλην του αναγνωριστικού κι έπειτα καλεί την μέθοδο *h* η οποία είναι μια ιδιαίτερα μακροσκελής μέθοδος μεγέθους 110 γραμμών. Στη μέθοδο αυτή εκτελούνται εντολές κατα συνθήκη ανάλογα με τον τύπο του πόρου στον οποίο αναφέρεται το αναγνωριστικό, και, στην περίπτωση του τραγουδιού εκτελείται ο ακόλουθος κώδικας:

```
    case a.Type.TRACK:
      return a.trackURI(p());
```

όπου η μέθοδος *p* είναι:

```
var h = 0,
  k = function() {
    return b[h++]
  },
  p = function() {
    var a = k();
    return 22 == a.length ? c.toHex(a, 32) : a
  },
```

και η μέθοδος *trackURI*:

```
a.trackURI = function(b) {
  22 == b.length && (b = c.toHex(b, 32));
  return new a(a.Type.TRACK, {
    id: b
  })
};
```

μας επιστρέφει το αντικείμενο που είχαμε στη μεταβλητή *q*. Αντιστοίχως η *toHex* είναι:

```
fromBytes: function(a, g) {
  var c = d(a.slice(0).reverse(), 256, 62);
  return n(k(c, g),
      "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")
    .join("")
},
toBytes: function(a, g) {
  var p = d(n(a, c), 62, 256);
  return k(p, g).reverse()
},
toHex: function(a, g) {
  var p = d(n(a, c), 62, 16);
  return n(k(p, g),
      "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")
    .join("")
},
fromHex: function(b, g) {
  var c = d(n(b, a), 16, 62);
  return n(k(c, g),
      "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ")
    .join("")
}
```

Με εκτενή και προσεκτική (δυναμική αλλά και στατική) ανάλυση της ακολουθίας αυτής των κλήσεων καταλήγουμε στο συμπέρασμα ότι τα δύο αναγνωριστικά συμβολίζουν τον ίδιο αριθμό με διαφορετική κωδικοποίηση. Στην πρώτη μεν φάση είναι κωδικοποιημένος με βάση το 62 σε μια συμβολοσειρά συνολικού μεγέθους 22 χαρακτήρων, ενώ στην τελική του μορφή είναι κωδικοποιημένος με βάση το 16 σε μία συμβολοσειρά 32 χαρακτήρων. Το αλφάβητο που χρησιμοποιείται για την κωδικοποίηση των ψηφίων σε χαρακτήρες είναι και στις δύο περιπτώσεις το ίδιο, και είναι το

**"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"**.

### 6.5.7 Δεδομένα επιστροφής της sp/track_uri

Η *sp/track_uri* επιστρέφει τη διεύθυνση του φυσικού πόρου σε πρωτόκολλο RTMP. Ενώ είναι ένας άμεσα χρησιμοποιήσιμος πόρος και μπορεί να καταγραφεί στον σκληρό δίσκο του χρήστη με χρήση των κατάλληλων προγραμμάτων, η πρόσβαση στο φυσικό πόρο του μέσου σε μορφή τύπου MP3[32] μέσω HTTP πρωτοκόλλου θα ήταν ακόμα πιο χρήσιμο. Για

το λόγο αυτό, κατα την έρευνα της εργασίας αυτής έγινε πειραματισμός με τις διάφορες τιμές που μπορούν να δοθούν ως ορίσματα στην *sp/track_uri* και, συγκεκριμένα στο τελευταίο όρισμα που αναφέρεται στον τύπο του μέσου που ζητούμε.

Έτσι, πραγματοποιήθηκε η ανακάλυψη του γεγονότος ότι **αν στην κλήση αυτή ο χρήστης στείλει το λεκτικό *http* αντί για *rtmp*, τότε αποκτά πρόσβαση μέσω του HTTP πρωτοκόλλου σε πόρο τύπου mp3.**

## 6.6 Δοκιμή και χρήση του πρωτοκόλλου για αυθαίρετους σκοπούς

Έχοντας συγκεντρώσει τα βασικά δομικά χαρακτηριστικά του πρωτοκόλλου επικοινωνίας, για να ελέγξουμε κατά πόσο τα ευρήματα μας είναι αρκετά για να μπορέσει ένας κακόβουλος χρήστης να αποκτήσει πρόσβαση στα αρχεία των φυσικών πόρων, απαιτείται η κατασκευή ενός προγράμματος πελάτη που να χρησιμοποιεί το πρωτόκολλο κατά τις ανάγκες μας.

Στα πλαίσια αυτής της εργασίας χρησιμοποιήθηκε η γλώσσα Ruby για την κατασκευή ενός σεναρίου εντολών που πραγματοποιεί την αυθεντικοποίηση στο Spotify Web Player μιμούμενο ένα browser και χρησιμοποιώντας διαπιστευτήρια που παρέχει ο χρήστης. Στη συνέχεια χρησιμοποιεί την *sp/track_uri* εντολή για να λάβει την διεύθυνση του αρχείου mp3 του τραγουδιού, το οποίο και λαμβάνει ως αντίγραφο στον υπολογιστή του χρήστη.

Το αποτέλεσμα της προσπάθειας αυτής ήταν επιτυχές και το σενάριο πραγματοποιήθηκε χωρίς την παραμικρή έλλειψη. Ο κώδικας που χρησιμοποιήθηκε δίδεται στο Μέρος Γ.

```
giorgos@giorgoss-mbp Thesis $ ruby spot_on.rb my_username my_pass spotify:track:3pH3
x0sJdAZIK1ODdKuSRs
Login complete...
WebSocket connecting to wss://lon2-linkap-a5.ap.spotify.com:443..
Doing work...
Playing ping pong...
Getting your info...
Your link... http://dsu0uct5x2puz.cloudfront.net/mp3/4a0277bc20b9f5bc17ad1813f21dfe3
928f177aa.mp3?Expires=1449436242&Signature=Reel80xoAGJXUJNDkGWn6OS6JERIUzbqWBSXyEnph
uyHnUxGSYmXam8pCQkc8FGo0PFk~utN2yXiS~3UFVNR1mk8w7XwJDPz1lccc2Rf5e3ZQ2q4gedH79Iez4i0x
vI9v~qfU04RmdEA6ypKm-Ew8G52CauqDAy8GVcrV6vGEAiHMKd5bkJNVrE9-tuwDWXY~ytSeKMr9-zVyeSf9
GM5z6epuKjRsajnbzOt9DNGLIVYeLMTr2FzR39vPSM4MwHskhARsuu-5Wa0MATUZIhrbLSDOmzwQ-9UgXIPw
bhfMZq6JveHHnukX~WbEpoazvticwTXL0b9TjrfeT8DmhHt0A__&Key-Pair-Id=APKAJXKSII4ED2EOGZZA
giorgos@giorgoss-mbp Thesis $
```

Φυσικά, η χρήση που παρουσιάζεται εδώ είναι πολυ μικρού εύρους και έχει μόνο σκοπό να αποδείξει το αν είναι ευάλωτη η υπηρεσία του Spotify Web Player. Ένας πραγματικά κακόβουλος χρήστης θα μπορούσε να το επεκτείνει αρκετά (μέσω της γνώσης που μπορεί να αποκτήσει μελετώντας τα πηγαία αρχεία και την επικοινωνία) αυτοματοποιώντας τις λήψεις με βάση τις λίστες αναπαραγωγής του χρήστη, με αναζήτηση σε καλλιτέχνες και λήψη ολόκληρης της δισκογραφίας κ.λπ. μεγιστοποιώντας έτσι τον κίνδυνο και τη ζημιά για την υπηρεσία.

# 7. Δοκιμές σε υπηρεσίες αντίστοιχης φύσης

Πέρα από το Spotify Web Player, έγιναν αντίστοιχες δοκιμές παραβίασης των αρχείων και σε άλλες υπηρεσίες αντίστοιχης φύσης, όπως το Google Play και το Deezer. Το πρώτο αποδείχτηκε το πιο ελαστικό από άποψη επιπέδων προστασίας, έχοντας μόνο ενός επιπέδου Challenge-Response και προσφέροντας τα αρχεία μέσω του πρωτοκόλλου HTTP **εξ'ορισμού** (οπότε και φαίνονται οι διευθύνσεις τους **πάντα** στην καρτέλα Network των Developer Tools). Το Deezer αποδείχτηκε το πιο διασφαλισμένο, αν και φυσικά κι αυτό όχι άτρωτο, προσφέροντας κι αυτό προστασία και στο επίπεδο του αρχείου Flash, αλλά με μία σημαντική διαφορά: τα αρχεία που λαμβάνει είναι κρυπτογραφημένα και τα αποκρυπτογραφεί με βάση μυστικές ποσότητες που ενυπάρχουν στο αρχείο αυτό, κάνοντας έτσι τη διαδικασία της αντίστροφης μηχανικής αρκετά πιο χρονοβόρα και απαιτητική.

Στα πλαίσια της εργασίας αυτής υπήρξε ενδιαφέρον να γίνουν αντίστοιχες δοκιμές και σε υπηρεσίες ταινιών όπως το Netflix, αλλά λόγω έλλειψης προσβασιμότητας στις υπηρεσίες αυτές λόγω του ότι δεν ήταν διαθέσιμες στην Ελλάδα κατά τη συγραφή της εργασίας αυτής, η προσπάθεια δεν καρποφόρησε.

# 8. Τρόποι αντιμετώπισης και αντίμετρα

Το πρόβλημα που αντιμετωπίζεται στην περίπτωση που μελετάμε, από τους υπεύθυνους της ασφάλειας, ανάγεται σε ένα πολύ λεπτό ζήτημα και δεν υπάρχει σαφής ή προφανής λύση. Οι ρίζες του αγγίζουν την ίδια τη φύση των διαδικτυακών εφαρμογών, και το γεγονός ότι η Javascript γλώσσα είναι μία διερμηνευμένη[33] και όχι μεταγλωττισμένη γλώσσα[34]. Άμεση συνέπεια αυτού, είναι μια τέτοια εφαρμογή να παραδίδεται στον υπολογιστή του τελικού χρήστη σε μορφή κώδικα και όχι γλώσσας μηχανής, κάτι το οποίο εν τέλει είναι πολύ πιο αναγνώσιμο από τους ανθρώπους και άρα πρόσφορο για αντίστροφη μηχανική.

Έχοντας λοιπόν δεδομένη τη φύση του ζητήματος και την αδυναμία παροχής αντιμέτρων τα οποία θα αφοπλίζουν εντελώς κάποιον κακόβουλο χρήστη, το αντικείμενο της έρευνας μετατοπίζεται. Αυτό που πλέον μετράει, είναι οποιαδήποτε προσπάθεια αντίστοιχης παραβίασης να είναι τόσο χρονοβόρα και με μεγάλο δείκτη δυσκολίας, ώστε να αποθαρρύνει εν τέλει τον επιτιθέμενο. Ιδανικά, θα καταφέρει και να μειώσει τον αριθμό των επιτιθέμενων που θα έχουν τις δεξιότητες να την ολοκληρώσουν με επιτυχία.

Στην αναζήτηση μας για τέτοιου είδους αντίμετρα, θα εστιάσουμε στις ελλείψεις που εντοπίστηκαν στις υπηρεσίες που μελετήθηκαν, καθώς και στα σημεία τα οποία μας δυσκόλεψαν και άρα θεωρείται χρήσιμο να υπάρχουν στην γραμμή άμυνας της υπηρεσίας. Έτσι, φιλοδοξούμε να παρουσιάσουμε μία λίστα με τεχνικές προδιαγραφές ασφάλειας, κυρίως από άποψη κώδικα, τις οποίες θα μπορεί να συμβουλευτεί μια τέτοια υπηρεσία για να θωρακίσει όσο γίνεται περισσότερο την εφαρμογή της.

- Πρώτιστα **κρίνεται απαραίτητο ο κώδικας Javascript να διαμοιράζεται σε obfuscated μορφή**. Ο μετασχηματισμός αυτός δυσχαιρένει σημαντικά την ανάγνωση του κώδικα ενώ δεν επηρεάζει καθόλου την απόδοση του (ενώ σε μερικές περιπτώσεις την αυξάνει[38]).
- Δεύτερον, **η παρουσία δεδομένων αποσφαλμάτωσης (debugging log data) μέσα στον κώδικα,** ακόμα κι αν δεν εκτυπώνονται τελικά στην κονσόλα, **δίνει στοιχεία και καθοδηγεί τον κακόβουλο χρήστη στην αντίστροφη μηχανική**. Κατα πρώτο λόγο γιατί ενδέχεται να ενεργοποιήσει την εκτύπωση των δεδομένων αυτών και να έχει πλήρη ενημέρωση για τις εσωτερικές διεργασίες, όπως έγινε και στα πλαίσια της εργασίας αυτής. Αλλά και δεύτερον γιατί ακόμα κι αν δεν γίνει δυναμική ανάλυση, αυτά τα δεδομένα παρέχουν πληροφορίες κατά την στατική ανάλυση, επισημαίνοντας τη λειτουργία που επιτελούν διάφορα σημεία του κώδικα.
- Τρίτον, **οποιαδήποτε αυθαίρετη κρυπτογράφηση της επικοινωνίας με τον εξυπηρετητή**, θα προσθέσει αρκετά εμπόδια στην προσπάθεια του κακόβουλου χρήστη. Στις περιπτώσεις που ηλέγχθησαν στην εργασία αυτή, η επικοινωνία μεταδιδόταν ως απλό κείμενο, χωρίς κανένα είδος κρυπτογράφησης, και γι'αυτό ήταν διαθέσιμη για επιθεώρηση μέσα από τα Developer Tools. Θα πρέπει να σημειωθεί όμως, ότι, σε περίπτωση που υιοθετηθεί σχήμα κρυπτογράφησης της επικοινωνίας, καλό θα είναι **το κλειδί να μην ειναι σταθερό αλλά σχετικό με τη συνεδρία** και επίσης **να μην υπάρχει κάπου στον κώδικα ενσωματωμένο** αλλά να παράγεται δυναμικά κατά την εκτέλεσή του. Ένα πολύ καλό παράδειγμα για προστασία του κλειδιού κρυπτογράφησης εντοπίστηκε κατά τη μελέτη της υπηρεσίας Deezer, όπου το κλειδί παραγόταν απο το συνδυασμό των δεκαεξαδικών κωδικών χρωμάτων δύο εικόνων, **αφού είχαν εφαρμοστεί σε αυτές δυναμικά κάποια φίλτρα επεξεργασίας**.

- Τέταρτον, για να θωρακιστεί ακόμα περισσότερο η επικοινωνία με τον εξυπηρετητή, κρίνεται χρήσιμο οι απαντήσεις του τελευταίου στις αιτήσεις του προγράμματος πελάτη να περιέχουν **πολύ περιορισμένα και κωδικοποιημένα μηνύματα λάθους**. Σε μία απο τις υπηρεσίες που καλύφθηκαν στην εργασία αυτή, σε περίπτωση λάθους στην κλήση κάποιων απομακρυσμένων μεθόδων, ο εξυπηρετητής απαντούσε με πλήρη περιγραφή του λάθους. Όσο χρήσιμο είναι αυτό στους μηχανικούς λογισμικού κατά τη φάση της ανάπτυξης της υπηρεσίας, αλλό τόσο διευκολύνει και τον επιτιθέμενο στην αντίστροφη μηχανική του πρωτοκόλλου επικοινωνίας καθώς κατά τον πειραματισμό του με αυτό για να το μιμηθεί, τον καθοδηγεί προς τη σωστή χρήση.
- Πέμπτον, ένας τρόπος να “αναπληρωθεί” το κενό ασφάλειας που αφήνει η φύση της Javascript ως διερμηνευμένη γλώσσα, είναι **να ενσωματωθούν επίπεδα ασφάλειας σε λειτουργικές μονάδες φτιαγμένες σε Adobe Flash**, ή κάποια άλλη μεταγλωττισμένη γλώσσα συμβατή με το μοντέλο των διαδικτυακών εφαρμογών. Τα εκτελέσιμα αρχεία του εν λόγω συστήματος είναι μεταγλωττισμένα στο αντίστοιχο bytecode, και διαμοιράζονται με αυτό τον τρόπο, ο οποίος προσδίδει αρκετά επίπεδα πολυπλοκότητας στην αντίστροφη μηχανική του. Υπάρχουν διάφορα εργαλεία ανάλυσης προγραμμάτων swf όπως το Eltima SWF to FLA converter[35], Sothink SWF Decompiler[36], ShowMyCode[37], κ.α. αλλά το obfuscation που μπορεί να γίνει σε αυτά έχει αποδειχτεί πολλές φορές αδύνατο να αντιστραφεί αυτομάτως.
- Έκτον, όπως και με την επικοινωνία με τον εξυπηρετητή, **κρίνεται απαραίτητο τα αρχεία μέσων να είναι κρυπτογραφημένα**. Για να αντιμετωπιστεί η πρόσθετη καθυστέρηση που χρειάζεται για την αποκρυπτογράφηση, μπορούν **να αποστέλλονται τμηματικά**. Έτσι, αυξάνεται ακόμα περισσότερο η δυσκολία αντιγραφής του μέσου. Φυσικά οι κανόνες κρυπτογράφησης που θα εφαρμοστούν εδώ θα πρέπει να συμμορφώνονται με τις απαιτήσεις που θέσαμε και στην κρυπτογραφία της επικοινωνίας.
- Έβδομο, **η προσφορά του τελικού μέσου σε μία μορφή πέραν από την κοινώς γνωστή των αρχείων mp3**, θα είναι μια σημαντική προσθήκη στην αμυντική γραμμή. Αντιστοίχως με το Spotify, μπορεί να προσφέρονται τα μέσα σε ένα πρωτόκολλο συνεχούς ροής και όχι αρχείου, και το οποίο μετά το πέρας του τραγουδιού θα μεταδίδει θόρυβο (αλλά το επίσημο πρόγραμμα πελάτης θα ξέρει να σταματήσει την αναπαραγωγή), δυσχαιρένοντας έτσι την αυτόματη αποθήκευση αντιγράφου.
- Επίσης, **δεν θα πρέπει να δίνεται στο πρόγραμμα πελάτη η δυνατότητα επιλογής του τρόπου με τον οποίο θα αποκτήσει πρόσβαση στο μέσο**. Από τη στιγμή που ο εξυπηξρετητής γνωρίζει ότι το πρόγραμμα πελάτης μπορεί να αναπαράγει ένα μέσο με χρήση RTMP πρωτοκόλλου, δεν θα πρέπει να μας προσφέρει το μέσο και σε άλλες μορφές, ενόσω πειραματιζόμαστε με τη διεπαφή επικοινωνίας με τον εξυπηρετητή.
- Τέλος, όσοι μηχανισμοί ασφάλειας υιοθετηθούν και υλοποιηθούν, **κρίνεται συνετό να αλλάζουν σε τακτά χρονικά διαστήματα**. Έτσι ακόμα και αν κάποιος χρήστης καταφέρει να βάλλει την υπηρεσία, μετά από λίγο καιρό να πρέπει να αναθεωρεί την επίθεσή του, πράγμα το οποίο από μόνο του δρα ανασταλτικά.

# 9. Μελλοντική Εργασία

Η μεθοδολογία που αναπτύχθηκε στα πλαίσια της εργασίας αυτής θα χρησιμοποιηθεί για να ελεγχθεί η ασφάλεια του σχήματος διανομής προστατευόμενου περιεχομένου και από άλλες υπηρεσίες, όπως είναι η Amazon, το Apple Music, το Rdio κ.α. Ο σκοπός είναι να ομαδοποιηθούν κοινές αδυναμίες με σκοπό την δημιουργία ενός μοντέλου αντιμετώπισης τέτοιων απειλών.

Μιά τέτοια έρευνα μπορεί να επικεντρωθεί είτε στην ανάπτυξη ενός σκελετού (framework) υψηλού επιπέδου ασφάλειας προς υιοθεσία από τις εκάστοτε υπηρεσίες, είτε στην ανάπτυξη μιας προτυποποιημένης πρότασης υλοποίησης, η οποία θα καλύπτει μεγάλο εύρος απειλών αλλά και χρηστικών αναγκών των υπηρεσιών.

Έχοντας λοιπόν δεδομένη τη φύση του ζητήματος και την αδυναμία παροχής αντιμέτρων τα οποία θα αφοπλίζουν εντελώς κάποιον κακόβουλο χρήστη, το αντικείμενο της έρευνας μετατοπίζεται. Αυτό που πλέον μετράει, είναι οποιαδήποτε προσπάθεια αντίστοιχης παραβίασης να είναι τόσο χρονοβόρα και με μεγάλο δείκτη δυσκολίας, ώστε να αποθαρρύνει εν τέλει τον επιτιθέμενο. Ιδανικά, θα καταφέρει και να μειώσει τον αριθμό των επιτιθέμενων που θα έχουν τις δεξιότητες να την ολοκληρώσουν με επιτυχία.

# 10. Σύνοψη

Η πρόοδος της τεχνολογίας και η διείσδυση της στην καθημερινή ζωή των ανθρώπων, έχει μετατοπίσει την πολυμεσική αγορά και έχει εισαγάγει ένα νέο μοντέλο, το συνδρομητικό. Η προτίμηση που έχουν δείξει οι χρήστες στη δυνατότητα να έχουν αμέτρητο αριθμό παραγωγών κατά βούληση (on demand) μέσω συνεχούς ροής έχει καταστήσει το μοντέλο αυτό απαραίτητο και δεν νοείται εκσυγχρονισμένη υπηρεσία προσφοράς πολυμέσων που να μην το προσφέρει.

Παρ'όλα αυτά σε αυτό το μοντέλο ελλοχεύει και ένας κίνδυνος που πιο παλιά δεν υπήρχε σε τόσο μεγάλο βαθμό, ο κίνδυνος της παράνομης διανομής υλικού. Μέσω των συνδρομητικών υπηρεσιών τα μέσα εκτίθενται σε πολύ μεγαλύτερο εύρος χρηστών, καθώς κάθε ένας έχει οποιαδήποτε στιγμή εν δυνάμει πρόσβαση σε όλα τα μέσα που προσφέρει μια υπηρεσία. Συνεπώς και ο κίνδυνος κακόβουλης χρήσης πολλαπλασιάζεται.

Δεν είναι μόνο αυτό το πρόβλημα όμως, καθώς η ίδια η φύση αυτού του μοντέλου προσφοράς περιεχομένου εμπεριέχει κινδύνους και ευπάθειες που πηγάζουν από τη φύση των διαδικτυακών εφαρμογών, και τη βασική αρχιτεκτονική τους. Όπως αναφέραμε, προφανής και σίγουρη λύση ασφάλειας δεν υπάρχει και δεν μπορεί να προταθεί σύνολο αντιμέτρων που να θωρακίζει απόλυτα το σύστημα.

Παρ'όλα αυτά, εαν μια υπηρεσία υιοθετήσει πρακτικές μεγιστοποίησης των γραμμών άμυνας με στόχο τη δυσχέρανση των κακόβουλων ενεργειών και εν τέλει την αποθάρρυνση των παράνομων χρηστών, και παράλληλα εναλλάσει τους μηχανισμούς αυτούς συχνά, μπορεί να επιτύχει μεγάλη μείωση στις προσβολές του περιεχομένου της και εν δυνάμει να τις εκμηδενίσει.

Στα πλαίσια της εργασίας αυτής προτάθηκαν τρόποι ισχυροποίησης της άμυνας και βέλτιστες πρακτικές αντιμετώπισης του κινδύνου αυτού.

# 11. Βιβλιογραφία


[1] O'Reilly, Tim. "What Is Web 2.0." *O'Reilly*. N.p., 30 Sept. 2005. Web. 1 Dec. 2015. <http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html?page=1>.

[2] "Spotify" <https://www.spotify.com/>.

[3] "Deezer" <http://www.deezer.com/>.

[4] "Google Play Music" <https://play.google.com/music/>.

[5] "Rdio" <http://www.rdio.com/>.

[6] "Apple Music" <http://www.apple.com/music/>.

[7] "Amazon Prime Music" <http://www.amazon.com/b?node=8335758011>.

[8] "Netflix" <https://www.netflix.com/>.

[9] "Hulu" <http://www.hulu.com/>.

[10] "Fiddler" <www.telerik.com/fiddler>.

[11] "FiddlerScript Editor" <http://www.telerik.com/download/fiddler/fiddlerscript-editor>.

[12] "Usage share of web browsers." Wikipedia. Wikimedia Foundation, 3 Dec. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/Usage_share_of_web_browsers>.

[13] "Google Chrome" <http://www.google.com/intl/el_gr/chrome/browser/desktop/index.html>.

[14] "Mozilla Firefox" <https://www.mozilla.org/el/firefox/new/>.

[15] "Internet Explorer" <http://windows.microsoft.com/el-gr/internet-explorer/download-ie>.

[16] "Microsoft Edge" <https://www.microsoft.com/en-us/windows/microsoft-edge>.

[17] "Safari" <http://www.apple.com/safari/>.

[18] "Opera" <http://www.opera.com/el>.

[19] "Simple HTTP File Server" <http://www.rejetto.com/hfs/>.

[20] "AJAX" Wikipedia. Wikimedia Foundation, 29 Nov. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/Ajax_(programming)>.

[21] “WebSocket” Wikipedia. Wikimedia Foundation, 18 Nov. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/WebSocket>.

[22] “JSON” Wikipedia. Wikimedia Foundation, 3 Dec. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/JSON>.

[23] “Transmission Control Protocol # Selective Acknowledgments” Wikipedia. Wikimedia Foundation, 29 Nov. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/Transmission_Control_Protocol#Selective_acknowledgments>.

[24] “Real Time Messaging Protocol” Wikipedia. Wikimedia Foundation, 21 Nov. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/Real_Time_Messaging_Protocol>.

[25] “Javascript Beautifier And Formatter Online” <http://www.danstools.com/javascript-beautify/>.

[26] “Protocol Buffers” Wikipedia. Wikimedia Foundation, 11 Nov. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/Protocol_Buffers>.

[27] “Challenge–response authentication” Wikipedia. Wikimedia Foundation, 20 Oct. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/Challenge%E2%80%93response_authentication>.

[28] “HTTP cookie” Wikipedia. Wikimedia Foundation. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/HTTP_cookie>.

[29] “Web Storage # localStorage” Wikipedia. Wikimedia Foundation, 17 Aug. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/Web_storage#localStorage>.

[30] “Eval # JavaScript” Wikipedia. Wikimedia Foundation, 10 Sep. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/Eval#JavaScript>.

[31] “Adobe Flash” Wikipedia. Wikimedia Foundation, 18 Nov. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/Adobe_Flash>.

[32] “MP3” Wikipedia. Wikimedia Foundation, 2 Dec. 2015. Web. 3 Dec. 2015. <https://en.wikipedia.org/wiki/MP3>.

[33] “Διερμηνευμένη γλώσσα” Wikipedia. Wikimedia Foundation, 20 Apr. 2013. Web. 3 Dec. 2015. <https://el.wikipedia.org/wiki/%CE%94%CE%B9%CE%B5%CF%81%CE%BC%CE%B7%CE%BD%CE%B5%CF%85%CE%BC%CE%AD%CE%BD%CE%B7_%CE%B3%CE%BB%CF%8E%CF%83%CF%83%CE%B1>.

[34] “Μεταγλωττιστής” Wikipedia. Wikimedia Foundation, 23 Jul. 2015. Web. 3 Dec. 2015. <https://el.wikipedia.org/wiki/%CE%9C%CE%B5%CF%84%CE%B1%CE%B3%CE%BB%CF%89%CF%84%CF%84%CE%B9%CF%83%CF%84%CE%AE%CF%82>.

[35] “Eltima SWF to FLA converter” <http://www.eltima.com/products/flashdecompiler/>.

[36] “Sothink SWF Decompiler” <http://download.cnet.com/Sothink-SWF-Decompiler/3000-6676_4-10073491.html>.

[37] “ShowMyCode” <http://www.showmycode.com/>.

[38] aditya, “Does minified javascript improve performance?”, StackOverflow Web. 3 Dec. 2015. <http://stackoverflow.com/a/1181461>.

# Μέρος Γ

# Παράρτημα Προγραμματιστικής Υλοποίησης

## Πρόγραμμα πελάτη για σύνδεση στο Spotify και λήψη τραγουδιών

Το πρόγραμμα πελάτη που κατασκευάστηκε στα πλαίσια αυτής της εργασίας είναι γραμμένο σε Ruby και κάνει χρήση των εξής Gems:

- phantomjs
- websocket-eventmachine-client
- selenium-webdriver

και απαιτείται να υπάρχει προεγκατεστημένο το ChromeDriver:

https://sites.google.com/a/chromium.org/chromedriver/

Η κλήση του πραγματοποιείται με τον εξής τρόπο:

**ruby spotify.rb <username> <password> <track_link>**

```
require 'phantomjs'
require 'http'
require 'json'
require 'websocket-eventmachine-client'
require "selenium-webdriver"

$settings = nil
$ws_id = 0
$ws = nil
$cbs = {}
$username = ARGV[0]
$password = ARGV[1]
$track_link = ARGV[2]
$hash = {'User-Agent' => 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1) '+
    'AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.73 Safari/537.36'}

def create_request
    HTTP.headers($hash)
end

def make_request(url)
    create_request().get(url)
end

def make_post_request(url, data)
    create_request().post(url, :form => data)
end

def login_web(username, password)
    res = make_request 'https://play.spotify.com'
    csrf_token = /"csrftoken":"([a-z0-9]*?)"/.match(res.body).captures[0]
    tracking_id = /"trackingId":"([a-z0-9]*?)"/.match(res.body).captures[0]
```

```
        vars = {
            :type => "sp",
            :username => username,
            :password => password,
            :secret => csrf_token,
            :trackingId => tracking_id,
            :referrer => "",
            :landingURL =>"play.spotify.com/",
            :cf => "",
            :f =>"login",
            :s =>"direct",
            :landingURL =>"play.spotify.com/",
        }

        JSON.parse(make_post_request("https://play.spotify.com/xhr/json/auth.php",
vars))["config"]
    end

    def get_ap
        url = "http://" + $settings["aps"]["resolver"]["hostname"].to_s + "?client=24:0:0:" +
$settings["version"].to_s
        req = make_request url
        ap = JSON.parse(req)["ap_list"][0]
        return "wss://#{ap}"
    end

    def send_ws(name, args, callback = nil)
        msg = {
                :name => name,
                :id => $ws_id,
                :args => args
        }.to_json
        $ws.send(msg)
        if callback != nil
                $cbs[$ws_id.to_s] = callback
        end
        $ws_id+=1
    end

    def send_connect
        creds = $settings["credentials"][0].split(':')
        send_ws("connect", [creds[0], creds[1], creds[2..-1].join(':')])
    end

    def do_work work
```

```
    work = work.gsub('this.reply', 'console.log')
    work += "phantom.exit();"
    res = Phantomjs.inline(work).gsub("\n","").to_i
    send_ws "sp/work_done", [res]
end

def ping_flash2 ping
    driver = Selenium::WebDriver.for :chrome
    driver.navigate.to "http://localhost:8000/player.html"
    pong = driver.execute_script("return spotifyPlayer.sp_run('#{ping}')");
    driver.quit
    send_ws "sp/pong_flash2", [pong.split("-").join(" ")]
end

$track_uri = Proc.new do
    send_ws "sp/track_uri", ["mp3160", $track_link, "http"], $show_track_uri
end

$show_track_uri = Proc.new do |o|
    puts "Your link... #{o["result"]["uri"]}"
    exit
end

def init_session
    send_ws "sp/log", [41, 1, 0, 0, 0, 0]
    send_ws "sp/log", [41, 1, 0, 0, 0, 0]
    send_ws "sp/user_info", [], $track_uri
end

$settings = login_web($username, $password)
puts "Login complete..."

ap = get_ap
puts "WebSocket connecting to #{ap}.."

EM.run do
    $ws = WebSocket::EventMachine::Client.connect(
            :uri => ap,
            :headers => $hash
    )
    $ws.onmessage do |msg|
            j = JSON.parse(msg)
            if j.has_key? "id"
                    if $cbs.has_key? j["id"].to_s
                            $cbs[j["id"].to_s].call(j)
                    else
```

```
                end
            elsif j.has_key? "message"
                if j["message"][0] == "do_work"
                    puts "Doing work..."
                    do_work j["message"][1]
                elsif j["message"][0] == "ping_flash2"
                    puts "Playing ping pong..."
                    ping_flash2 j["message"][1]
                elsif j["message"][0] == "login_complete"
                  puts "Getting your info..."
                  init_session
                end
            end
        end

        $ws.onopen do
          send_connect
        end
    end
```

Επιπροσθέτως, χρειάζεται και το παρακάτω αρχείο να σερβίρεται από κάποιο τοπικο Web Server:

```
<!DOCTYPE html>
<html>
<head>
  <title>Spotify Ping Pong</title>
  <script>
    var spotifyPlayer;
    window.onload = function () {
      spotifyPlayer = document.getElementById('spotifyPlayer');
    };
  </script>
</head>
<body>
<embed id="spotifyPlayer" src="player.swf" type="application/x-shockwave-flash"></embed>
</body>
</html>
```

Και το αρχείο player.swf από την υπηρεσία του Spotify.

Τέλος, πρέπει να εκτελείται το PhantomJs σε περιβάλλον webdriver:

**phantomjs --webdriver=9134**