Πολυτεχνειο Κρητης
Τμημα Ηλεκτρολογων Μηχανικων και
Μηχανικων Υπολογιστων

Διπλωματικη Εργασια

Εύρεση και Αντιμετώπιση ευάλωτου κώδικα για επιθέσεις Buffer Overflow και Format String

Κρικοριάν Αρμάν

Επιβλέπων Καθηγητής
Καθ.Δ.Πνευματικάτος

15 Δεκεμβρίου 2017

Περίληψη

Στην σημερινή εποχή η ασφάλεια των πληροφοριακών συστημάτων είναι μείζονος σημασίας. Πάρα πολλά συστήματα ανεξαρτήτως χρήσης διαχειρίζονται καθημερινά εκατομμύρια δεδομένα. Κάποια από αυτά τα δεδομένα μπορεί να μην είναι τόσο κρίσιμα. Κάποια άλλα όμως μπορεί να είναι και άρα η προστασία τους είναι επιτακτική ανάγκη. Ένα σύστημα όσο πιο ασφαλώς έχει προγραμματιστεί τόσο πιο μικρές είναι οι πιθανότητες να παραβιασθεί από κάποιον κακόβουλο χρήστη και άρα τόσο πιο ασφαλή είναι τα δεδομένα και η λειτουργία του από αυτόν. Όσο περνάνε τα χρόνια και η τεχνολογία εξελίσσεται τόσο πιο αποτελεσματικές γίνονται οι μέθοδοι προστασίας των πληροφοριακών συστημάτων. Δυστυχώς μέχρι και σήμερα όλα τα συστήματα παρουσιάζουν κάποιες αδυναμίες, οι οποίες βέβαια όσο πιο καλά είναι κρυμμένες ή όσο πιο μικρές είναι τόσο πιο δύσκολο είναι για έναν κακόβουλο χρήστη να τις εντοπίσει και να τις εκμεταλλευτεί προς όφελος του. Για να θεωρείται ένα σύστημα προστασίας καλό, πρέπει αφενός να παρέχει την μέγιστη δυνατή ασφάλεια και αφετέρου να είναι γρήγορο και αποτελεσματικό. Με τον όρο ¨μέγιστη δυνατή ασφάλεια' προφανώς εννοείται το σύστημα ασφαλείας να είναι ικανό να εντοπίσει και να αντιμετωπίσει όλες τις πιθανές επιθέσεις πράγμα που εν γένει θεωρείται πολύ δύσκολο. Με τους όρους ¨γρήγορο και αποτελεσματικό' εννοείται ότι το σύστημα ασφαλείας πρέπει όλα τα παραπάνω να τα πραγματοποιεί σε έναν σύντομο χρονικό διάστημα και χωρίς αστοχίες. Λαμβάνοντας υπόψιν όλα τα παραπάνω η συγκεκριμένη διπλωματική εργασία αποτελεί μια υλοποίηση ενός τέτοιου συστήματος το οποίο είναι υλοποιημένο στην γλώσσα προγραμματισμού Java και εντοπίζει και τροποποιεί, με σκοπό να προστατέψει, ευάλωτους κώδικες σε επιθέσεις Buffer Overflow και Format String Attacks, από αυτές.

Ευχαριστίες

Κατά την διάρκεια των σπουδών μου υπήρχαν πολλοί άνθρωποι που με στήριξαν και με βοήθησαν.

Πρωτίστως θα ήθελα να ευχαριστήσω την οικογένεια μου που καθ΄ όλη την διάρκεια των σπουδών μου με στήριξε και αποτελούσε τον οδηγό μου για κάθε απόφαση που έπρεπε να πάρω καθώς και για κάθε βήμα που έκανα.

Επίσης θα ήθελα να ευχαριστήσω όλους τους καθηγητές μου που μου δώσανε όλα τα εφόδια και μου μεταλαμπάδευσαν όλες τις γνώσεις τους κατά την διάρκεια των ακαδημαϊκών μου σπουδών.

Τέλος, θα ήθελα να ευχαριστήσω τον επιτηρητή μου, τον Καθηγητή Δ.Πνευματικάτο για την καθοδήγηση και τις χρήσιμες συμβουλές που μου προσέφερε κατά την διάρκεια της υλοποίησης της διπλωματικής μου εργασίας, καθώς και τους καθηγητές Γ.Παπαευσταθίου και Β.Σαμολαδά που δέχτηκαν να είναι μέλη της επιτροπής αξιολόγησης της εργασίας αυτής.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Η ασφάλεια των πληροφοριακών συστημάτων είναι ένα ζήτημα το οποίο απασχολεί τον κόσμο μας σε καθημερινή βάση. Το γεγονός αυτό δεν είναι τυχαίο προφανώς, καθώς η προστασία δεδομένων ή και ολόκληρων συστημάτων είναι ένα θέμα μεγάλης σημασίας. Στην εποχή που ζούμε, τα περισσότερα αγαθά που έχουμε, όπως τα χρήματα μας σε τραπεζικούς λογαριασμούς, οι αγορές που κάνουμε από ηλεκτρονικά καταστήματα, οι προσωπικές πληροφορίες μας σε μέσα κοινωνικής δικτύωσης, αλλά και τα ευαίσθητα δεδομένα που διαχειρίζονται εταιρίες και αποτελούν απαραίτητα στοιχεία για την ομαλή λειτουργία τους, βρίσκονται σε ηλεκτρονική - ψηφιακή μορφή, γεγονός το οποίο καθιστά την προστασία τους πολύ σημαντική.

Η παραβίαση της ακεραιότητας της ασφάλειας ενός συστήματος προέρχεται από κάποιον εξωτερικό παράγοντα. Πιο συγκεκριμένα, από κάποιο δεδομένο ή κάποιο κομμάτι κώδικα το οποίο εισέρχεται σε ένα πληροφοριακό σύστημα με σκοπό να εντοπίσει ή να εκμεταλλευτεί μία ή και περισσότερες αδυναμίες του. Ο τελικός σκοπός αυτής της κίνησης είναι είτε να υποκλαπούν κάποιες πληροφορίες είτε να τροποποιηθεί η λειτουργία του συστήματος προς όφελος του επιτιθέμενου είτε ακόμα και να σταματήσει η λειτουργία του (crash).

Ένας μεγάλος αριθμός συστημάτων που παρέχουν ασφάλεια σε κάποιο ή κάποια πληροφοριακά συστήματα έχουν ένα βασικό ελάττωμα. Αυτό το ελάττωμα είναι ότι πολλές φορές η προσπάθεια εντοπισμού όλων των πιθανών κακόβουλων δεδομένων ή τμημάτων κώδικα δεν επιτυγχάνεται με αποτέλεσμα

κάποια από αυτά να μην εντοπίζονται και εν τέλει να δημιουργείται πρόβλημα. Για αυτόν το λόγο είναι και αναγκαία η διαρκής ενημέρωση και συντήρηση των συστημάτων ασφαλείας.

Στην παρούσα εργασία γίνεται μια προσομοίωση ενός τέτοιου συστήματος το οποίο δέχεται σαν εισόδους ορισμένα προγράμματα στην γλώσσα προγραμματισμού C, τα ελέγχει και αν εμφανίζουν κάποιο πρόβλημα ασφάλειας και θεωρούνται επικίνδυνα είτε τα σταματάει είτε τα διορθώνει και τα μετατρέπει σε ασφαλή.

1.1 Στόχος και Συνεισφορά της Διπλωματικής Εργασίας

Η παρούσα Διπλωματική Εργασία διερευνά την διαδικασία απαλλαγής ενός κώδικα από ευάλωτες εντολές οι οποίες τον καθιστούν και αυτόν ευάλωτο στις επιθέσεις Buffer Overflows και Format String Attacks. Η προσπάθεια αυτή γίνεται όσο το δυνατόν πιο ρεαλιστική με την αντιμετώπιση εντολών που θα είχαν όντως αρνητικές επιπτώσεις για τον υπολογιστή. Αυτό σημαίνει ότι πέραν των εντολών ελέγχονται και τα δεδομένα που χρησιμοποιούν για να βγει ένα πιο ακριβές συμπέρασμα για την επικινδυνότητα τους. Η συγκεκριμένη προσπάθεια γίνεται με την χρήση διαφορετικών μεθόδων αντιμετώπισης όπου και αξιολογούνται βάσει απόδοσης, αποτελεσματικότητας και ταχύτητας πραγματοποίησης. Εν τέλει, στόχος αυτής της Διπλωματικής Εργασίας είναι, από έναν κώδικα ευάλωτο στις δύο προαναφερθείσες επιθέσεις, να παράξει έναν κώδικα ασφαλή.

Η συνεισφορά αυτής της Διπλωματικής Εργασίας συνοπτικά είναι η εξής:

1. Αναλύει τις επιθέσεις Buffer Overflows και Format String Attacks, τους τρόπους εκτέλεσης τους καθώς και τις επιπτώσεις τους.

2. Παρουσιάζει μια μέθοδο αντιμετώπισης ευάλωτου κώδικα στις επιθέσεις Buffer Overflows και Format String Attacks.

3. Για την πραγματοποίηση αυτού του σκοπού χρησιμοποιούνται διαφορετικές μέθοδοι (που θα αναλυθούν στο Κεφάλαιο 5) αντιμετώπισης μελετώντας έτσι τις δυνατότητες της κάθε μιας από αυτές και καταγράφοντας τα πλεονεκτήματα και τα μειονεκτήματα αυτών σε θέματα απόδοσης αποτελεσματικότητας και ταχύτητας.

4. Κατά την διάρκεια της αντιμετώπισης του ευάλωτου κώδικα, ο μηχανισμός επιχειρεί να είναι όσο το δυνατόν πιο ακριβής. Αυτό σημαίνει ότι δεν ελέγχονται μόνο οι τυχόν ευάλωτες εντολές αλλά και η επικινδυνότητα τους σε συνδυασμό με τα δεδομένα που διαχειρίζονται. Με αυτόν τον τρόπο συμπεραίνουμε πότε και υπό ποιες προϋποθέσεις μπορεί μια εντολή από ευάλωτη να γίνει και επικίνδυνη.

1.2 Οργάνωση Διπλωματικής Εργασίας

Η παρούσα Διπλωματική Εργασία είναι οργανωμένη με τον εξής τρόπο:

1. Το Κεφάλαιο 1 αποτελεί μια εισαγωγή στο θέμα της εργασίας.

2. Το Κεφάλαιο 2 περιέχει μια ιστορική αναδρομή σχετικά με τις δύο επιθέσεις που πραγματεύεται η Διπλωματική Εργασία αυτή, καθώς και παρουσίαση των πιο γνωστών επιθέσεων στον σύγχρονο κόσμο.

3. Το Κεφάλαιο 3 αποτελεί μια παρουσίαση του πρώτου είδους επίθεσης,Buffer Overflow.

4. Το Κεφάλαιο 4 αποτελεί μια παρουσίαση του δεύτερου είδους επίθεσης,Format String Attack.

5. Το Κεφάλαιο 5 αποτελεί μια λεπτομερή περιγραφή του μηχανισμού που αναπτύχθηκε για τις ανάγκες της παρούσας Διπλωματικής Εργασίας.

6. Το Κεφάλαιο 6 περιέχει μια σύνοψη και τα συμπεράσματα που προκύπτουν από την παρούσα Διπλωματική Εργασία.

Κεφάλαιο 2

Διαδεδομένες Επιθέσεις και η Σημασία της Ασφάλειας

Σε αυτό το κεφάλαιο θα αναφερθούμε εκτενώς σε διάφορες επιθέσεις που συμβαίνουν στον κόσμο μας διαρκώς. Πιο συγκεκριμένα θα γίνει λεπτομερής περιγραφή τους καθώς και παρουσίαση του τρόπου ή των τρόπων αντιμετώπισης τους.

2.1 Περιγραφή

Ο κόσμος μας, όσο περνάνε τα χρόνια βασίζεται όλο και περισσότερο στα ηλεκτρονικά και τα ψηφιακά μέσα. Είναι αλήθεια πως πλέον η ζωή όλων μας στηρίζεται στο διαδίκτυο καθώς και στις διάφορες διευκολύνσεις που αυτό προσφέρει. Όπως και στον φυσικό κόσμο έτσι λοιπόν και στον ψηφιακό υπήρξαν και θα συνεχίσουν να υπάρχουν προβλήματα ασφάλειας.

Στις μέρες μας πλέον υπάρχουν πάρα πολλοί τρόποι επίθεσης σε ψηφιακά δεδομένα. Οι δύο βασικές κατηγορίες επιθέσεων είναι οι επιθέσεις υψηλού και χαμηλού επιπέδου. Οι επιθέσεις χαμηλού επιπέδου όπως οι Buffer Overflow και Format String Attacks έχουν σαν στόχο την δημιουργία προβλήματος σε κάποιο υπολογιστικό σύστημα σε επίπεδο μνήμης (απώλεια-τροποποίηση δεδομένων μέχρι και εισαγωγή κακόβουλου κώδικα στην μνήμη). Οι επιθέσεις υψη-

λού επιπέδου είναι αυτές που εκτελούνται σε γλώσσες υψηλού επιπέδου όπως η Java και η SQL και αντίστοιχα οι χαμηλού επιπέδου αυτές που εκτελούνται σε γλώσσες χαμηλού επιπέδου όπως η C.

Οι πιο διαδεδομένες επιθέσεις υψηλού επιπέδου την εποχή αυτή είναι το SQL Injection , το Cross-Site Scripting, το Phising και οι επιθέσεις Brute Force.

2.2 Επιθέσεις Χαμηλού Επιπέδου - Buffer Overflow και Format String Attacks

Οι επιθέσεις με τις οποίες ασχολείται εκτενώς η συγκεκριμένη Διπλωματική Εργασία είναι το Buffer Overflow και τα Format String Attacks. Και οι δύο αυτές επιθέσεις είναι χαμηλού επιπέδου. Με αυτόν τον όρο εννοούμε επιθέσεις που πραγματοποιούνται σε γλώσσες χαμηλού επιπέδου ή ακόμα και σε υψηλού όταν είναι γνωστό ότι ο κώδικας στον οποίο είναι γραμμένες, θα μεταφραστεί σε γλώσσα χαμηλότερου επιπέδου,και έχουν ως στόχο την τροποποίηση,διαγραφή,προσπέλαση και εμφάνιση δεδομένων που βρίσκονται στην μνήμη ενός υπολογιστικού συστήματος.

2.2.1 Ιστορική Αναδρομή

Η πρώτη καταγεγραμμένη επίθεση Buffer Overflow ήταν το 1988. Αποτελούσε μια από τις πολλές αδυναμίες που εκμεταλλεύτηκε το Morris Worm για να μεταδοθεί μέσω του Διαδικτύου. Το πρόγραμμα που εκμεταλλεύτηκε το Morris Worm ήταν μια υπηρεσία στο Unix με την ονομασία finger. Αργότερα, το 1995, ο Thomas Lopatic ανακάλυψε εκ νέου την επίθεση Buffer Overflow και δημοσιοποίησε τα ευρήματα του στην λίστα ασφαλείας Bugtraq, που ήταν μια λίστα μηνυμάτων σχετικά με την ασφάλεια των υπολογιστικών συστημάτων. Ένα χρόνο αργότερα ο Elias Levy δημοσιοποίησε στο περιοδικό Phrack μια βήμα προς βήμα εισαγωγή για το τρόπο με τον οποίο μπορεί κανείς να εκμεταλλευτεί αδυναμίες στην επίθεση Stack based Buffer Overflow. Από

τότε μόνο 2 διαδικτυακά worms έχουν εκμεταλλευτεί και έχουν εκθέσει έναν μεγάλο αριθμό συστημάτων με αδυναμίες στο Buffer Overflow. Το 2001, το Code Red worm εξέθεσε μια αδυναμία στο Buffer Overflow στο Microsoft's Internet Information Services και το 2003 το SQL Slammer worm 'κατέλαβε' συστήματα που 'τρέχανε' το Microsoft SQL Server 2000. Τέλος, το 2003 το Buffer Overflow εμφανίζεται στις κονσόλες XBOX,Wii και PlayStation 2 με σκοπό να επιτρέπει σε λογισμικά χωρίς άδεια, να εκτελούνται στην εκάστοτε κονσόλα χωρίς κάποια αλλαγή στο υλισμικό.

Όσον αφορά τα Format String Attacks, παλαιότερα οι αδυναμίες σε αυτήν την επίθεση θεωρούνταν αμελητέες και δεν δινόταν καμία σημασία. Το γεγονός αυτό οδήγησε πολλά κοινά προγράμματα και εργαλεία να έχουν στον κώδικα τους τέτοιου είδους αδυναμίες. Ανάμεσα στις χρονολογίες 2001 και 2006 η αδυναμία απέναντι στα Format String Attacks αποτελούσε την ένατη πιο αναφερθείσα αδυναμία. Τα πρώτα format bugs (αδυναμία στις επιθέσεις Format String), παρατηρήθηκαν το 1989 κατά την διάρκεια του fuzz testing (αυτόματη τεχνική ελέγχου που περιλαμβάνει την εισαγωγή λανθασμένων,τυχαίων ή και μη προβλέψιμων δεδομένων εισόδου σε ένα πρόγραμμα με σκοπό να παρατηρηθεί η απόκριση του) που πραγματοποιούταν στο πανεπιστήμιο του Wisconsin. Η χρήση των format bugs σαν μέσο εξάπλωσης μιας επίθεσης στο σύστημα παρατηρήθηκε από τον Tymm Twillman κατά την διάρκεια ενός ελέγχου ασφαλείας στο ProFTPD (File Transfer Protocol σερερ συμβατό σε διάφορα συστήματα Unix και Windows. Ο έλεγχος αυτός έδειξε ότι συγκεκριμένα ορίσματα σε συναρτήσεις παρόμοιες με την prinft μπορούσαν να καταλήξουν μέχρι σε 'παράνομη' πρόσβαση σε δεδομένα που είναι προστατευμένα από τον χρήστη ή από το σύστημα. Το συμπέρασμα αυτό οδήγησε στην δημοσιοποίηση αυτών των αδυναμιών στην λίστα ασφαλείας Bugtraq τον Σεπτέμβριο του 1999. Παρ ΄ όλα αυτά η επιστημονική κοινότητα αντιλήφθηκε τον κίνδυνο που εγκυμονούσαν αυτού του είδους οι αδυναμίες αρκετούς μήνες μετά, όταν και βγήκαν στην επιφάνεια και άλλα λογισμικά που παρουσίαζαν αυτές τις αδυναμίες προς τα Format String Attacks.

2.3 Επιθέσεις Υψηλού Επιπέδου

Στην τωρινή εποχή οι επιθέσεις μείζονος σημασίας είναι οι υψηλού επιπέδου που έχουν σαν κύριο στόχο διαδικτυακά δεδομένα και εφαρμογές. Μπορεί,λοιπόν, οι επιθέσεις υψηλού επιπέδου να μην αποτελούν το κύριο θέμα που πραγματεύεται η συγκεκριμένη Διπλωματική Εργασία αλλά αξίζει να σημειωθούν και να γίνει μια σύντομη περιγραφή ορισμένων εξ ΄ αυτών.

2.3.1 SQL Injections

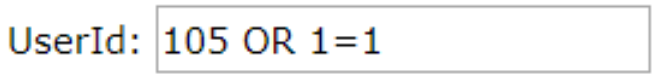
Μια επίθεση SQL Injection είναι στην ουσία το 'πέρασμα' ενός κακόβουλου SQL Query μέσω μιας φόρμας εισαγωγής δεδομένων που σου παρέχεται από την ίδια την εφαρμογή. Μια τέτοια φόρμα θα μπορούσε να είναι ένα login και ένα password. Μια επιτυχημένη τέτοια επίθεση μπορεί να προκαλέσει διαφόρων ειδών προβλήματα, όπως το διάβασμα ευαίσθητων δεδομένων από τον επιτιθέμενο, τροποποίηση των δεδομένων αυτών (αλλαγή/διαγραφή), εκτέλεση ενεργειών του διαχειριστή (π.χ απενεργοποίηση της βάσης δεδομένων).

Περιγραφή Επίθεσης

Όπως αναφέρθηκε και παραπάνω μια επίθεση SQL Injection είναι 'πέρασμα' ενός κακόβουλου SQL Query μέσα στην βάση με σκοπό να προκαλέσει κάποιου είδους δυσλειτουργία σε αυτήν. Για να γίνει καλύτερα αντιληπτή η συγκεκριμένη επίθεση ακολουθεί ένα παράδειγμα.

Έστω ότι έχουμε μια βάση δεδομένων που σε έναν πίνακα της κρατάει κάποια δεδομένα κάποιων χρηστών. Ας υποθέσουμε επίσης ότι έχουμε μια εφαρμογή στην οποία απαιτείται να κάνει κάποιος login για να αλληλεπιδράσει με το σύστημα. Κατά την διαδικασία του login τα δεδομένα που εισάγει ο χρήστης συμμετέχουν σε ένα SQL Query το οποίο θα εκτελεστεί με σκοπό την εξακρίβωση των στοιχείων που έχει εισάγει ο χρήστης (user authentication). Θα ελεγχθεί δηλαδή εάν υπάρχει ο χρήστης με το συγκεκριμένο username και το

συγκεκριμένο password. Έστω όμως ότι ο χρήστης είναι κάποιος ο οποίος προσπαθεί να πραγματοποιήσει μια επίθεση τύπου SQL Injection. Σε αυτήν την περίπτωση εισάγει στο πεδίο του username μια διαφορετική συμβολοσειρά.



Σχήμα 2.1: Συμβολοσειρά που εισάγει ο κακόβουλος χρήστης.

Το SQL Query το οποίο αναζητάει τον χρήστη αυτόν είναι το εξής.

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

Σχήμα 2.2: Το SQL Query το οποίο αναζητάει τον χρήστη .

Το SQL Query αυτό είναι προφανώς πάντα έγκυρο αφού το 1=1 είναι πάντα μια αληθής πρόταση. Άρα αυτό που θα επιστραφεί από την βάση είναι όλες οι γραμμές/εγγραφές από τον πίνακα αυτόν. Εάν αυτός ο πίνακας όμως έχει μέσα και κωδικούς, τότε ο επιτιθέμενος έχει στα χέρια του όλα τα στοιχεία κάθε χρήστη και μπορεί να εισέλθει χωρίς κανένα πρόβλημα στο σύστημα.

Τρόποι Αντιμετώπισης

Οι βασικοί τρόποι με τους οποίους αντιμετωπίζεται η συγκεκριμένη επίθεση είναι 3.

1. **Parameterized Statement**: Αποτελεί μια μέθοδο κατά την οποία στην διάρκεια προγραμματισμού μιας βάσης δεδομένων δημιουργούνται πρότυπα SQL Query τα οποία μένουν απαράλλαχτα και απλά δέχονται τα δεδομένα εισόδου στην κατάλληλη θέση. Η μέθοδος αυτή αυξάνει την αποτελεσματικότητα της βάσης καθώς επίσης και την ασφάλεια της. Στην περίπτωση αυτή η είσοδος του προηγούμενου παραδείγματος θα αντιμετωπιζόταν σαν

συμβολοσειρά και όχι σαν λογική πρόταση. Αυτό σημαίνει ότι στην βάση θα γινόταν αναζήτηση για εγγραφή με το όνομα '1=1'.

2. **Escaping**: Είναι μια μέθοδος κατά την οποία αποφεύγονται ειδικοί χαρακτήρες, δηλαδή χαρακτήρες που έχουν κάποια συγκεκριμένη σημασία για την γλώσσα SQL. Για παράδειγμα κάθε φόρα που συναντάμε μονό εισαγωγικό (′), θα πρέπει να αλλάζει με διπλό για να δημιουργείται έγκυρη συμβολοσειρά στην γλώσσα SQL.

3. **Database Permissions**: Είναι μια μέθοδος κατά την οποία ο προγραμματιστής της βάσης δίνει συγκεκριμένες διαθέσιμες ενέργειες σε κάθε πίνακα της βάσης. Αυτή η μέθοδος δεν προσφέρει πλήρη προστασία απέναντι στην επίθεση SQL Injection αλλά περιορίζει κατά πολύ την πιθανότητα να συμβεί. Υπάρχουν επιθέσεις που θα απαιτήσουν από πίνακες να αλλάξουν ή να διαγραφούν. Εάν αυτοί οι πίνακες δεν έχουν προγραμματιστεί να έχουν αυτήν την άδεια δεν θα συμβεί κάτι κατά την διάρκεια της επίθεσης.

2.3.2 Brute Force Attacks

Επιθέσεις 'ωμής βίας' ή αλλιώς Brute Force Attacks ονομάζονται αυτές όπου ο επιτιθέμενος προσπαθεί δοκιμάζοντας διαφορετικούς κωδικούς ή ονόματα να έχει πρόσβαση σε ένα σύστημα.

Περιγραφή Επίθεσης

Όπως αναφέρθηκε παραπάνω οι επιθέσεις αυτές δεν είναι τίποτα άλλο από την διαρκή προσπάθεια ενός επιτιθέμενου ή ενός μηχανισμού που δουλεύει για λογαριασμό του επιτιθέμενου να βρει έναν κωδικό, ένα όνομα χρήστη ή και τα δύο, δοκιμάζοντας πολλούς διαφορετικούς συνδυασμούς, μέχρι να βρεθεί ο σωστός. Η μέθοδος αυτή δεν θεωρείται καλή καθώς πολλές φορές απαιτεί πολλή μεγάλη προσπάθεια και πολύ χρόνο. Είναι σημαντικό να αναφερθεί ότι αποτελεί την τελευταία λύση για έναν επιτιθέμενο που την χρησιμοποιεί όταν δεν έχει βρει άλλα αδύναμα σημεία σε ένα σύστημα.

Τρόποι Αντιμετώπισης

Οι τρόποι αντιμετώπισης αυτής της επίθεσης είναι πολύ συγκεκριμένοι.

1. **Strong Passwords**: Πλέον κάθε διαδικτυακή εφαρμογή συμβουλεύει τους χρήστες να μην χρησιμοποιούν ίδιους κωδικούς παντού και ακόμη οι κωδικοί αυτοί να είναι μακροσκελείς και δύσκολοι με διάφορους χαρακτήρες μέσα. Έτσι αυξάνεται εκθετικά ο αριθμός προσπαθειών του επιτιθέμενου να βρει τον σωστό κωδικό.

2. **Limited tries**: Ο πιο αποτελεσματικός τρόπος αντιμετώπισης αυτής της επίθεσης είναι να υπάρχει περιορισμός προσπαθειών εισαγωγής κωδικού. Έτσι το να μαντέψει κάποιος τον κωδικό σε 3 ή 5 προσπάθειες είναι σχεδόν μηδενικές.

2.3.3 Cross-Site Scripting (XSS)

Το Cross-Site Scripting (XSS) αποτελεί μια επίθεση κατά την οποία κακόβουλα κομμάτια κώδικα (scripts) εισέρχονται μέσα στον κώδικα μιας, συνήθως, καλής και αξιόλογης ιστοσελίδας προκαλώντας σε αυτήν διάφορα προβλήματα, όπως λανθασμένη ή μη αναμενόμενη λειτουργία.

Περιγραφή Επίθεσης

Όπως αναφέρθηκε και παραπάνω μια επίθεση Cross-Site Scripting (XSS) είναι το 'πέρασμα' ενός κακόβουλου script μέσα στον κώδικα μιας ιστοσελίδας. Για να γίνει καλύτερα αντιληπτή η συγκεκριμένη επίθεση θα πρέπει αρχικά να χωριστεί σε 2 βασικές κατηγορίες. Η πρώτη κατηγορία ,που είναι και η πιο συνηθισμένη, είναι οι non-persistent ή reflected επιθέσεις XSS οι οποίες συνήθως σαν αποτέλεσμα μιας ενέργειας του χρήστη επιστρέφουν τα δεδομένα που έχει εισάγει σε μορφή σφάλματος ή αποτελέσματος μιας αναζήτησης ή κάποιας άλλης διεργασίας, χωρίς αυτά τα δεδομένα να είναι ασφαλή παρουσιαστούν στον browser. Η δεύτερη κατηγορία ,που ίσως είναι και η πιο επικίνδυνη, είναι οι persistent ή stored επιθέσεις XSS στις οποίες τα δεδομένα ή τα scripts τα οποία εισάγει ένας επιτιθέμενος μένουν αποθηκευμένα στον server και από εκεί εμφανίζονται στην ιστοσελίδα χωρίς να είναι ασφαλή για να παρουσιάζονται εκεί (π.χ script το οποίο είναι σύνδεσμος που οδηγεί στην εκτέλεση ενός κακόβουλου κώδικα που έχει εισάγει στην σελίδα ο επιτιθέμενος (payload)).

Τρόποι Αντιμετώπισης

Οι βασικοί τρόποι με τους οποίους αντιμετωπίζεται η συγκεκριμένη επίθεση είναι 2.

1. **Input Validation**: Αποτελεί μια μέθοδο κατά την οποία αξιολογείται κάθε είσοδος είτε από HTTP Requests, είτε σε μορφή URL Links, HTML form fields,HTTP headers,HTTP cookies. Ύποπτα πρέπει να θε-

ωρούνται και δεδομένα που έρχονται από την βάση ή από backend web services της σελίδας. Μια καλή αρχή είναι να θεωρούνται όλες οι είσοδοι κακόβουλες και μετά να κρατούνται μόνο αυτές που αποδεδειγμένα δεν είναι κακόβουλες.

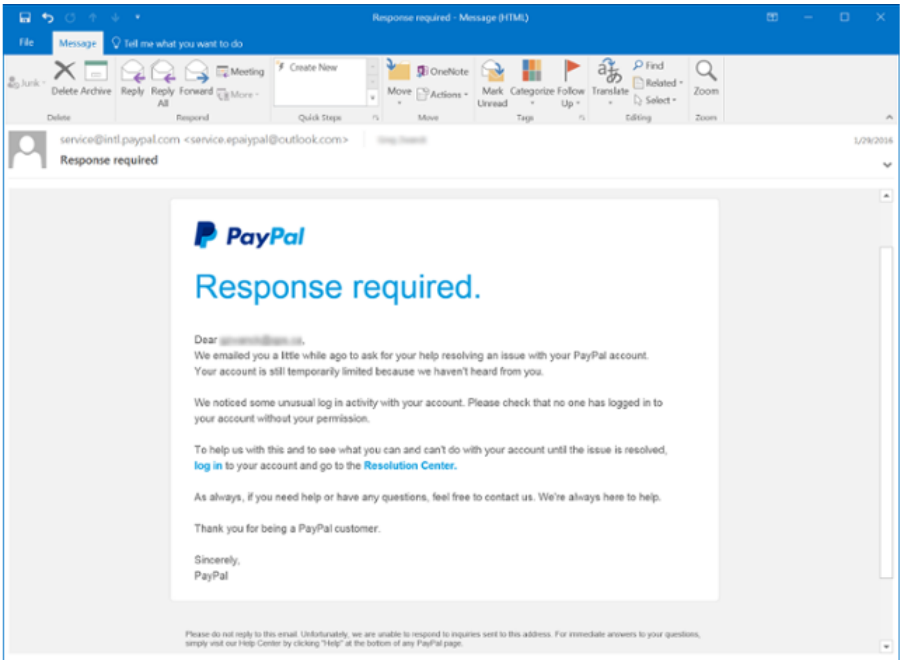
2. **Output Escaping**: Είναι μια ακόμη αποτελεσματική μέθοδος καταπολέμησης της επίθεσης XSS. Σύμφωνα με αυτήν ειδικοί χαρακτήρες όπως για παράδειγμα το "<script>" θα πρέπει να μεταχειρίζονται σαν κώδικας HTML αλλά και να μην αποτελούν κομμάτι του κώδικα της Javascript.

2.3.4 Phising

Το Phising ορίζεται σαν μια επίθεση η οποία έχει σαν σκοπό την κλοπή προσωπικών δεδομένων από χρήστες χωρίς να το αντιληφθούν. Αυτό επιτυγχάνεται σχετικά απλά, αφού οι μηχανισμοί αυτοί είναι μασκαρεμένοι σαν μια διαδικτυακή υπηρεσία άξια εμπιστοσύνης.

Περιγραφή Επίθεσης

Με βάση τον παραπάνω ορισμό γίνεται εύκολα κατανοητό ότι η διαδικασία αυτή στοχεύει άμεσα στην απροσεξία ή ακόμα και στην απειρία ενός χρήστη. Παρακάτω ακολουθούν κάποιες εικόνες για την ευκολότερη κατανόηση του τρόπου λειτουργίας αυτή της επίθεσης.



Σχήμα 2.3: Ένα E-mail, από έμπιστο φορέα το οποίο ενημερώνει τον παραλήπτη ότι παρατηρήθηκε ύποπτη κίνηση στον λογαριασμό του.

Η παραπάνω εικόνα μας δείχνει ένα E-mail, από έμπιστο φορέα το οποίο ενημερώνει τον παραλήπτη ότι παρατηρήθηκε ύποπτη κίνηση στον λογαριασμό του. Εάν ο χρήστης δεν υποψιαστεί κάτι θα πατήσει στον έτοιμο σύνδεσμο που του δίνεται θα εισάγει τα στοιχεία του για να εισέλθει στον λογαριασμό του και αυτομάτως θα στοιχεία αυτά θα υποκλαπούν από τον επιτιθέμενο.

Μια τέτοιου είδους επίθεση βέβαια μπορεί να πραγματοποιηθεί και με πολύ πιο απλό τρόπο. Έστω ότι ένας κακόβουλος χρήστης θέλει να υποκλέψει διάφορα προσωπικά δεδομένα από χρήστες. Ένας απλός τρόπος είναι να φτιάξει ένα ψεύτικο προφίλ είτε ανθρώπου είτε κάποιας υπηρεσίας, στα μέσα κοινωνικής δικτύωσης, το οποίο όμως θα πρέπει να προκαλεί στους χρήστες εμπιστοσύνη για να μην έχουν υπόνοιες ότι πρόκειται για κάτι που θα τους προκαλέσει ζημιά, και να ζητάει στοιχεία από αυτούς όπως κάποιο τηλέφωνο ή κάποια διεύθυνση.

Τρόποι Αντιμετώπισης

Η αλήθεια είναι ότι οι τρόποι αντιμετώπισης του Phising είναι λίγο περιορισμένοι γιατί η επίθεση αυτή βασίζεται στον παράγοντα του ανθρώπινου λάθους και όχι σε κάποια αδυναμία κώδικα. Συνεπώς η καλύτερη αντιμετώπιση της επίθεσης αυτής είναι η σωστή εκπαίδευση των χρηστών απέναντι σε τέτοια φαινόμενα. Κάποιοι άλλοι τρόποι είναι:

1. **Mail carriers with good spam filters**: Η χρήση παροχών ηλεκτρονικής αλληλογραφίας που έχουν καλούς μηχανισμούς αναγνώρισης spam μηνυμάτων.

2. **Browser plugins**: Η χρήση επεκτάσεων σε browsers που ελέγχουν κατά πόσο έμπιστη είναι η κάθε ιστοσελίδα που θέλει να επισκεφθεί ο εκάστοτε χρήστης.

2.4 Σημασία της Ασφάλειας

Μετά από όσα παρουσιάστηκαν παραπάνω γίνεται αντιληπτό το πόσο σημαντική είναι η ασφάλεια στον χώρο του διαδικτύου στις μέρες μας. Καθημερινά, υπάρχουν χρήστες στον κόσμο που πέφτουν θύματα επιθέσεων και κάποιες φορές οι συνέπειες είναι πάρα πολύ σοβαρές σε πολλά φάσματα της ζωής τους. Το πεδίο της ασφάλειας των συστημάτων συνεχώς αναπτύσσεται και νέοι τρόποι προστασίας βρίσκονται διαρκώς. Παρ ΄όλα αυτά όμως αναγκαία θεωρείται κιόλας η ορθή και προσεκτική χρήση των υπολογιστών και του διαδικτύου από τους χρήστες.

Στα επόμενα 2 κεφάλαια θα γίνει παρουσίαση των 2 επιθέσεων που πραγματεύεται η παρούσα Διπλωματική Εργασία, του Buffer Overflow και των Format String Attacks. Όπως αναφέρθηκε και παραπάνω, σε αντίθεση με τις παραπάνω επιθέσεις που είναι υψηλού επιπέδου, αυτές οι 2 είναι χαμηλού επιπέδου. Αυτό σημαίνει ότι οι συνέπειες τους δεν είναι απώλεια πληροφοριών κάποιων χρηστών στο διαδίκτυο ή παράνομη είσοδος σε κάποια διαδικτυακή εφαρμογή ή ιστοσελίδα, αλλά δημιουργία προβλημάτων που αφορούν πιο άμεσα τα δεδομένα μνήμης ενός υπολογιστικού συστήματος.

Κεφάλαιο 3

Υπερχείλιση Προσωρινής Μνήμης(Buffer Overflow)

Σε αυτό το κεφάλαιο θα αναφερθούμε εκτενώς στην επίθεση Buffer Overflow που είναι μια από τις πιο διαδεδομένες επιθέσεις στον κόσμο των υπολογιστών. Θα υπάρξει περιγραφή της επίθεσης αυτής, θα υπάρξει αναφορά στα χαρακτηριστικά της, σε παραδείγματα καθώς και στους τρόπους αντιμετώπισής της.

3.1 Περιγραφή

Η επίθεση Buffer Overflow είναι μια από τις πιο διαδεδομένες επιθέσεις στον κόσμο των υπολογιστών, όπως αναφέρθηκε και παραπάνω. Η επίθεση αυτή ορίζεται σαν μια ανωμαλία κατά την διάρκεια εκτέλεσης ενός προγράμματος κατά την οποία τα δεδομένα που γράφονται σε έναν buffer επειδή είναι περισσότερα από όσα χωράει ο buffer υπερχειλίζουν. Αυτό έχει ως αποτέλεσμα ο παραπανίσιος όγκος δεδομένων να γράφεται σε γειτονικές θέσεις μνήμης.

Οι buffers είναι δομές (προσωρινές μνήμες) που είναι σχεδιασμένοι να αποθηκεύουν δεδομένα τα οποία μπορεί να τα μεταφέρουν συχνά από κάποια θέση του προγράμματος σε μια άλλη ή ακόμα και από ένα πρόγραμμα σε ένα άλλο πρόγραμμα ή υποπρόγραμμα (π.χ συναρτήσεις). Η Υπερχείλιση Προσωρινής

Μνήμης μπορεί να δημιουργηθεί από λανθασμένου μεγέθους δεδομένα. Έστω λοιπόν ότι κατά την εκτέλεση ενός προγράμματος ο χρήστης ο οποίος εισάγει τα δεδομένα θεωρεί πως όλα τα δεδομένα είναι μικρότερα από ένα συγκεκριμένο μέγεθος το οποίο δηλώνει το πόση χωρητικότητα έχουν οι προσωρινές μνήμες (buffers). Με αυτήν την λογική όλα τα δεδομένα θα έχουν το κατάλληλο μέγεθος για να αποθηκευτούν μέσα στις μνήμες. Όμως εάν μια δυσλειτουργία εντός του προγράμματος παράξει παραπάνω δεδομένα αυτά αυτομάτως εάν επιχειρήσουν να αποθηκευτούν εντός των προσωρινών μνημών θα προκαλέσουν την υπερχείλιση τους και τα παραπανίσια δεδομένα θα γραφτούν σε γειτονικές θέσεις μνήμης.

Το πρόβλημα αυτό μπορεί να μην φαντάζει τόσο σοβαρό παρ΄ όλα αυτά αποτελεί μια πολύ επικίνδυνη κατάσταση. Το γεγονός ότι τα δεδομένα που περισσεύουν θα αποθηκευτούν σε γειτονικές θέσεις μνήμης μπορεί να προκαλέσει στο πρόγραμμα που συνέβη το Buffer Overflow λανθασμένη λειτουργία, παραγωγή λανθασμένων αποτελεσμάτων μέχρι και να το κάνει μη εκτελέσιμο (crash).

Σε πολλά συστήματα όσο καλά και αν είναι ορισμένη η δομή της μνήμης ή και ολόκληρου του συστήματος στέλνονται από κακόβουλους χρήστες δεδομένα τα οποία έχουν σαν στόχο να προκαλέσουν Buffer Overflow, με αποτέλεσμα από την υπερχείλιση να γράφονται δεδομένα σε θέσεις μνήμης που είναι ικανά να προκαλέσουν διάφορα προβλήματα. Εκτός αυτού πολλές φορές ο στόχος του επιτιθέμενου είναι να αποκτήσει πρόσβαση στο σύστημα χωρίς να έχει τα διαπιστευτήρια (π.χ όνομα χρήστη, κωδικό).

Οι γλώσσες προγραμματισμού στις οποίες είναι διαδεδομένη η Υπερχείλιση Προσωρινής Μνήμης είναι η C και η C++.

3.2 Παραδείγματα

Με βάση όλα όσα αναφέρθηκαν παραπάνω ήρθε η ώρα να δούμε τα αποτελέσματα της Υπερχείλισης Προσωρινής Μνήμης στην πράξη μέσω διαφορετικών παραδειγμάτων. Στο συγκεκριμένο τμήμα της του Κεφαλαίου 2 θα γίνει εκτενής παρουσίαση παραδειγμάτων από την συγκεκριμένη επίθεση καθώς και γραφική παρουσίαση των αποτελεσμάτων και συνεπειών της.

3.2.1 Παραδείγμα 1

Όπως αναφέρθηκε παραπάνω η Υπερχείλιση Προσωρινής Μνήμης προκαλείται από την προσπάθεια αποθήκευσης σε μια δομή μνήμης, δεδομένων μεγαλύτερου όγκου από την ίδια με αποτέλεσμα τα δεδομένα που περισσεύουν να υπερχειλίζουν και να γράφονται σε διπλανές θέσεις μνήμης προκαλώντας έτσι διάφορα προβλήματα σε ένα σύστημα.

Έστω λοιπόν ότι σε ένα πρόγραμμα γραμμένο στην γλώσσα C συναντάμε αυτές τις 2 γειτονικές μεταβλητές.

```
char           A[8] = "";
unsigned short B    = 1979;
```

Σχήμα 3.1: Δυο γειτονικές μεταβλητές σε ένα πρόγραμμα γραμμένο στην γλώσσα. C

Ας δούμε τώρα πως φαίνονται αυτές οι μεταβλητές στην μνήμη του υπολογιστή μας.

variable name	A								B	
value	[null string]								1979	
hex value	00	00	00	00	00	00	00	00	07	BB

Σχήμα 3.2: Δυο γειτονικές μεταβλητές στην μνήμη με τις αρχικές τιμές τους.

Όπως βλέπουμε οι 2 μεταβλητές μας είναι σε γειτονικές θέσεις μνήμης και η μια έχει αρχική τιμή "" (κενή συμβολοσειρά) όπως ορίζεται από τον κώδικα στο Σχήμα 3.1 ενώ η άλλη έχει αρχική τιμή ¨1979¨. Οι αντίστοιχες τιμές μεταφράζονται και στο δεκαεξαδικό σύστημα με βάση το Σχήμα 3.2.

Έστω ότι η μεταβλητή 'A' , που είναι ένας πίνακας buffer 8-bytes χαρακτήρων, παίρνει μια τιμή μέσω της εντολής strcpy. Η εντολή αυτή δέχεται μια συμβολοσειρά και την αποθηκεύει σε έναν buffer και η δομή της είναι η ακόλουθη:

```
strcpy(A, "excessive");
```

Σχήμα 3.3: Εντολή strcpy που δέχεται μια συμβολοσειρά και την αποθηκεύει στον buffer 'A'.

Όπως παρατηρούμε η συμβολοσειρά "excessive" είναι μήκους 9 bytes και μαζί με τον τερματικό χαρακτήρα ("null terminator", που χρησιμοποιείται στην γλώσσα C για να τερματίζει τις συμβολοσειρές), γίνεται μήκους 10 bytes. Όπως καταλαβαίνουμε η συμβολοσειρά "excessive" είναι μεγαλύτερη από το μέγεθος του buffer και άρα η εντολή του Σχήματος 3.3 θα προκαλέσει υπερχείλιση.

variable name	A								B	
value	'e'	'x'	'c'	'e'	's'	's'	'i'	'v'	25856	
hex	65	78	63	65	73	73	69	76	65	00

Σχήμα 3.4: Η υπερχείλιση της μνήμης μας.

Είναι προφανές μετά από την εκτέλεση της εντολής strcpy ότι η τιμή της μεταβλητής 'B' έχει αλλάξει λόγω της υπερχείλισης που συνέβη. Η αλλαγή αυτή οφείλεται στο 'e' και στον τερματικό χαρακτήρα της συμβολοσειράς, γιατί αυτοί ήταν οι χαρακτήρες που δεν χωρούσαν μέσα στον buffer A . Στην περίπτωση μας ο συνδυασμός αυτός παρήγαγε τον αριθμό 25856.

Είναι λογικό ένας προγραμματιστής να μην ξέρει στην είσοδο που θα εισάγει ο εκάστοτε χρήστης. Παρ΄ όλα αυτά, ο ίδιος μπορεί να αποτρέψει το πρόβλημα αυτό με πολλούς τρόπους στους οποίους θα γίνει εκτενής αναφορά παρακάτω. Μια πιθανή λύση θα ήταν η αντικατάσταση της εντολής strcpy με την strncpy η οποία έχει την ίδια λειτουργία αλλά δεν είναι ευάλωτη στην Υπερχείλιση Προσωρινής Μνήμης γιατί ένα από τα ορίσματα της είναι το μέγεθος του buffer.

```
strncpy(A, "excessive", sizeof(A));
```

Σχήμα 3.5: Η εντολή strncpy που θα μπορούσε να αποτρέψει το παραπάνω πρόβλημα της Υπερχείλισης Προσωρινής Μνήμης

3.2.2 Παραδείγμα 2

Παραπάνω έγινε αναφορά στο γεγονός ότι πολλά συστήματα δέχονται επιθέσεις Buffer Overflow με σκοπό ο επιτιθέμενος μέσω μιας υπερχείλισης που θα προκαλέσει να έχει πρόσβαση σε αυτό στην ουσία 'παράνομα'. Είναι σημαντικό να αναφερθεί ότι η υπερχείλιση μιας μνήμης μπορεί να έχει και παράπλευρες συνέπειες πέρα από την ίδια την υπερχείλιση της, όπως είναι και να αποκτήσει πρόσβαση ένας κακόβουλος χρήστης σε ένα σύστημα με σκοπό να προκαλέσει κάποιου είδους ζημιά ή ακόμα και να βρει ή να τροποποιήσει ευαίσθητες πληροφορίες. Το παρακάτω παράδειγμα χρησιμοποιείται και θεωρείται σημαντικό επειδή αποτελεί μια προσομοίωση αυτής της περίπτωσης.

Έστω λοιπόν ότι έχουμε τον παρακάτω κώδικα στην γλώσσα προγραμματισμού C.

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char buff[15];
    int pass = 0;
        printf("\n Enter the password : \n");
    gets(buff);
    if(strcmp(buff, "thegeekstuff"))
    {
        printf ("\n Wrong Password \n");
    }
    else
    {
        printf ("\n Correct Password \n");
        pass = 1;
    }
    if(pass)
    {
       /* Now Give root or admin rights to user*/
        printf ("\n Root privileges given to the user \n");
    }
    return 0;
}
```

Στο συγκεκριμένο πρόγραμμα λοιπόν ορίζεται ένας buffer 15-bytes ο οποίος θα δεχθεί μια συμβολοσειρά από τον χρήστη. Η συμβολοσειρά αυτή θα είναι ο κωδικός πρόσβασης ο οποίος εάν είναι ο σωστός τότε θα εισέλθει ο χρήστης στο σύστημα. Αντιθέτως, εάν ο κωδικός είναι ο οποιοσδήποτε άλλος δεν θα μπορέσει να έχει πρόσβαση. Η εντολή που διαχειρίζεται την λήψη του κωδικού είναι η gets που όπως είναι γνωστό είναι ευάλωτη στην επίθεση Buffer Overflow. Ποιο θα είναι το αποτέλεσμα εάν υπερχειλίσει ο buffer "buff".

Πριν απαντηθεί η προηγούμενη ερώτηση ας δούμε εάν με τον σωστό κωδικό ο χρήστης έχει πρόσβαση στο σύστημα.

```
$ ./bfrovrflw

 Enter the password :
thegeekstuff

 Correct Password

 Root privileges given to the user
```

Σχήμα 3.6: Με τον σωστό κωδικό ο χρήστης έχει πρόσβαση στο σύστημα.

Τώρα ας δούμε σε περίπτωση που ο χρήστης βάλει κωδικό μεγαλύτερο απο 15-bytes τι θα συμβεί.

```
$ ./bfrovrflw

 Enter the password :
hhhhhhhhhhhhhhhhhhhh

 Wrong Password

 Root privileges given to the user
```

Σχήμα 3.7: Με λάθος και μεγαλύτερο κωδικό απο 15-bytes ο χρήστης έχει πάλι πρόσβαση

Η πρώτη απορία που μας γεννάται είναι για ποιόν λόγο συνέβη αυτό. Η απάντηση είναι σχετικά απλή. Όπως και στο πρώτο παράδειγμα οι μεταβλητές buff και και pass είναι γειτονικές. Με τον λανθασμένο κωδικό ο buffer buff υπερχείλισε και έτσι άλλαξε την τιμή της μεταβλητής pass. Έτσι παρά τον λανθασμένο κωδικό η μεταβλητή έγινε μη μηδενική και ο επιτιθέμενος απέκτησε πρόσβαση στο σύστημα.

Η αλλαγή που θα απέτρεπε αυτήν την επίθεση θα ήταν στην θέση της εντολής gets να χρησιμοποιηθεί η ασφαλής εκδοχή της fgets.

3.3 Τρόποι Αντιμετώπισης

Το Buffer Overflow σαν επίθεση μπορεί να αντιμετωπιστεί με διάφορες τεχνικές. Οι τεχνικές αυτές εφαρμόζονται κατά την ανάπτυξη ενός κώδικα ούτως ώστε να είναι όσο το δυνατόν πιο ασφαλής απέναντι σε τέτοιου είδους επιθέσεις. Οι πιο πρόσφατες και δημοφιλείς τεχνικές είναι το Bound Checking,τα Canaries,το Address Space Layout Randomization,το Executable Space Protection και η Χρήση Ασφαλών εντολών. Παρακάτω θα αναλυθεί κάθε μια τεχνική ξεχωριστά.

3.3.1 Bounds Checking

Η πρώτη μέθοδος είναι γνωστή στα αγγλικά σαν bound checking. Σκοπός της είναι ο έλεγχος των δεδομένων που πρόκειται να εισαχθούν σε έναν buffer πριν αυτά εισέλθουν σε αυτόν. Ο έλεγχος αυτός αφορά το μέγεθος των δεδομένων αυτών. Για παράδειγμα εάν μια συμβολοσειρά 20 χαρακτήρων πρέπει να αποθηκευτεί σε έναν buffer που είναι 10 θέσεων ο έλεγχος αυτός το αποτρέπει. Το bound checking σαν μέθοδος είναι καλή και αποτελεσματική και θεωρείται ασφαλής. Παρ΄ όλα αυτά δεν θεωρείται καθόλου πρακτική. Δυστυχώς, για μεγάλο αριθμό δεδομένων και γενικότερα σε μεγάλα και περίπλοκα προγράμματα καθυστερεί πολύ την εκτέλεση τους επειδή αυξάνει κατά πολύ τον φόρτο του προγράμματος λόγω των συνεχών ελέγχων και έτσι η τεχνική αυτή κατά των Buffer Overflows θεωρείται μη αποδοτική. Για αυτόν τον λόγο στην συγκεκριμένη διπλωματική εργασία δεν υπάρχει υλοποίηση της μεθόδου αυτής.

3.3.2 Canaries

Αυτή η μέθοδος περιλαμβάνει την χρήση canaries που είναι μια από τις πιο διαδεδομένες καθώς αποτελεί και συνολικά την πιο αποτελεσματική. Η μέθοδος αυτή χρησιμοποιεί τυχαίες ακέραιες τιμές οι οποίες αποθηκεύονται ανάμεσα στους buffers (εάν είναι ένας ο buffer αποθηκεύονται στο τέλος του). Ο τρόπος αντιμετώπισης του Buffer Overflow με αυτήν την μέθοδο είναι η εξής:

1. Παράγονται τυχαίες ακέραιες μεταβλητές για κάθε έναν από τους buffers που υπάρχουν στο πρόγραμμα και αποθηκεύονται αμέσως μετά από τον κάθε ένα buffer.

2. Σε κάθε εισαγωγή δεδομένων σε κάποιον ή σε κάποιους buffers ελέγχονται οι τιμές αυτές εάν είναι ίδιες με αυτές που έχουν οριστεί εξ άρχής.

3. Εάν οι τιμές είναι ίδιες τότε δεν υπήρξε Buffer Overflow.

4. Εάν υπήρξε αλλαγή τότε αυτό είναι δείγμα ότι υπήρξε Buffer Overflow στον ή στους buffers των οποίων οι τιμές των canaries τους άλλαξαν. Έτσι μετά από αυτήν την παρατήρηση η εκτέλεση του προγράμματος διακόπτεται.

Η λογική που ακολουθείτε σε αυτήν την μέθοδο είναι ότι σε περίπτωση Buffer Overflow τα παραπανίσια δεδομένα θα αρχίσουν να γράφονται στις αμέσως επόμενες θέσεις μνήμης. Το γεγονός αυτό θα προκαλέσει αλλαγή στις τιμές των canaries και αυτό αυτομάτως θα σημαίνει ότι ανάλογα με ποια τιμή άλλαξε, ο αντίστοιχος buffer θα έχει πρόβλημα υπερχείλισης εφόσον η αντιστοιχία canaries και δεδομένων είναι ένα προς ένα.

Είναι πολύ σημαντικό να αναφερθεί ότι η τυχαιότητα των τιμών των canaries θεωρείτε απαραίτητη καθώς από μόνη της προσφέρει μια μεγαλύτερη ασφάλεια. Αυτό συμβαίνει γιατί επειδή ακριβώς οι τιμές αυτές είναι τυχαίες ο επιτιθέμενος δεν μπορεί να τις μαντέψει ή έστω να τις εκμαιεύσει με κάποιον τρόπο. Σε αντίθετη περίπτωση θα μπορούσε ο επιτιθέμενος να προσθέσει τις τιμές αυτές στα δεδομένα που ξέρει πως θα προκαλέσουν υπερχείλιση και έτσι να αντικαταστήσει τις τιμές αυτές με τις ίδιες ακριβώς. Έτσι η επίθεση του δεν θα ήταν

εμφανής και θα πέρναγε απαρατήρητη. Αξίζει επίσης να αναφερθεί ότι οι τιμές των canaries καλό είναι να έχουν αρκετά ψηφία για τους ίδιους λόγους. Στην περίπτωση λοιπόν που παρατηρηθεί Buffer Overflow, δηλαδή αλλαγή σε κάποιο ή κάποια τιμή των canaries η εκτέλεση του προγράμματος σταματάει.

3.3.3 Χρήση Ασφαλών εντολών

Αυτή η μέθοδος είναι στην ουσία η εξ΄ αρχής χρήση εντολών που δεν θεωρούνται ευάλωτες στο Buffer Overflow. Όπως είναι γνωστό υπάρχουν εντολές στην γλώσσα C, όπως για παράδειγμα η scanf, gets, strcpy, που θεωρούνται ευάλωτες προς αυτήν την επίθεση. Η μέθοδος αυτή είναι αρκετά αποτελεσματική καθώς εξ όρισμού αποτρέπει αυτού του είδους τις επιθέσεις. Η χρήση εντολών που κρίνονται ασφαλής ενάντια στο Buffer Overflow (π.χ fgets, strncpy κλπ) θεωρείται αναγκαία και αποτελεί και δείγμα σωστού προγραμματισμού, γιατί εξ όρισμού οι εντολές αυτές έχουν σαν όρισμα το μέγεθος του buffer στον οποίο θα αποθηκευτούν τα εκάστοτε δεδομένα που διαχειρίζονται. Η συγκεκριμένη μέθοδος έχει υλοποιηθεί για τις παρακάτω εντολές:

1. gets.
2. strcpy.
3. strcat.

Έτσι λοιπόν όποια από τις εντολές αυτές παρατηρηθεί αντικαθίσταται με την αντίστοιχη ασφαλή της, δηλαδή

1. fgets.
2. strncpy.
3. strncat.

Σε αυτό το σημείο αξίζει να σημειωθεί ότι η μέθοδος αυτή λειτουργεί συμπληρωματικά με την τεχνική των canaries στην υλοποίηση της παρούσας Διπλωματικής Εργασίας.

3.3.4 Address Space Layout Randomization

Η συγκεκριμένη μέθοδος χρησιμοποιείται για την προστασία διάφορων συστημάτων από το Buffer Overflow. Η τεχνική αυτή έχει σαν σκοπό την τοποθέτηση απαραίτητων τμημάτων μνήμης μιας διεργασίας σε διευθύνσεις στην μνήμη που είναι πολύ δύσκολο να μαντέψει ο επιτιθέμενος, επειδή είναι τυχαίες. Συνεπώς η συγκεκριμένη τεχνική βασίζεται στην χαμηλή πιθανότητα που έχει ο επιτιθέμενος να μαντέψει σωστά την τοποθεσία στοιχείων που έχουν κατανεμηθεί τυχαία στην μνήμη. Το Address Space Layout Randomization θεωρείται μια αρκετά δημοφιλής τεχνική και έχει υλοποιηθεί στα συστήματα Linux, Windows, Android, IOS, OS X κ.λ.π.

Κεφάλαιο 4

Format String Attacks

Σε αυτό το κεφάλαιο θα αναφερθούμε εκτενώς στην επίθεση Format String Attack που και αυτή όπως και η Υπερχείλιση Προσωρινής μνήμης που αναφέρθηκε παραπάνω είναι μια από τις πιο διαδεδομένες επιθέσεις στον κόσμο των υπολογιστών. Θα υπάρξει περιγραφή της επίθεσης αυτής, θα υπάρξει αναφορά στα χαρακτηριστικά της, σε παραδείγματα καθώς και στους τρόπους αντιμετώπισής της.

4.1 Περιγραφή

Η συγκεκριμένη επίθεση είναι μια από τις πιο διαδεδομένες επιθέσεις στον κόσμο των υπολογιστών. Το Format String Attack μπορεί να συμβεί όταν μια συμβολοσειρά που μπαίνει σαν είσοδος στο σύστημα δεν αξιολογείται/διαχειρίζεται σαν μια συμβολοσειρά αλλά σαν μια εντολή. Στην ουσία είναι λανθασμένη αναγνώριση μιας εισόδου που έχει ως επακόλουθο την λάθος διαχείρισή της που πολλές φορές οδηγεί και σε δυσάρεστα αποτελέσματα.

Εάν υποθέσουμε ότι μια τέτοια επίθεση γίνεται σε ένα σύστημα, ο επιτιθέμενος έχει την δυνατότητα να διαβάσει την στοίβα της μνήμης ακόμα και να προκαλέσει segmentation fault. Με αυτόν τον τρόπο καταφέρνει ο επιτιθέμενος να αλλάξει προσωρινά ή και μόνιμα τον τρόπο λειτουργίας του συστήματος θέτοντας έτσι την ασφάλεια και την σταθερότητα του σε κίνδυνο.

Για την ευκολότερη κατανόηση της παρουσίασης της επίθεσης αυτής θεωρείται απαραίτητη η επεξήγηση κάποιων όρων.

1. Format Function ονομάζεται μια συνάρτηση όπως για παράδειγμα οι fprintf,printf κλπ που μετατρέπουν μη κατανοητές στον άνθρωπο μεταβλητές (που είναι σε γλώσσα μηχανής) σε κατανοητές συμβολοσειρές.

2. Format String ονομάζεται το όρισμα του Format Function που μπορεί να περιέχει κείμενο και παραμέτρους Format String στις οποίες θα αναφερθούμε παρακάτω.

3. Format String Parameter ονομάζονται τα σύμβολα όπως %s, %x που προσδιορίζουν τον τύπο μεταβλητής που θα διαχειριστεί το Format Function.

Όπως αναφέρθηκε και παραπάνω μια επίθεση Format String Attack μπορεί να συμβεί όταν δεν αξιολογείται σωστά μια συμβολοσειρά - είσοδος στο σύστημα. Στην περίπτωση αυτή για παράδειγμα εάν ένα Format String parameter όπως το %s είναι μέσα σε μια Format Function, ο υπολογιστής αυτόματα θα προσπαθήσει να το αντιστοιχίσει σε μια τέτοιου είδους μεταβλητή. Εάν το Format String parameter αυτό δεν είναι μέσα στο Format Function τότε αυτομάτως ο υπολογιστής την μεταβλητή που θα αντιστοιχούσε σε αυτό το Format String parameter δεν μπορεί να την διαχειριστεί σωστά, γιατί δεν αναγνωρίζει τι είδους δεδομένο είναι. Με αυτόν τον τρόπο ο επιτιθέμενος είναι ελεύθερος να προκαλέσει πρόβλημα στο σύστημα είτε διαβάζοντας δεδομένα από θέση ή θέσεις μνήμης που δεν θα έπρεπε να γνωρίζει, είτε να προκαλέσει στο σύστημα crash.

4.2 Παραδείγματα

Με βάση όλα όσα αναφέρθηκαν παραπάνω ήρθε η ώρα να δούμε τα αποτελέσματα του Format String Attack στην πράξη μέσω διαφορετικών παραδειγμάτων. Στο συγκεκριμένο τμήμα του Κεφαλαίου 4 θα γίνει εκτενής παρουσίαση παραδειγμάτων από την συγκεκριμένη επίθεση καθώς και γραφική παρουσίαση των αποτελεσμάτων και συνεπειών της.

4.2.1 Παραδείγμα 1

Το συγκεκριμένο παράδειγμα αναφέρεται στην περίπτωση όπου μια συμβολοσειρά , η οποία εισέρχεται στο σύστημα σαν είσοδος χρήστη, διαχειρίζεται λανθασμένα από το σύστημα αυτό έχει σαν αποτέλεσμα να εμφανιστούν στον επιτιθέμενο πληροφορίες που δεν θα έπρεπε να γνωρίζει, όπως περιεχόμενα από θέσεις μνήμης κ.ο.κ .

Αρχικά πρέπει να αναφερθούμε στον ίδιο τον κώδικα, ο οποίος και παρατίθεται παρακάτω.

```
#include  <stdio.h>
#include  <string.h>
#include  <stdlib.h>

int main (int argc, char **argv)
{
        char buf [100];
        int x = 1 ;
        snprintf ( buf, sizeof buf, argv [1] ) ;
        buf [ sizeof buf -1 ] = 0;
        printf ( "Buffer size is: (%d) \nData input: %s \n" , strlen (buf) , buf ) ;
        printf ( "X equals: %d/ in hex: %#x\nMemory address for x: (%p) \n" , x, x, &x) ;
        return 0 ;
}
```

Σχήμα 4.1: Κώδικας με τον οποίο θα παρουσιαστεί το πρώτο παράδειγμα του Format String Attack.

Όπως βλέπουμε έχουμε έναν κώδικα ο οποίος διαχειρίζεται 2 ειδών μεταβλητές, μια ακέραια και έναν πίνακα 100 θέσεων με χαρακτήρες. Ο πίνακας χαρακτήρων γεμίζει με μια είσοδο που δίνεται από τον χρήστη κατά την διάρκεια εκτέλεσης του προγράμματος μέσω της εντολής sprintf. Πιο μετά στο πρόγραμμα όπως φαίνεται ακολουθούν άλλες 2 εντολές τύπου Format Function, οι οποίες παρουσιάστηκαν παραπάνω. Η πρώτη από αυτές εμφανίζει το μέγεθος του πίνακα χαρακτήρων καθώς και το περιεχόμενο του και αντίστοιχα η επόμενη εμφανίζει το περιεχόμενο της ακέραιας μεταβλητής καθώς και την θέση μνήμης της.

Το συγκεκριμένο παράδειγμα μας δείχνει πως θα συμπεριφερθεί η συγκεκριμένη εφαρμογή εάν δεν γίνει σωστός χειρισμός της εισόδου από τις Format Functions. Για τις ανάγκες του συγκεκριμένου παραδείγματος θα γίνει πρώτα παρουσίαση της κανονικής λειτουργίας του προγράμματος και έπειτα της λανθασμένης.

Έστω ότι η είσοδος που θα βάλει ο χρήστης είναι μια κανονική συμβολοσειρά, όπως φαίνεται και στην παρακάτω εικόνα.

```
./formattest "Bob"
```

Σχήμα 4.2: Κανονική είσοδος στο πρόγραμμα που παρουσιάστηκε παραπάνω.

Με βάση αυτήν την είσοδο η έξοδος του προγράμματος αυτού διαμορφώνεται ως εξής.

```
Buffer size is (3)
Data input : Bob
X equals: 1/ in hex: 0x1
Memory address for x (0xbffff73c)
```

Σχήμα 4.3: Με την είσοδο "Bob" φαίνεται η κανονική λειτουργία του προγράμματος.

Ήρθε η ώρα τώρα να δούμε τι αποτελέσματα θα έχουμε με μια λίγο τροποποιημένη είσοδο και κατά πόσο θα αλλάξει η συμπεριφορά και η λειτουργία του προγράμματος. Έστω ότι η είσοδος θα είναι η εξής:

```
./formattest "Bob %x %x"
```

Σχήμα 4.4: Στο πρόγραμμα μπαίνει η είσοδος "Bob %x %x".

Όπως βλέπουμε μέσα στην είσοδο ο χρήστης έχει βάλει και Format String Parameters. Το αποτέλεσμα αυτών θα το δούμε στο παρακάτω σχήμα.

Το πρόγραμμα εκτελέστηκε και τα αποτελέσματα είναι τα ακόλουθα:

```
Buffer size is (14)
Data input : Bob bffff 8740
X equals: 1/ in hex: 0x1
Memory address for x (0xbffff73c)
```

Σχήμα 4.5: Στο πρόγραμμα όταν μπαίνει η είσοδος "Bob %x %x" αλλάζουν τα αποτελέσματα των εξόδων καθώς και η συμπεριφορά του προγράμματος αφού πλέον παρουσιάζονται περιεχόμενα από θέσεις μνήμης.

Όπως είναι προφανές τα αποτελέσματα είναι ίδια με αυτά με της προηγούμενης εκτέλεσης του προγράμματος. Η μόνη διαφορά είναι το περιεχόμενο του πίνακα χαρακτήρων. Όπως βλέπουμε ο πίνακας περιέχει το όνομα Bob το οποίο ακολουθείται από περιεχόμενα κάποιων θέσεων μνήμης οι οποίες προκύπτουν από τους χαρακτήρες %x. Αυτό συνέβη επειδή στην εντολή snprintf δεν χρησιμοποιήθηκε το όρισμα Format String. Έτσι το πρόγραμμα δεν ήξερε ότι η είσοδος είναι συμβολοσειρά και στις θέσεις των %x αναζήτησε στην μνήμη δεδομένα και τα τοποθέτησε μέσα στον πίνακα σε δεκαεξαδική μορφή.

Οι χαρακτήρες αυτοί από την στιγμή που μπήκαν μέσα στην συμβολοσειρά και αποδόθηκαν μέσα στον πίνακα buf εμφανίστηκαν μέσα από την εντολή printf.

4.2.2 Παραδείγμα 2

Στο παράδειγμα αυτό θα παρουσιαστεί μια περίπτωση στην οποία η επίθεση Format String Attack μπορεί να προκαλέσει crash, δηλαδή θα αναγκάσει το πρόγραμμα να σταματήσει να λειτουργεί.

Έστω λοιπόν ότι έχουμε έναν κώδικα σαν τον παρακάτω προς εκτέλεση:

```
#include <stdio.h>

int main ()
{
  printf ("%s%s%s%s%s%s%s%s%s%s%s%s");
}
```

Σχήμα 4.6: Κώδικας που θα προκαλέσει crash στο σύστημα.

Όταν θα έρθει η στιγμή να εκτελεστεί από ο κώδικας αυτός για κάθε χαρακτήρα %s που βρίσκεται στην εντολή printf ο υπολογιστής θα φέρει έναν αριθμό από την μνήμη τον οποίο και θα διαχειριστεί σαν διεύθυνση μνήμης και θα εμφανίσει τα περιεχόμενα αυτής της θέσης μνήμης σαν συμβολοσειρά (αυτό καθορίζεται απο τον χαρακτήρα %s) μέχρι να βρεθεί ένας χαρακτήρας NULL. Απο την στιγμή όμως που ο αριθμός ο οποίος θα έρθει από την μνήμη μπορεί να μην είναι καν διεύθυνση η θέση μνήμης στην οποία δείχνει ο αριθμός αυτός μπορεί να μην υπάρχει(δηλαδή δεν θα υπάρχει φυσική μνήμη που να είναι σε αντιστοίχιση με αυτήν την διεύθυνση) και έτσι το πρόγραμμα θα πάψει να λειτουργεί. Φυσικά υπάρχει και η περίπτωση η εντολή printf να φέρει αριθμούς που όντως να αντιστοιχούν σε κάποιες υπάρχουσες θέσεις μνήμης οι οποίες όμως να μην είναι προσπελάσιμες γιατί θα είναι δεσμευμένες από το σύστημα. Σε αυτήν την περίπτωση πάλι το πρόγραμμα θα πάψει να λειτουργεί.

4.3 Τρόποι Αντιμετώπισης

Η αντιμετώπιση της συγκεκριμένης επίθεσης γίνεται με αρκετούς τρόπους. Οι επικρατέστεροι από αυτούς είναι το Address Space Layout Randomization, η σωστή χρήση των Format Functions και η εξέταση των εισόδων του συστήματος. Στο σημείο αυτό θα γίνει παρουσίαση των τρόπων αυτών. Αξίζει να σημειωθεί ότι οι 2 τελευταίες μέθοδοι είναι αυτές που έχουν υλοποιηθεί στην παρούσα διπλωματική εργασία και η πιο συγκεκριμένη περιγραφή τους θα γίνει στο Κεφάλαιο 4.

4.3.1 Address Space Layout Randomization

Η συγκεκριμένη μέθοδος χρησιμοποιείται για την προστασία διάφορων συστημάτων από το Format String Attack. Η τεχνική αυτή έχει σαν σκοπό την τοποθέτηση απαραίτητων τμημάτων μνήμης μιας διεργασίας σε διευθύνσεις στην μνήμη που είναι πολύ δύσκολο να μαντέψει ο επιτιθέμενος. Συνεπώς η συγκεκριμένη τεχνική βασίζεται στην χαμηλή πιθανότητα που έχει ο επιτιθέμενος να μαντέψει σωστά την τοποθεσία στοιχείων που έχουν κατανεμηθεί τυχαία στην μνήμη. Το Address Space Layout Randomization θεωρείται μια αρκετά δημοφιλής τεχνική και έχει υλοποιηθεί στα συστήματα Linux, Windows, Android, IOS, OS X κ.λ.π. Δυστυχώς το μειονέκτημα της τεχνικής αυτής είναι ότι δεν παρέχει απόλυτη ασφάλεια επειδή κάθε επίθεση που πανογράφει δεδομένα που δεν είναι δείκτες μπορεί να παραβλέψει την προστασία που παρέχει το Address Space Layout Randomization.

4.3.2 Σωστή χρήση των Format Functions

Η συγκεκριμένη μέθοδος δεν είναι ακριβώς μέθοδος αντιμετώπισης της επίθεσης Format String Attack όσο τεχνική καλού προγραμματισμού που συνολικά αποτρέπει αυτού του είδους τις επιθέσεις. Πρόκειται για την σωστή χρήση των εντολών της οικογένειας printf. Σαν σωστή χρήση ή ορισμό αυτών των εντολών ορίζεται η χρήση των Format String Parameters πάντοτε. Παρακάτω ακολουθούν 2 σωστές και 2 λανθασμένες χρήσεις αυτών των εντολών.

```
printf(user_supplied_data);
printf("%s", user_supplied_data);

 fprintf(stderr, user_supplied_data);
fprintf(stderr, "%s", user_supplied_data);
```

Σχήμα 4.7: Σωστή και Λάθος (αντίστοιχα) χρήση των Format String Functions.

4.3.3 Εξέταση δεδομένων στο σύστημα

Η συγκεκριμένη μέθοδος στην ουσία προτείνει τον έλεγχο των δεδομένων που εισέρχονται ή που υπάρχουν, στο σύστημα, όταν αυτές χρησιμοποιούνται από Format Functions. Για τις ανάγκες της μεθόδου αυτής απαιτείται η ανάπτυξη κάποιου ξεχωριστού συστήματος ελέγχου ο οποίος έχει σαν στόχο να ανιχνεύει εάν στις εισόδους του συστήματος παρατηρούνται Format String Parameters, όπως για παράδειγμα αυτή "Hell%xo". Το σύστημα αυτό λοιπόν θα ελέγχει τις εισόδους αυτές και εάν εμπεριέχουν Format String Parameters, που είναι ικανά να προκαλέσουν κάποιο πρόβλημα, να τα αφαιρεί. Η διαδικασία αυτή έχει ένα μειονέκτημα το οποίο είναι ότι μπορεί να παραποιεί τις εισόδους αυτές στην προσπάθεια του να αφαιρέσει από τα Format String Parameters. Στην περίπτωση μας όμως δεν θεωρείται σοβαρό μειονέκτημα καθώς διασφαλίζει την ασφάλεια του συστήματος.

Κεφάλαιο 5

Περιγραφή του Συστήματος Εύρεσης και Αντιμετώπισης Ευάλωτου Κώδικα για Buffer Overflow και Format String Attacks

Στο συγκεκριμένο κεφάλαιο θα γίνει ανάλυση του συστήματος που φτιάχτηκε για την αντιμετώπιση των 2 προαναφερθέντων επιθέσεων. Θα γίνει εκτενής ανάλυση των μεθόδων που χρησιμοποιήθηκαν για την αντιμετώπιση τους, των πλεονεκτημάτων αλλά και των μειονεκτημάτων κάθε μιας και θα γίνει στο τέλος και μια γενική αξιολόγηση των δυνατοτήτων του συστήματος αυτού καθώς και παράθεση του κόστους σε εντολές assembly που προστίθεται στον ασφαλή κώδικα που παράγει εν τέλει ο μηχανισμός, από την εκτέλεση της κάθε μεθόδου αντιμετώπισης. Σημειώνεται ότι τα κόστη σε εντολές assembly, είναι οι παραπανίσιες εντολές ελέγχου που προστίθενται, δηλαδή έχουν υπολογιστεί στατικά. Αυτό σημαίνει οτι υπολογίζεται ο αριθμός εντολων ενός προγράμματος πριν και μετά την διόρθωση απο τον μηχανισμό.

5.1 Γενική Περιγραφή

Ο μηχανισμός ο οποίος αναπτύχθηκε στα πλαίσια της συγκεκριμένης Διπλωματικής εργασίας στην ουσία απαρτίζεται από 2 διακριτά σκέλη. Το πρώτο σκέλος είναι αυτό της εύρεσης κάποιας πιθανής επίθεσης και το δεύτερο είναι αυτό της αντιμετώπισης της. Οι τρόποι αντιμετώπισης αυτών των επιθέσεων έχουν αναφερθεί και παραπάνω στα κεφάλαια 3 και 4 μέσα στην περιγραφή των επιθέσεων. Προφανώς έχουν υλοποιηθεί κάποιοι από αυτούς τους τρόπους και όχι όλοι.

Το σύστημα αυτό είναι ικανό να ανιχνεύσει και να αντιμετωπίσει επιθέσεις Buffer Overflow και ορισμένες επιθέσεις Format String Attacks αυστηρά και μόνο σε επίπεδο λογισμικού. Είναι σημαντικό να αναφερθεί ότι είναι προγραμματισμένο στην γλώσσα προγραμματισμού Java και ότι είναι αυτόματος μηχανισμός.

Η αντιμετώπιση της κάθε επίθεσης χωρίζεται όπως είπαμε σε 2 στάδια. Όσον αφορά το δεύτερο στάδιο, που είναι αυτό της αντιμετώπισης, απαρτίζεται από τρεις τρόπους που είναι οι παρακάτω και εφαρμόζονται ξεχωριστά ανάλογα την περίπτωση της κάθε επίθεσης που αντιμετωπίζεται:

1. Πλαισιώνοντας τις επικίνδυνες εντολές είτε με πρόσθετες μεταβλητές ελέγχου είτε με συναρτήσεις ελέγχου, που έχουν σαν βασικό στόχο τον έλεγχο των εντολών σε συνδυασμό με τα ορίσματα που μπαίνουν σε μια εντολή.

2. Διόρθωση εντολών (κυρίως στην περίπτωση που η αδυναμία τους προέρχεται από ελλειπή σύνταξη).

3. Παύση εκτέλεσης του προγράμματος που χρησιμοποιείται συνήθως εάν κάποια από τις παραπάνω μεθόδους δεν είναι εφικτή.

Ο μηχανισμός όπως αναφέρθηκε και παραπάνω είναι αυτόματος. Με τον όρο αυτόματος εννοούμε ότι με την εκτέλεση του κώδικα του, τα πέντε στάδια του που περιγράφονται αμέσως μετά συμβαίνουν αυτόματα και σειριακά χωρίς να πρέπει να παρεμβληθεί κάτι άλλο για να συνεχιστεί κάποια διεργασία του. Συνολικά η λειτουργία όλου του μηχανισμού απαρτίζεται από τις παρακάτω διεργασίες.

1. Το πρόγραμμα δέχεται σαν είσοδο ένα αρχείο .txt όπου προσομοιώνει μια επίθεση από τις 2 κατηγορίες που αναφέρθηκαν παραπάνω σε γλώσσα C.

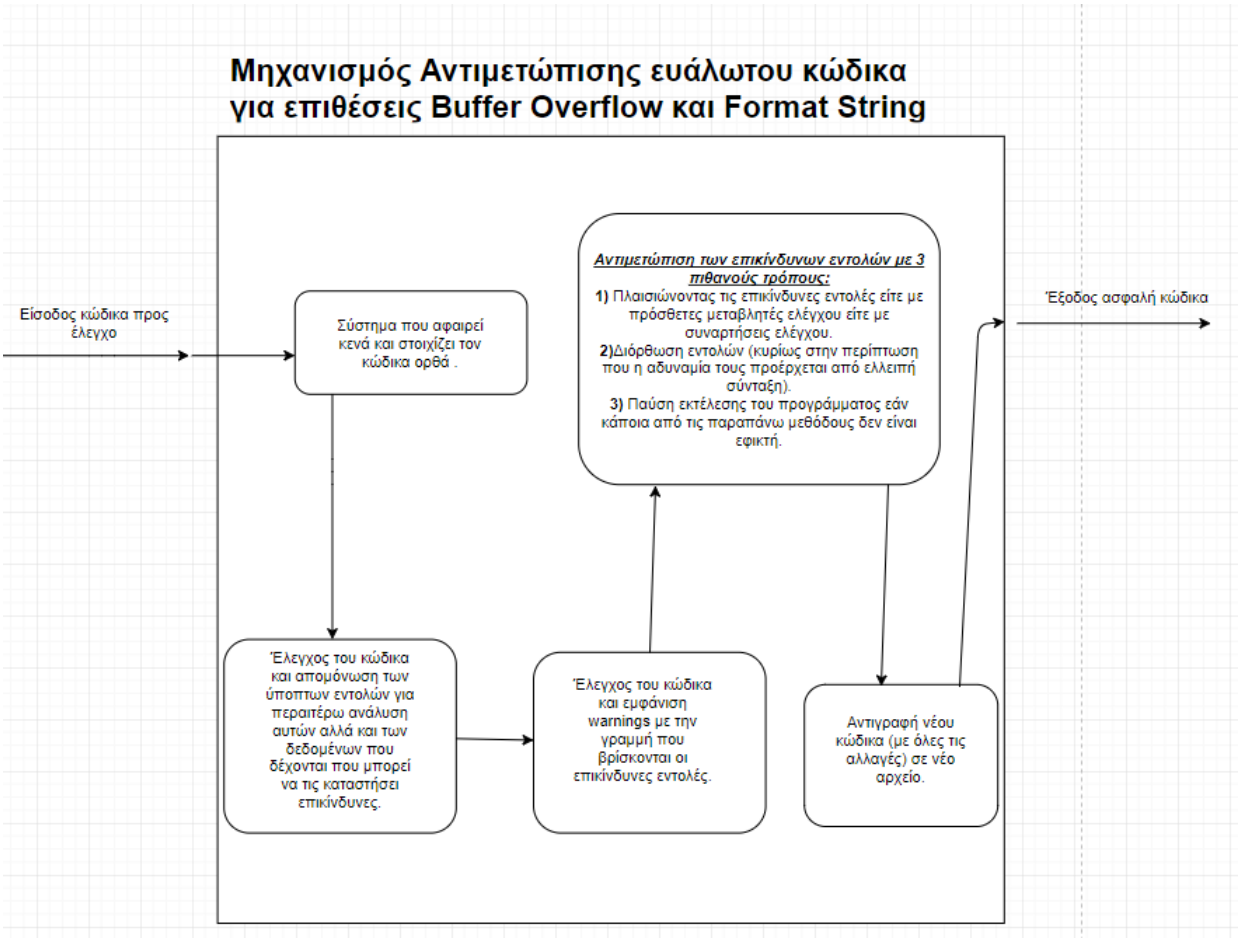
2. Ο κώδικας αυτός σαρώνεται και εντοπίζονται, εάν υπάρχουν, τα σημεία εκείνα τα οποία θεωρούνται ύποπτα ή επικίνδυνα.

3. Εμφανίζονται στον χρήστη warnings σχετικά με τις ευάλωτες αυτές εντολές καθώς και σχετικά με την θέση τους (γραμμή τους στον κώδικα).

4. Οι εντολές που κρίθηκαν επικίνδυνες ή ύποπτες αντιμετωπίζονται με διάφορους τρόπους που θα παρουσιαστούν παρακάτω.

5. Η έξοδος του μηχανισμού αυτού είναι πλέον ένα αρχείο C το οποίο είναι απαλλαγμένο από κινδύνους και κακόβουλο κώδικα.

Μια αναλυτική γραφική απεικόνιση του τρόπου λειτουργίας και της αλληλουχίας του μηχανισμού είναι η παρακάτω:



Σχήμα 5.1: Γραφική απεικόνιση του τρόπου λειτουργίας και της αλληλουχίας του μηχανισμού.

Όπως αναφέρθηκε παραπάνω ο μηχανισμός αντιμετώπισης που δημιουργήθηκε έχει την δυνατότητα εμφάνισης κάποιων warnings σχετικά με την ύπαρξη ευάλωτων σημείων στον κώδικα.Παρακάτω ακολουθούν σχετικές εικόνες και για τα 2 είδη.

```
WARNING: There is a vulnerable to buffer overflow command that needs to be changed in line 17
WARNING: There is a vulnerable to buffer overflow command in line 11
WARNING: There is a vulnerable to buffer overflow command in line 12
WARNING: There is a vulnerable to buffer overflow command in line 13
WARNING: There is a vulnerable to buffer overflow command in line 14
WARNING: There is a vulnerable to buffer overflow command in line 15
WARNING: There is a vulnerable to buffer overflow command in line 16
```

Σχήμα 5.2: Ειδοποιήσεις του συστήματος για την τοποθεσία των ευάλωτων εντολών.

```
WARNING: There is a vulnerable command to Format String Attack in line 13
```

Σχήμα 5.3: Ειδοποιήσεις του συστήματος για την τοποθεσία των ευάλωτων εντολών.

Εικόνες με την αντιμετώπιση κάθε επίθεσης θα είναι διαθέσιμες στις παρακάτω σελίδες

Αξίζει να σημειωθεί ότι το σύστημα αυτό προγραμματίστηκε στην πλατφόρμα Eclipse και ότι όλα τα παραδείγματα επιθέσεων ήταν στην γλώσσα C. Τέλος συνολικά η εργασία αυτή για λόγους ευκολίας έγινε στο λειτουργικό Linux Ubuntu.

5.2 Buffer Overflow

Στο Κεφάλαιο 3 έχει γίνει αναλυτικότατη περιγραφή της επίθεσης αυτής καθώς και παρουσίαση διάφορων τρόπων αντιμετώπισής της. Οι τρόποι που έχουν υλοποιηθεί σε αυτήν την εργασία είναι 2: η χρήση των canaries και η χρήση ασφαλών εντολών αντί εντολών που είναι ευάλωτες προς αυτήν την επίθεση. Όπως αναφέρθηκε και παραπάνω ο μηχανισμός που αναπτύχθηκε για την αντιμετώπιση των 2 επιθέσεων που πραγματεύεται η παρούσα εργασία έχει 2 σκέλη. Είναι σημαντικό να αναφερθεί ότι για αυτήν την επίθεση ο μηχανισμός μας καλύπτει τις εντολές : gets, strcpy, strcat, sscanf, fscanf, sprintf, scanf.

5.2.1 Αντιμετώπιση του Buffer Overflow με χρήση ασφαλών εντολών

Η χρήση ασφαλών εντολών έναντι άλλων που είναι ευάλωτες προς την επίθεση Buffer Overflow, είναι μια μέθοδος που εξ ΄ ορισμού είναι αποδοτική. Αυτό ισχύει γιατί εξ ‘ ορισμού οι εντολές αυτές έχουν σαν όρισμα το μέγεθος του buffer και δεν αποθηκεύουν τα δεδομένα εάν το μέγεθός τους δεν είναι ανάλογου μεγέθους. Η συγκεκριμένη μέθοδος έχει υλοποιηθεί και δοκιμαστεί για τις παρακάτω εντολές:

1. gets
2. strcpy
3. strcat

Οι εντολές αυτές αλλάζουν από τον μηχανισμό μας και αντικαθίστανται με τις παρακάτω:

1. fgets
2. strncpy
3. strncat

Ο μηχανισμός λοιπόν ξεκινάει την δουλειά του έχοντας σαν αρχή ότι αυτές οι εντολές θεωρούνται ύποπτες και μπορεί να εγκυμονούν κινδύνους. Με βάση την αρχή αυτή ο μηχανισμός αρχίζει και σαρώνει τον κώδικα του επιτιθέμενου με σκοπό να εντοπίσει και να απομονώσει τις εντολές αυτές. Όταν τις εντοπίσει τις απομονώνει. Σε αυτό το σημείο το πρόγραμμα του επιτιθέμενου που είναι στην διαδικασία ελέγχου έχει σαρωθεί και έχουν εντοπιστεί οι εντολές προς διόρθωση οι οποίες και έχουν απομονωθεί σε ένα νέο αρχείο. Στην συνέχεια ο μηχανισμός εστιάζει την δουλεία στο νέο αρχείο αυτό. Κάθε μία από τις εντολές που έχουν απομονωθεί τίθεται υπό επεξεργασία με σκοπό πρώτον να τροποποιηθεί το όνομα της και να μετονομαστεί στην ασφαλή της εκδοχή και δεύτερον να τροποποιηθούν τα ορίσματα της καθώς ή ασφαλής της εκδοχή έχει διαφορετικό αριθμό και είδος ορισμάτων. Έπειτα οι τροποποιημένες εντολές αυτές γράφονται σε ένα νέο αρχείο από το οποίο μία - μία τοποθετούνται μαζί και με τις άλλες εντολές που υπήρχαν στο τελικό ασφαλές αρχείο που παράγει ο μηχανισμός . Τέλος, το πρόγραμμα του επιτιθέμενου είναι έτοιμο προς εκτέλεση χωρίς όμως να εγκυμονεί κινδύνους πλέον. Είναι σημαντικό να αναφερθεί ότι παρά την όποια διορθωτική κίνηση η λειτουργία του προγράμματος είναι ακριβώς η ίδια.

5.2.2 Αντιμετώπιση του Buffer Overflow με χρήση canaries

Ο δεύτερος τρόπος που έχει υλοποιηθεί για την αντιμετώπιση της επίθεσης Buffer Overflow είναι η χρήση canaries. Η αναλυτική περιγραφή της μεθόδου αυτής έχει γίνει στο Κεφάλαιο 3. Εν συντομία τα canaries είναι μεταβλητές όπου οι τιμές τους είναι ακέραιες και το πιο σημαντικό είναι ότι είναι και τυχαίες. Τα canaries λοιπόν τοποθετούνται σε γειτονικές θέσεις μνήμης από τους πίνακες και εάν κάποιος από αυτούς υπερχειλίσει αυτόματα αλλάζει η τιμή του αντίστοιχου canary και έτσι γίνεται αντιληπτή η επίθεση. Η διαδικασία που ακολουθεί ο μηχανισμός που αντιμετωπίζει το Buffer Overflow είναι και σε αυτήν την περίπτωση συγκεκριμένη. Αρχικά σαρώνεται ο κώδικας του επιτιθέμενου

με σκοπό να βρεθούν όλες οι δηλώσεις πινάκων, που στην ουσία είναι οι δομές που μπορεί να υπερχειλίσουν. Από την στιγμή που ξέρουμε ποιες είναι οι δηλώσεις αυτές, είναι γνωστός και ο αριθμός τους, επομένως είναι γνωστός και ο αριθμός των canaries. Αυτό συμβαίνει γιατί η αναλογία των canaries με τους πίνακες είναι ένα προς ένα, δηλαδή κάθε canary είναι 'υπεύθυνο' για έναν πίνακα. Έπειτα ακολουθεί μια διαδικασία δόμησης του προγράμματος. Τα canaries με τους πίνακες ορίζονται μέσα σε μια δομή (struct). Αυτό συμβαίνει γιατί ο compiler δεν τοποθετεί, αναγκαστικά, σε γειτονικές θέσεις μνήμης μεταβλητές που έχουν δηλωθεί δίπλα - δίπλα. Επιπλέον οι μεταβλητές που έχουν αρχικοποιηθεί δεν τοποθετούνται στην ίδια θέση με αυτές που δεν έχουν αρχικοποιηθεί. Η δήλωση πινάκων και canaries μέσα σε μία δομή (είτε όλα μη αρχικοποιημένα, είτε αρχικοποιημένα) εξασφαλίζει ότι όλα τα περιεχόμενα της δομής τοποθετούνται σε ένα συνεχόμενο πεδίο μνήμης και σε γειτονικές θέσεις μνήμης, που είναι και το ζητούμενο στην περίπτωση μας (η τοποθέτηση των δεδομένων εξαρτάται επίσης και από τα alignment και το padding της δομής). Βεβαίως η τεχνική αυτή αποτελεί πολύ καλύτερη από την δήλωση των canaries έξω από μια δομή αλλά παρουσιάζει και ένα μειονέκτημα το οποίο είναι το εξής. Εφόσον η μνήμη είναι χωρισμένη σε τετράδες bytes και έτσι τοποθετούνται τα δεδομένα σε αυτήν, εάν ένας buffer δεν έχει μέγεθος πολλαπλάσιο του τέσσερα θα μείνουν κάποιες θέσεις μνήμης κενές ανάμεσα σε αυτόν και στο canary και έτσι ενδέχεται η υπερχείλιση να μην εντοπιστεί εάν τα παραπανίσια δεδομένα είναι τόσων ή και λιγότερων bytes από τα bytes των κενών θέσεων. Μετά από την διαδικασία ορισμού των πινάκων και των canaries, ακολουθεί μια επιπλέον σάρωση του προγράμματος η οποία εντοπίζει όλες τις εντολές που δέχονται δεδομένα από τον έξω κόσμο και τα τοποθετούν μέσα στους πίνακες. Ο μηχανισμός στο σημείο αυτό επεμβαίνει στον κώδικα και τοποθετεί μετά από κάθε τέτοια εντολή έλεγχο των τιμών των canaries. Εάν αυτός ο έλεγχος γίνει και το αποτέλεσμα του είναι, ότι η τιμή κάποιου από τα canaries είναι διαφορετική από αυτήν που του έχει δοθεί αρχικά σταματάει την εκτέλεση του προγράμματος καθώς έχει γίνει επίθεση Buffer Overflow.

5.2.3 Κόστη Αντιμετώπισης (Overheads)

Κάθε μέθοδος η οποία εφαρμόζεται για την επίλυση ενός προβλήματος έχει ένα κόστος. Με τον όρο κόστος εννοούμε των αριθμό των εντολών που χρησιμοποιούνται, την μνήμη που καταλαμβάνει μια τέτοια διεργασία καθώς και τον χρόνο που χρειάζεται για να ολοκληρωθεί μια τέτοιου είδους διαδικασία.

Canaries

Η χρήση των Canaries θεωρείται μια πολύ αξιόπιστη λύση για την αντιμετώπιση του Buffer Overflow. Όσον αφορά το κόστος τους προφανώς για μικρά σε μέγεθος προγράμματα όπως αυτά που παρατίθενται παρακάτω το κόστος της συγκεκριμένης μεθόδου είναι αμελητέο. Αντικειμενικά όμως, εάν αναλογιστούμε ότι υλοποιείται μια δομή για την αποθήκευση πινάκων και των αντιστοιχών τους Canaries που σε αριθμό είναι όσοι και οι πίνακες χρειαζόμαστε παραπανίσια μνήμη για τόσους ακέραιους αριθμούς, όσος είναι και ο αριθμός των πινάκων. Επίσης χρειαζόμαστε και παραπανίσια μνήμη για τις κλήσεις την συνάρτησης ελέγχου των Canaries. Είναι αντιληπτό ότι η συγκεκριμένη μέθοδος δεν αποτελεί πολύ αποδοτική από άποψη μνήμης. Επίσης ο αριθμός των εντολών είναι αυξημένος καθώς ελέγχονται όλα τα Canaries σε κάθε είσοδο ενός δεδομένου. Όσον αφορά την ταχύτητα η μέθοδος των Canaries, στα παραδείγματά μας ήταν γρήγορη. Γενικότερα όμως, η ταχύτητα της εξαρτάται και από τον αριθμό των πινάκων προς έλεγχο. Θεωρητικά, αποτελεί μια γρήγορη μέθοδο, εφόσον κανένας έλεγχος δεν είναι ιδιαίτερα πολύπλοκος και χρονοβόρος.

Χρήση Ασφαλών εντολών

Η χρήση των ασφαλών εντολών θεωρείται επίσης μια καλή λύση αν και δεν είναι πάντα εύκολο να υλοποιηθεί καθώς δεν έχουν όλες οι εντολές μια ασφαλή εκδοχή τους και για αυτές είναι απαραίτητο να υλοποιούνται άλλες μέθοδοι ελέγχου. Η μέθοδος αυτή δεν περιλαμβάνει παραπάνω εντολές καθώς ο στόχος

της είναι η εύρεση των ευάλωτων εντολών και η αντικατάσταση τους με τις αντίστοιχες ασφαλείς. Όσον αφορά την μνήμη η μέθοδος αυτή δεν έχει ιδιαίτερες απαιτήσεις καθώς και πάλι απλά αλλάζει η εντολή και μπορεί να προστεθούν ελάχιστα νέα δεδομένα (π.χ μέγεθος δεδομένων που εισάγονται). Τέλος η ταχύτητα υλοποίησης θεωρείται σχετικά μεγάλη καθώς δεν απαιτούνται πολλές και πολύπλοκες διεργασίες για την αντικατάσταση των εντολών αυτών.

Κόστος σε Assembly

Στο παράδειγμα το οποίο παρουσιάζεται παρακάτω γίνανε οι μετρήσεις εντολών σε Assembly, για να γίνει αντιληπτό το μέγεθος του κόστους που παραπάνω αναλύθηκε λεκτικά. Πιο συγκεκριμένα:

Αριθμός Εντολών Assembly ανά περίπτωση		
Περίπτωση	Αριθμός Εντολών Assembly	Αριθμός Εντολών Assembly με -O2 βελτιστοποίηση
Πρίν την διόρθωση	103	73
Μετά την διόρθωση	159	135
Επιπλέον συνάρτηση	34	33

Πίνακας 5.1: Πίνακας με αριθμό εντολών Assembly για το παράδειγμα του Buffer Overflow.

Όπως γίνεται αντιληπτό η συγκεκριμένη μέθοδος έχει σαν αποτέλεσμα οι εντολές της ασφαλούς εκδοχής του κώδικα να είναι 56 παραπάνω από αυτές του αρχικού κώδικα, γεγονός που την καθιστά πιο βεβαρημένη όσον αφορά τους πόρους που καταναλώνει σε έναν υπολογιστή. Στην βελτιστοποιημένη περίπτωση είναι 62 παραπάνω οι εντολές του ασφαλή κώδικα, αλλά συγκριτικά με την μη βελτιστοποιημένη περίπτωση τα αντίστοιχα νούμερα είναι μικρότερα

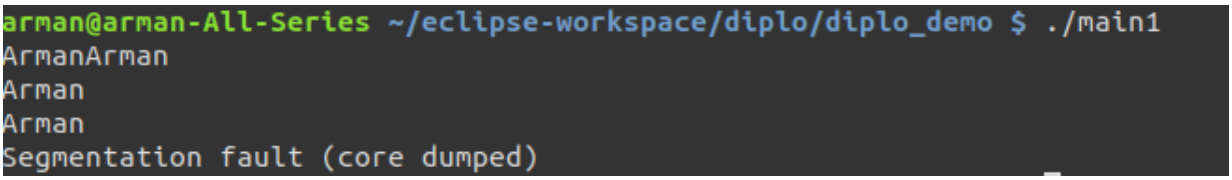
5.2.4 Παράδειγμα Αντιμετώπισης του Buffer Overflow

Σε αυτό το σημείο ήρθε η ώρα να γίνει ένα παράδειγμα και να δούμε πώς ο μηχανισμός αντιμετωπίζει ένα πρόγραμμα που είναι ευάλωτο σε επιθέσεις Buffer Overflow.

Παρακάτω ακολουθεί ένα παράδειγμα:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void func(char* f,char* g);
void func(char* f,char* g){
char b[9],n[4];
scanf("%s",b);
sprintf(n,"%s",b);
scanf("%s",f);
scanf("%s",g);
strcpy(b, "aa");
return;
}
int main(int argc, char** argv){
char dest[70],sD[10];
double scout = 5340;
float po = 0.64;
func(dest,sD);
return 0;
}
```

Ας δούμε τώρα πώς αντιδράει σε ένα ενδεχόμενο Buffer Overflow.



Σχήμα 5.4: Σε ενδεχόμενο Buffer Overflow το πρόγραμμα 'κρασάρει' και η επίθεση δεν διαχειρίζεται.

Ας δούμε τώρα πώς ο μηχανισμός μας τροποποιεί το πρόγραμμα και το κάνει ασφαλές.

```
#include<stdint.h>
#include"check_func.h"
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void func(char* f,char* g);
struct t1{
        char b[9];
        uint32_t canary_1;
        char n[4];
        uint32_t canary_2;
        char dest[70];
        uint32_t canary_3;
        char sD[10];
        uint32_t canary_4;
}s1;

void func(char* f,char* g){
s1.canary_1=846145399;
s1.canary_2=214019545;
s1.canary_3=75223248;
s1.canary_4=556868748;
scanf("%s",s1.b);
CheckForBufOvf(s1.canary_1,846145399);
CheckForBufOvf(s1.canary_2,214019545);
CheckForBufOvf(s1.canary_3,75223248);
CheckForBufOvf(s1.canary_4,556868748);
sprintf(s1.n,"%s",s1.b);
CheckForBufOvf(s1.canary_1,846145399);
CheckForBufOvf(s1.canary_2,214019545);
CheckForBufOvf(s1.canary_3,75223248);
CheckForBufOvf(s1.canary_4,556868748);
scanf("%s",f);
CheckForBufOvf(s1.canary_1, 846145399);
CheckForBufOvf(s1.canary_2, 214019545);
CheckForBufOvf(s1.canary_3, 75223248);
CheckForBufOvf(s1.canary_4, 556868748);
scanf("%s",g);
CheckForBufOvf(s1.canary_1, 846145399);
CheckForBufOvf(s1.canary_2, 214019545);
CheckForBufOvf(s1.canary_3, 75223248);
CheckForBufOvf(s1.canary_4, 556868748);
strncpy(s1.b, "aa",sizeof(s1.b));
s1.b[sizeof(s1.b)-1]='\0';
return;
}
int main(int argc, char** argv){
s1.canary_1=846145399;
s1.canary_2=214019545;
s1.canary_3=75223248;
s1.canary_4=556868748;
double scout = 5340;
float po = 0.64;
func(s1.dest,s1.sD);
return 0;
}
```

Όπως βλέπουμε στο τροποποιημένο πρόγραμμα έχουν προστεθεί τα canaries και η εντολή strcpy έχει αντικατασταθεί με την ασφαλή της έκδοση.

Ας δούμε τώρα πώς αντιδράει σε ένα ενδεχόμενο Buffer Overflow τροποποιημένο πρόγραμμα.

```
arman@arman-All-Series ~/eclipse-workspace/diplo/diplo_demo $ ./main
ArmanArman
buffer overlow detected
Aborted (core dumped)
```

Σχήμα 5.5: Σε ενδεχόμενο Buffer Overflow το πρόγραμμα διαχειρίζεται και εντοπίζει το πρόβλημα και σταματάει.

Η συνάρτηση CheckForBufOvf είναι αυτή η οποία εντοπίζει το αν έγινε Buffer Overflow και έχει την εξής δομή:

```
#include "check_func.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

void CheckForBufOvf(uint32_t x, uint32_t y){

        if(x!=y){printf("buffer overlow detected\n");
                        abort();
        }
}
```

5.3 Format String Attacks

Στο Κεφάλαιο 4 αναπτύχθηκε και περιγράφηκε εκτενώς η επίθεση Format String Attacks. Σε αντίθεση με την επίθεση Buffer Overflow, η επίθεση Format String Attacks έχει ένα αρκετές εκδοχές και δεν είναι μονοδιάστατη. Παρακάτω θα παρουσιαστούν οι εκδοχές που αντιμετωπίζει ο μηχανισμός μας καθώς και ο τρόπος αντιμετώπισης τους. Είναι σημαντικό να αναφερθεί ότι, ο μηχανισμός καλύπτει όλες τις συναρτήσεις της οικογένειας Format Functions (fprintf, printf, sprintf, snprintf, vprintf, vsprintf, vsnprintf, vfprintf).

5.3.1 Format String Parameters μέσα σε buffer

Έστω λοιπόν ότι δεχόμαστε ένα πρόγραμμα το οποίο έχει κάποιους πίνακες χαρακτήρων που δέχονται τιμές που εμπεριέχουν Format String Parameters και όχι μια απλή συμβολοσειρά ή και κάποιους που είναι ορισμένοι και αρχικοποιημένοι με τέτοιου είδους τιμές, όπως για παράδειγμα το %x. Το γεγονός αυτό από μόνο του προφανώς δεν δημιουργεί κάποιο πρόβλημα στην εκτέλεση του προγράμματός μας. Αλλά στο ενδεχόμενο όπου αυτός ο πίνακας θα χρησιμοποιηθεί μέσω κάποιας συνάρτησης από την οικογένεια συναρτήσεων printf που δεν θα έχει ορισμένο , από τον προγραμματιστή, τον παράγοντα format argument, θα παρουσιαστεί πρόβλημα. Το θέμα θα είναι ότι όταν η συνάρτηση εμφάνισης δεδομένων θα συναντήσει τα Format String Parameters τότε αντί για τον χαρακτήρα θα εμφανίσει τα περιεχόμενα μιας θέσης μνήμης που θα βρει από το Format String Parameter. Αυτό αυτομάτως αποτελεί πρόβλημα καθώς τα δεδομένα σε διάφορες θέσεις μνήμης θα είναι εκτεθειμένα στα μάτια κάποιου επιτιθέμενου. Το συγκεκριμένο είδος Format String Attack θα αντιμετωπισθεί από τον μηχανισμό με την μέθοδο του ελέγχου των εισόδων/δεδομένων στους πίνακες ενός προγράμματος.

Αντιμετώπιση των Format String Attacks με έλεγχο δεδομένων

Η περιγραφή της μεθόδου αυτής έχει γίνει στο Κεφάλαιο 4. Έστω λοιπόν ότι το πρόγραμμα προς έλεγχο είναι το ακόλουθο.

```
#include  <stdio.h>
#include  <string.h>
#include  <stdlib.h>
#include <stdarg.h>

int main (int argc, char **argv)
{
char a[20];
char b[20]="Frank";
char str[20];
/*ssss*/
gets(a);

pr
intf(

a);
snprintf ( str, sizeof str, argv [1] ) ;
str[sizeof(str)-1]='\0';
printf(str);
printf("Hello%s",b);
return 0;
}
```

Παρατηρούμε ότι το πρόγραμμα δέχεται εισόδους από χρήστη οι οποίες μπορεί να εμπεριέχουν Format String Parameters. Το γεγονός αυτό, εάν δεν υπήρχαν οι συναρτήσεις της οικογένειας εντολών printf, δεν θα δημιουργούσε πρόβλημα, από την στιγμή όμως που υπάρχουν και πόσο μάλλον όταν και ορισμένες από αυτές δεν εμπεριέχουν τα κατάλληλα Format String Parameters (%s), στο format argument, δημιουργούν το πρόβλημα που αναφέρθηκε παραπάνω.

Σε αυτό το σημείο έρχεται ο μηχανισμός που φτιάχτηκε για να επέμβει. Ο μηχανισμός λοιπόν εντοπίζει τις συναρτήσεις της οικογένειας εντολών printf που χρησιμοποιούν τους πίνακες και αυτομάτως τοποθετεί ένα κάλεσμα μιας συνάρτησης, που φτιάχτηκε για τις ανάγκες τις περίπτωσης αυτής, η οποία έχει σαν σκοπό να απαλλάξει τους πίνακες αυτούς από την παρουσία των Format String Parameters, εάν φυσικά υπάρχουν. Παρατηρούμε επίσης ότι στον αρχικό κώδικα η πρώτη εντολή printf έχει κενά ανάμεσα της. Όπως θα δούμε και στον τελικό κώδικα ο μηχανισμός δεν έχει επηρεαστεί από το γεγονός αυτό καθώς στοιχίζει τον κώδικα.

Πλέον το πρόγραμμα μετατρέπεται ως εξής:

```
#include"remove.h"
#include  <stdio.h>
#include  <string.h>
#include  <stdlib.h>
#include <stdarg.h>
int main (int argc, char **argv){
char a[20];
char b[20]="Frank";
char str[20];
/*ssss*/
gets(a);
func(1,a);
printf(a);
func(1, argv [1] ) ;
snprintf ( str, sizeof str, argv [1] ) ;
str[sizeof(str)-1]='\0';
func(1,str);
printf(str);
func(1,b);
printf("Hello%s",b);
return 0;
}
```

Η συνάρτηση func η οποία έχει σαν σκοπό να απαλλάξει τους πίνακες από τα Format String Parameters είναι η ακόλουθη

```
#include "remove.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include <string.h>
void func(int num,...){
  va_list valist;
  char *a, *b;
  int i=0;
  size_t j;
  int count=0;
  va_start(valist,num);
  for (i = 0; i < num; i++) {
    char *t = va_arg(valist,char *);
        size_t len=strlen(t);
        for(j=0;t[j]!='\0';j++){
                if(t[j]=='%'){
                        if(t[j+1]=='s'){break;}
                        else if(t[j+1]=='x'){break;}
                        else if(t[j+1]=='d'){break;}
                        else if(t[j+1]=='f'){break;}
                        else if(t[j+1]=='p'){break;}
                        else if(t[j+1]=='u'){break;}
                        else if(t[j+1]=='c'){break;}
                        else if(t[j+1]=='n'){break;}
                        else if(t[j+1]=='0'){break;}
                        else if(t[j+1]=='1'){break;}
                        else if(t[j+1]=='2'){break;}
                        else if(t[j+1]=='3'){break;}
                        else if(t[j+1]=='4'){break;}
                        else if(t[j+1]=='5'){break;}
                        else if(t[j+1]=='6'){break;}
                        else if(t[j+1]=='7'){break;}
                        else if(t[j+1]=='8'){break;}
                        else if(t[j+1]=='9'){break;}
                        else{}
                        }
        }
        t[j]='\0';
   }
   va_end(valist);
}
```

Τα αποτελέσματα της συγκεκριμένης μεθόδου είναι εμφανή και στις εκτελέσεις του προγράμματος πριν την διόρθωση και μετά την διόρθωση.Παρακάτω ακολουθούν και εικόνες που αποδεικνύουν τον παραπάνω ισχυρισμό.

Πρίν την διόρθωση η εκτέλεση του προγράμματος ήταν η εξής:

```
arman@arman-All-Series ~/eclipse-workspace/formatstrbuf $ ./main1 "Bo%x%xb"
A%x%x
A782578258fdc8890Bo01bHelloFrankarman@arman-All-Series ~/eclipse-workspace/form
tstrbuf $
```

Σχήμα 5.6: Αποτελέσματα πριν την διόρθωση του προγράμματος.

Μετά την διόρθωση η εκτέλεση του προγράμματος είναι η εξής:

```
arman@arman-All-Series ~/eclipse-workspace/formatstrbuf $ ./main "Bo%x%xb"
A%x%x
ABoHelloFrankarman@arman-All-Series ~/eclipse-workspace/formatstrbuf $
```

Σχήμα 5.7: Αποτελέσματα μετά την διόρθωση του προγράμματος.

Όπως είναι προφανές η εκτέλεση του προγράμματος αφότου διορθώθηκε δεν παρουσιάζει κάποιο πρόβλημα. Η μέθοδος αυτή που χρησιμοποιήθηκε όμως παρουσιάζει δύο μειονεκτήματα τα οποίο είναι πρώτον ότι 'κόβει' την συμβολοσειρά εκεί που βρίσκει το Format String Parameter και έτσι οι συμβολοσειρές τροποποιούνται καθώς μένουν λειψές και δεύτερον ότι εφαρμόζεται μόνο σε περιπτώσεις που δεν υπάρχει μέσα στην Format Function, format argument (όταν δηλαδή το δεδομένο που υπάρχει ή δεχόμαστε σαν είσοδο χρησιμοποιείται από την Format Function, λανθασμένα, σαν format argument). Οι υπόλοιπες περιπτώσεις καλύπτονται από την μέθοδο 'ελέγχου του ορθού ορισμού των Format Functions', που θα παρουσιαστεί παρακάτω.

Κόστος της Αντιμετώπισης των Format String Attacks με έλεγχο δεδομένων

Η συγκεκριμένη μέθοδος όπως αναφέρθηκε και παραπάνω δεν αποτελεί την ιδανική λύση καθώς παρουσιάζει τα δύο μειονεκτήματα που αναφέρθηκαν παραπάνω. Όσον αφορά το κόστος της μεθόδου αυτής, με μια πρώτη ματιά φαίνεται ότι δεν εμπεριέχει πολλές παραπάνω εντολές καθώς για κάθε μια Format Function που είναι ευάλωτη στην επίθεση Format String Attack προστίθεται ένα κάλεσμα συνάρτησης. Παρ΄όλα αυτά κάθε κλήση της συνάρτησης προϋποθέτει και την εκτέλεση όλων των εντολών της, άρα τελικά προστίθενται αρκετές νέες εντολές. Από πλευράς μνήμης, μπορεί να θεωρηθεί απαιτητική μέθοδος καθώς προστίθενται τα δεδομένα της συνάρτησης καθώς οι κλήσεις της μαζί με τα ήδη υπάρχοντα δεδομένα του κώδικα. Τέλος σχετικά με την ταχύτητα η μέθοδος αυτή μπορεί να γίνει αργή για μεγάλα προγράμματα που κάθε Format String Function εμφανίζει πολλές συμβολοσειρές ταυτόχρονα που η κάθε μια από αυτές είναι μεγάλη. Η παρατήρηση αυτή μπορεί να γίνει κατανοητή από την ύπαρξη της εμφωλευμένης επαναληπτικής δομής μέσα στην συνάρτηση func.

Πιο αναλυτικά:

Αριθμός Εντολών Assembly ανά περίπτωση		
Κατάσταση	Αριθμός Εντολών Assembly	Αριθμός Εντολών Assembly με -O2 βελτιστοποίηση
Πρίν την διόρθωση	66	58
Μετά την διόρθωση	88	71
Επιπλέον συνάρτηση	288	94

Πίνακας 5.2: Πίνακας με αριθμό εντολών Assembly για το παράδειγμα του Format String Attacks με καθαρισμό από Format String Parameters σε buffers.

Όπως γίνεται αντιληπτό η συγκεκριμένη μέθοδος έχει σαν αποτέλεσμα οι εντολές της ασφαλούς εκδοχής του κώδικα να είναι παραπάνω από αυτές του αρχικού κώδικα, γεγονός που την καθιστά πιο βεβαρημένη όσον αφορά τους πόρους που καταναλώνει σε έναν υπολογιστή. Πρέπει να υπολογίσουμε βέβαια και τις 288 εντολές της παραπανίσιας συνάρτησης (94 στην βελτιστοποιημένη μέτρηση) οι οποίες προσθέτουν και αυτές παραπανίσιο φόρτο.

5.3.2 Λανθασμένος ορισμός της συνάρτησης syslog

Μια ακόμη περίπτωση των Format String Attacks είναι και η λανθασμένη χρήση συναρτήσεων όπως είναι η syslog. Η συνάρτηση syslog δέχεται σαν όρισμα κάποιο μήνυμα το οποίο και το καταγράφει στο log του συστήματος. Τι γίνεται όμως εάν η εντολή δεν εμπεριέχει Format String Parameters και το μήνυμα εμπεριέχει· Τότε η syslog περνάει το μήνυμα αυτό στο σύστημα και αυτόματα στην θέση του ή των Format String Parameters εμφανίζονται δεδομένα σε κάποια θέση μνήμης που έχει ανασύρει ο υπολογιστής από την παρουσία των Format String Parameters. Η αντιμετώπιση του προβλήματος αυτού είναι αρκετά απλή και είναι η σωστή χρήση και σύνταξη της συγκεκριμένης εντολής. Σε καμία περίπτωση η συνάρτηση syslog δεν θα πρέπει να χρησιμοποιείται χωρίς ορθά ορισμένο format argument από τον προγραμματιστή.

Αντιμετώπιση των Format String Attacks στην συνάρτηση syslog με διόρθωση σύνταξης

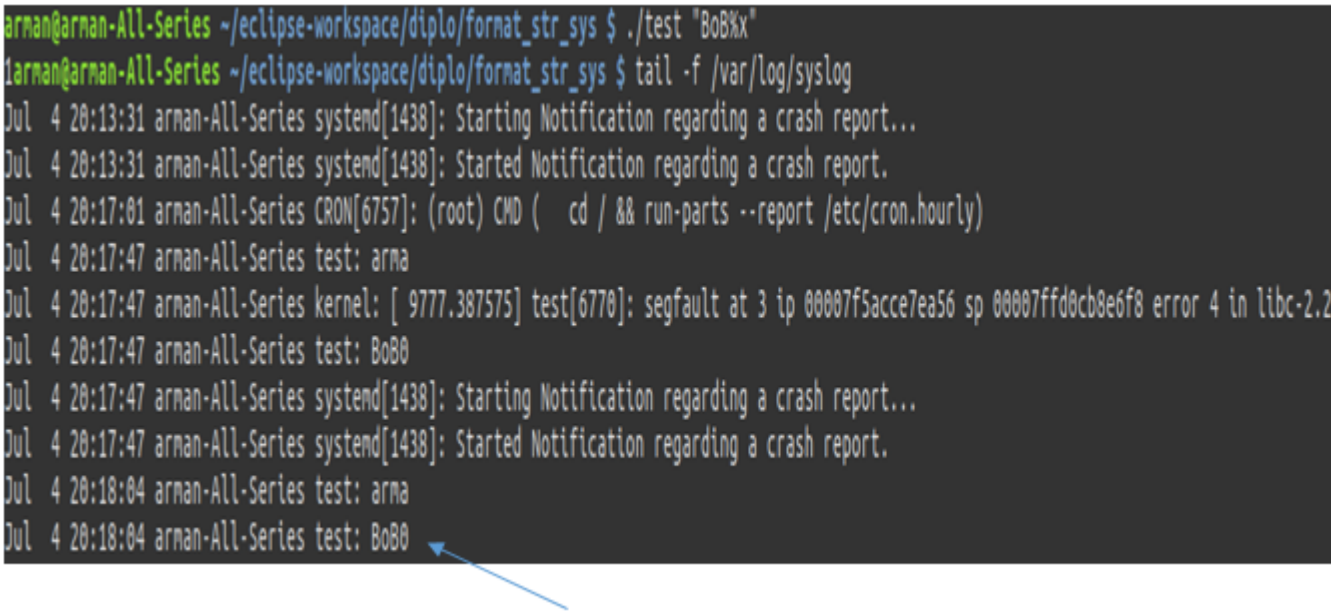
Όπως είπαμε παραπάνω η σύνταξη της συνάρτησης αυτής είναι ένας τρόπος αποφυγής Format String Attacks που σχετίζονται με αυτήν. Έστω λοιπόν πως έχουμε ένα κομμάτι κώδικα όπως ο παρακάτω.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <regex.h>
#include <syslog.h>

int main(int argc , char **argv){
char buf[100]="arma" , a[10] , b[10];
int x=1;
printf("%d",x);
int y=1;
syslog(LOG_ERR, "%s" ,buf);
int z=10;
syslog(LOG_ERR, argv[1]);
return 0;
}
```

Στην περίπτωση αυτή παρατηρούμε πως η δεύτερη syslog δεν έχει Format String Parameter και άρα ο υπολογιστής δεν ξέρει την μορφή της εισόδου-μηνύματος και άρα δεν ξέρει και πως να την διαχειριστεί εμφανίζοντας περιεχόμενο μιας θέσης μνήμης. Αυτό το οποίο θα καταγράψει στο σύστημα λοιπόν όταν το πρόγραμμα αυτό θα εκτελεστεί θα είναι το εξής:

Όπως βλέπουμε το σύστημα δεν ξέρει πώς να διαχειριστεί το Format String Parameter που τοποθετείται στο δεδομένο που δέχεται.



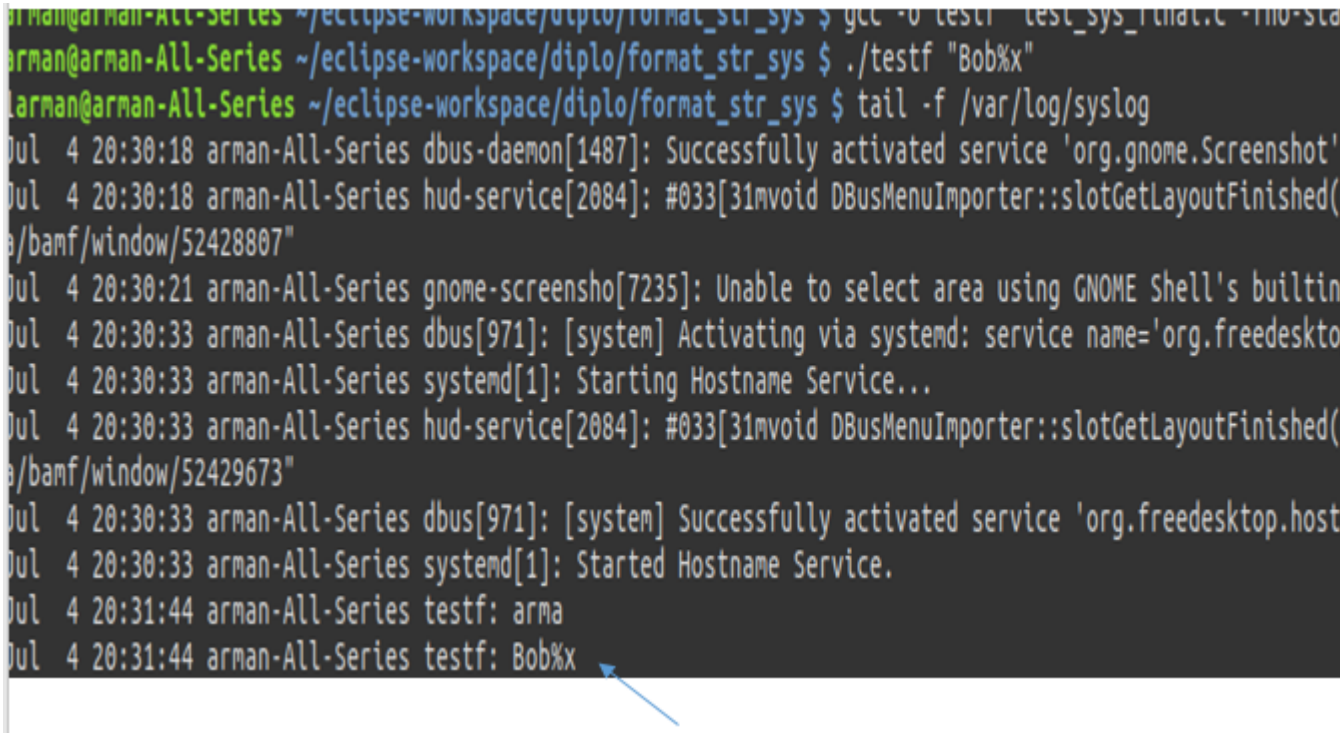
Σχήμα 5.8: Αποτελέσματα πριν την διόρθωση του προγράμματος.

Σε αυτό το σημείο επεμβαίνει ο μηχανισμός αντιμετώπισης των Format String Attacks εντοπίζοντας ποια συνάρτηση δεν έχει Format String Parameter και τοποθετώντας σε αυτήν όσα χρειάζεται για να είναι ορθά ορισμένη. Κατά αυτόν τον τρόπο κώδικας διαμορφώνεται ως εξής:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <regex.h>
#include <syslog.h>

int main(int argc , char **argv){
char buf[100]="arma" , a[10];
char  b[10];
int x=1;
printf("%d",x);
int y=1;
syslog(LOG_ERR, "%s" ,buf);
int z=10;
syslog(LOG_ERR, "%s" ,argv[1]);
return 0;
}
```

Με αυτόν τον τρόπο βλέπουμε πώς πλέον το δεύτερο syslog έχει Format String Parameter. Η εκτέλεση του προγράμματος πλέον γίνεται ως εξής:



Σχήμα 5.9: Αποτελέσματα μετά την διόρθωση του προγράμματος.

Μετά την παρέμβαση του μηχανισμού η συνάρτηση syslog διαχειρίζεται το Format String Parameter σαν έναν απλό χαρακτήρα εντός μιας συμβολοσειράς και δεν προκαλείται κάποια ανωμαλία στην καταγραφή του μηνύματος στο σύστημα.

Σε αυτό το σημείο πρέπει να σημειωθεί ότι η μέθοδος αυτή όπως και η παραπάνω καλύπτει και την περίπτωση όπου το όρισμα format δεν είναι μέσα σε εισαγωγικά αλλά είναι με την μορφή συμβολοσειράς μέσα σε πίνακα χαρακτήρων (δηλαδή τύπου: const char *buffer). Βέβαια σε αυτήν την περίπτωση ακολουθείται μια άλλη διαδικασία μέσω μιας συνάρτησης που δημιουργήθηκε για αυτήν την περίπτωση. Η συνάρτηση αυτή είναι η εξής:

```
#include "check.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include <string.h>
void countf(int num,...){
  va_list valist;
  char *a, *b;
  int i=0;
  size_t j;
  int count=0;
  va_start(valist,num);

  for (i = 0; i < 1; i++) {
    char *t = va_arg(valist,char *);
        size_t len=strlen(t);
        for(j=0;t[j]!='\0';j++){
                if(t[j]=='%'){
                        if(t[j+1]=='s'){count++;}
                        else if(t[j+1]=='x'){count++;}
                        else if(t[j+1]=='d'){count++;}
                        else if(t[j+1]=='f'){count++;}
                        else if(t[j+1]=='p'){count++;}
                        else if(t[j+1]=='u'){count++;}
                        else if(t[j+1]=='c'){count++;}
                        else if(t[j+1]=='n'){count++;}
                        else if(t[j+1]=='0'){count++;}
                        else if(t[j+1]=='1'){count++;}
                        else if(t[j+1]=='2'){count++;}
                        else if(t[j+1]=='3'){count++;}
                        else if(t[j+1]=='4'){count++;}
                        else if(t[j+1]=='5'){count++;}
                        else if(t[j+1]=='6'){count++;}
                        else if(t[j+1]=='7'){count++;}
                        else if(t[j+1]=='8'){count++;}
                        else if(t[j+1]=='9'){count++;}
                        else{}
                        }
        }
        if(count>num){abort();}
   }
   va_end(valist);
}
```

Η συνάρτηση αυτή εάν εντοπίσει ότι ο αριθμός των Format String Parameters στο όρισμα format της συνάρτησης syslog είναι μεγαλύτερος από τον αριθμό των ορισμάτων της αυτομάτως εκτελεί την εντολή abort();.Παρακάτω φαίνεται και ένα παράδειγμα χρήσης της συνάρτησης αυτής που έχει ονομαστεί countf.Ακολουθεί ο κώδικας πριν την διόρθωση .

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <regex.h>
#include <syslog.h>

int main(int argc , char **argv){
char buf[100]="arma" , a[10]="Hi" , b[10]="Hello";
const char *f="This is %s%s";
const char *g="This is %x%x";
int y=1;
syslog(LOG_ERR,"a",a);
syslog(LOG_ERR,g);
int z=10;
syslog(LOG_ERR, f,argv[1],argv[1]);
syslog(LOG_ERR,argv[1]);
return 0;
}
```

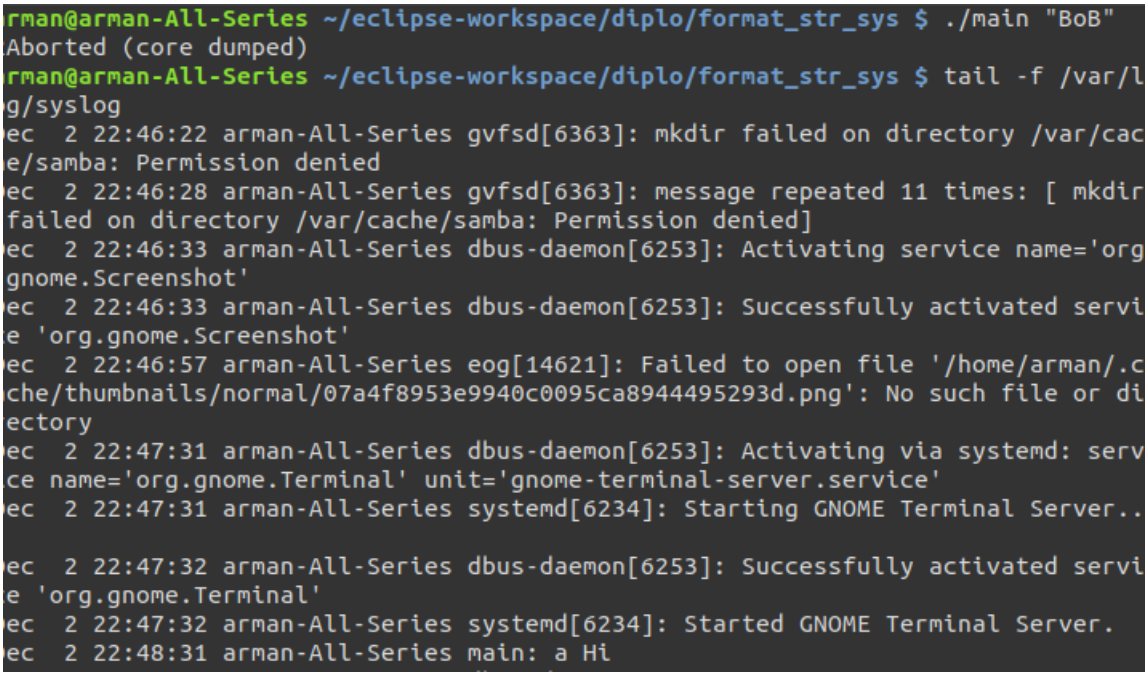
Ακολουθεί ο κώδικας μετά την διόρθωση .

```
#include"check.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <regex.h>
#include <syslog.h>
int main(int argc , char **argv){
char buf[100]="arma" , a[10]="Hi" , b[10]="Hello";
const char *f="This is %s%s";
const char *g="This is %x%x";
int y=1;
syslog(LOG_ERR,"a %s",a);
countf(0,g);
syslog(LOG_ERR,g);
int z=10;
countf(2,f);
syslog(LOG_ERR,f,argv[1],argv[1]);
syslog(LOG_ERR,"%s",argv[1]);
return 0;
}
```

Σε αυτό το δεύτερο παράδειγμα, όταν τρέχει ο κώδικας τα αποτελέσματα είναι τα παρακάτω.

```
arman-All-Series systemd[6234]: Started GNOME Terminal Server.
arman-All-Series main: a
arman-All-Series main: This is 116d7b010
arman-All-Series main: This is BoBBoB
arman-All-Series main: BoB
```

Σχήμα 5.10: Αποτελέσματα πρίν την διόρθωση του προγράμματος.



Σχήμα 5.11: Αποτελέσματα μετά την διόρθωση του προγράμματος.

Κόστος της Αντιμετώπισης του λανθασμένου ορισμού της συνάρτησης syslog

Το παράδειγμα αυτό επειδή ακριβώς αναφέρεται σε μια συγκεκριμένη εντολή και άρα δεν γενικεύεται για ένα σύνολο περιπτώσεων έχει σχετικά μικρό κόστος. Όσον αφορά την μνήμη δεν αποτελεί απαιτητική μέθοδο καθώς δεν κάνει χρήση νέων δεδομένων ή συναρτήσεων. Επιπλέον δεν απαιτεί την χρήση παραπανίσιων εντολών για να πλαισιώσουν την προσπάθεια αντιμετώπισης της συγκεκριμένης ευάλωτης εντολής. Η ταχύτητα εκτέλεσης της μεθόδου αυτής εξαρτάται καθαρά από το μέγεθος του προγράμματος και την τοποθεσία της ευάλωτης εντολής syslog εάν αυτή υπάρχει. Η τοποθέτηση του Format String Parameter δεν αποτελεί χρονοβόρα διαδικασία. Πρέπει να σημειωθεί όμως ότι ακόμα και το πρόγραμμα να είναι μεγάλο η ταχύτητα εκτέλεσης δεν αλλάζει δραματικά.

Πιο αναλυτικά:

Αριθμός Εντολών Assembly ανά περίπτωση		
Κατάσταση	Αριθμός Εντολών Assembly	Αριθμός Εντολών Assembly με -O2 βελτιστοποίηση
Πρίν την διόρθωση	73	55
Μετά την διόρθωση	73	55
Επιπλέον συνάρτηση	293	109

Πίνακας 5.3: Πίνακας με αριθμό εντολών Assembly για το 1ο παράδειγμα του Format String Attacks με διόρθωση της λανθασμένης χρήσης της συνάρτησης syslog.

Όπως γίνεται αντιληπτό η συγκεκριμένη μέθοδος έχει σαν αποτέλεσμα ο ασφαλής κώδικας που παράγεται από τον μηχανισμό να έχει ακριβώς τις ίδιες εντολές με τον προηγούμενο. Βέβαια εάν το παράδειγμα αλλάξει και αναγκαστεί ο μηχανισμός να αντιμετωπίσει το ευάλωτο κομμάτι με την συνάρτηση countf(σελ. 69) τότε ο αριθμός εντολών Assembly θα αυξηθεί. Επίσης θα υπάρχει μια σχετικά μεγάλη επιβάρυνση λόγω του των 293 εντολών της περαιτέρω συνάρτησης.

Πιο συγκεκριμένα:

Αριθμός Εντολών Assembly ανά περίπτωση		
Κατάσταση	Αριθμός Εντολών Assembly	Αριθμός Εντολών Assembly με -O2 βελτιστοποίηση
Πρίν την διόρθωση	90	54
Μετά την διόρθωση	102	64
Επιπλέον συνάρτηση	293	109

Πίνακας 5.4: Πίνακας με αριθμό εντολών Assembly για το 2ο παράδειγμα του Format String Attacks με διόρθωση της λανθασμένης χρήσης της συνάρτησης syslog.

5.3.3 Λανθασμένη χρήση των Format Functions

Μια ακόμη περίπτωση (και η πιο σημαντική) των Format String Attacks είναι και η λανθασμένη χρήση των Format Functions. Η σύνταξη τους χωρίς την χρήση των των Format String Parameters δεν παράγει λάθος αποτελέσματα αλλά κρύβει κινδύνους για να δεχτεί το πρόγραμμα Format String Attacks.

Αντιμετώπιση της λανθασμένης χρήσης των Format Functions

Στις παραπάνω περιπτώσεις ο μηχανισμός μας λειτουργεί πάλι με την χρήση κάποιων μοτίβων (patterns) και προσπαθεί να εντοπίσει τις λανθασμένες χρήσεις στις Format Functions. Όταν και εάν τις εντοπίσει εφαρμόζει έναν άλλο τρόπο αντιμετώπισης από ότι στα παραπάνω ερωτήματα. Αυτός ο τρόπος είναι η παύση εκτέλεσης του προγράμματος με την χρήση της εντολής abort(). Είναι προφανές πως η διόρθωση καταστάσεων που αφήνουν ένα πρόγραμμα εκτεθειμένο σε κακόβουλες επιθέσεις αποτελεί μια πιο ολοκληρωμένη λύση αλλά εφόσον η διακοπή λειτουργίας του προγράμματος χρησιμοποιείται και αυτή σαν λύση, είναι δεδομένο πως πρέπει να παρουσιαστεί.

Τώρα θα παρουσιαστούν κάποια παραδείγματα λειτουργίας του μηχανισμού που αναπτύχθηκε για τις ανάγκες της παρούσας Διπλωματικής Εργασίας.

Έστω ότι ο παρακάτω κώδικας τίθεται προς έλεγχο.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <regex.h>
#include <syslog.h>
#include <stdarg.h>
void func (char *format, ...){
va_list args;
va_start(args, format);
vprintf(format, args);
va_end(args);
}
int main(int argc , char **argv){
char buf[100] , a[10]="aa" , b[10]="Arman";
char str[10]="Hello";
char str1[30];
int x=1;
char *f="This is %s";
if(x==1){
func(f,b);
snprintf(b,10,"Hi",str);
b[sizeof(b)-1]='\0';
sprintf(b,f,a);
b[sizeof(b)-1]='\0';
snprintf(str1,30,argv[1]);
str1[sizeof(str1)-1]='\0';
}
else{
printf("%s%s%s",argv[1],argv[1],argv[1]);
}
return 0;
}
```

Όπως παρατηρούμε υπάρχουν Format Functions της οικογένειας εντολών printf που δεν έχουν οριστεί σωστά. Συνεπώς το πρόγραμμα αυτό είναι ευάλωτο σε επίθεση Format String Attack. Η εκτέλεση του προγράμματος αυτού με μια είσοδος που περιέχει Format String Parameter είναι η εξής:

Ο μηχανισμός τώρα επεμβαίνει με τον τρόπο που αναφέρθηκε παραπάνω στον κώδικα και τον τροποποιεί για να διασφαλίσει την ασφάλεια των δεδομένων μας.

Ο κώδικας πλέον έχει αυτήν την μορφή.

```
#include"check.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <regex.h>
#include <syslog.h>
#include <stdarg.h>
void func (int num,char *format, ...){
va_list args;
va_start(args, format);
countf(num,format);
vprintf(format, args);
va_end(args);
}
int main(int argc , char **argv){
char buf[100] , a[10]="aa" , b[10]="Arman";
char str[10]="Hello";
char str1[30];
int x=1;
char *f="This is %s";
if(x==1){
func(1,f,b);
snprintf(b,10,"Hi",str);
b[sizeof(b)-1]='\0';
countf(1,f);
sprintf(b,f,a);
b[sizeof(b)-1]='\0';
countf(0,argv[1]);
snprintf(str1,30,argv[1]);
str1[sizeof(str1)-1]='\0';
}
else{
printf("%s%s%s",argv[1],argv[1],argv[1]);
}
return 0;
}
```

Όπως παρατηρούμε πάνω από τις Format Functions προς έλεγχο έχει τοποθετηθεί η συνάρτηση countf(). Επειδή και αυτή η περίπτωση καλύπτει το ενδεχόμενο το όρισμα format να είναι σε μορφή πίνακα χαρακτήρων, έχει δημιουργηθεί η συνάρτηση countf η οποία μετράει εάν τα Format String Parameters στο όρισμα format έχουν τον ίδιο αριθμό με τα ορίσματα της Format Function που ελέγχεται κάθε φορά. Εάν Format String Parameters είναι περισσότερα από τα arguments, τότε το πρόγραμμα διακόπτεται από την εντολή abort() που είναι μέσα στην συνάρτηση countf. Η εκτέλεση του προγράμματος αυτού είναι η εξής:

```
arman@arman-All-Series ~/eclipse-workspace/diplo/format_str $ ./main "Bo%x%xb"
This is ArmanAborted (core dumped)
arman@arman-All-Series ~/eclipse-workspace/diplo/format_str $
```

Σχήμα 5.12: Αποτελέσματα μετά την διόρθωση του προγράμματος.

Όπως βλέπουμε η λειτουργία του προγράμματος σταματάει από την εντολή abort() καθώς το σύστημα έχει παρατηρήσει την επικίνδυνη χρήση της Format Function snprintf και δεν επιτρέπει να συμβεί κάτι παραπέρα από την εντολή αυτή.

Η συνάρτηση countf είναι η εξής:

```
#include "check.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include <string.h>
void countf(int num,...){
  va_list valist;
  char *a, *b;
  int i=0;
  size_t j;
  int count=0;
  va_start(valist,num);
  for (i = 0; i < 1; i++) {
    char *t = va_arg(valist,char *);
        size_t len=strlen(t);
        for(j=0;t[j]!='\0';j++){
        if(t[j]=='%'){
                        if(t[j+1]=='s'){count++;}
                        else if(t[j+1]=='x'){count++;}
                        else if(t[j+1]=='d'){count++;}
                        else if(t[j+1]=='f'){count++;}
                        else if(t[j+1]=='p'){count++;}
                        else if(t[j+1]=='u'){count++;}
                        else if(t[j+1]=='c'){count++;}
                        else if(t[j+1]=='n'){count++;}
                        else if(t[j+1]=='0'){count++;}
                        else if(t[j+1]=='1'){count++;}
                        else if(t[j+1]=='2'){count++;}
                        else if(t[j+1]=='3'){count++;}
                        else if(t[j+1]=='4'){count++;}
                        else if(t[j+1]=='5'){count++;}
                        else if(t[j+1]=='6'){count++;}
                        else if(t[j+1]=='7'){count++;}
                        else if(t[j+1]=='8'){count++;}
                        else if(t[j+1]=='9'){count++;}
                        else{}
                        }
        }
        if(count>num){abort();}
   }
   va_end(valist);
}
```

Ένα δεύτερο παράδειγμα το οποίο αναφέρεται στην λανθασμένη χρήση των Format Functions είναι το παρακάτω.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <regex.h>
#include <syslog.h>

int main(int argc , char **argv){
char buf[100] , a[10] , b[10];
int x=1;
if(x==1){
printf(argv[1]);
}
else{
printf("%s%s%s",argv[1],argv[1],argv[1]);
}
return 0;
}
```

Όπως βλέπουμε σε αυτήν την περίπτωση η πρώτη printf δεν έχει specifiers. Αυτό το γεγονός αποτελεί πρόβλημα γιατί όταν εκτελεστεί αυτή η printf δεν θα βρεθεί specifier για το argument και εάν το argument εμπεριέχει Format String Parameters αυτόματα ο υπολογιστής θα ψάξει στην επόμενη θέση μνήμης της οποίας τα περιεχόμενα θα εμφανίσει στον επιτιθέμενο. Μια εκτέλεση του προγράμματος αυτού θα έχει τα εξής αποτελέσματα.

```
arman@arman-All-Series ~/eclipse-workspace/diplo/format_str $ ./main
arman@arman-All-Series ~/eclipse-workspace/diplo/format_str $ ./main "Bo%x%xb"
Bo5a3ff8585a3ff870barman@arman-All-Series ~/eclipse-workspace/diplo/format_str
```

Σχήμα 5.13: Αποτελέσματα πριν την διόρθωση του προγράμματος.

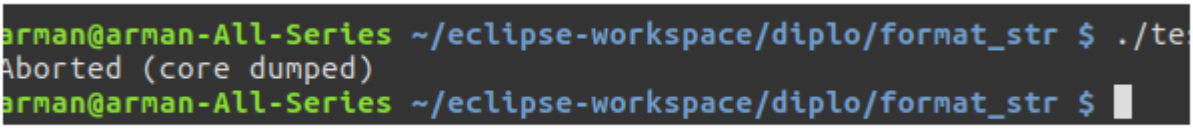
Όπως βλέπουμε ο υπολογιστής εμφάνισε το περιεχόμενο θέσεων μνήμης επειδή δεν ήξερε πώς να διαχειριστεί τα Format String Parameters της εισόδου .

Ο μηχανισμός τώρα επεμβαίνει με τον τρόπο που αναφέρθηκε παραπάνω στον κώδικα και τον τροποποιεί για να διασφαλίσει την ασφάλεια των δεδομένων μας. Ο νέος κώδικας είναι ο παρακάτω

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <regex.h>
#include <syslog.h>

int main(int argc , char **argv){
char buf[100];
char  a[10];
char  b[10];
int x=1;
if(x==1){
abort();
printf(argv[1]);
}
else{
printf("%s%s%s",argv[1],argv[1],argv[1]);
}
return 0;
}
```

Όπως βλέπουμε ο μηχανισμός εντόπισε την προβληματική Format Function και ακριβώς πρίν εκτελεστεί πρόσθεσε την εντολή abort(). Η εκτέλεση του νέου προγράμματος έχει τα εξής αποτελέσματα:



Σχήμα 5.14: Αποτελέσματα μετά την διόρθωση του προγράμματος.

Όπως βλέπουμε ο μηχανισμός μας απέτρεψε την εκτέλεση του προγράμματος εφόσον η Format Function δεν είχε οριστεί σωστά.

Κόστος Αντιμετώπισης της λανθασμένης χρήσης των Format Functions

Για την αντιμετώπιση αυτού του προβλήματος έχουμε χρησιμοποιήσει την μέθοδο της διακοπής εκτέλεσης του προγράμματος σε αντίθεση με τις παραπάνω που έχουν αναφερθεί στην παρούσα Διπλωματική Εργασία στις οποίες ο μηχανισμός έχει παρέμβει διορθωτικά στον κώδικα. Το κόστος αυτής της μεθόδου δεν είναι μεγάλο προφανώς γιατί και η αντιμετώπιση που προτείνεται είναι αρκετά απλή στην λογική και την υλοποίηση της. Η μέθοδος αυτή προϋποθέτει μια παραπανίσια εντολή, την abort() (είτε αμέσως, είτε εμμέσως μέσω της επιπλέον συνάρτησης), η οποία τοποθετείται ακριβώς στην προηγούμενη γραμμή από αυτήν που παρουσιάζει το πρόβλημα. Η παραπανίσια μνήμη που καταναλώνεται από την ασφαλή έκδοση του κώδικα, είναι μηδαμινή καθώς δεν προστίθεται κάποια παραπάνω εντολή ή συνάρτηση πέραν της abort(). Στην περίπτωση όμως που ο μηχανισμός αναγκάζεται να κάνει χρήση της επιπλέον συνάρτησης τότε η μνήμη που απαιτείται αυξάνεται. Η ταχύτητα ,και πάλι, εξαρτάται από το σημείο όπου θα βρεθεί η λανθασμένα ορισμένη Format Function και από το μέγεθος του αρχείου. Είναι άξιο να σημειωθεί ότι ακόμα και μεγάλο να είναι το αρχείο και να αναζητηθεί ολόκληρο για να βρεθεί (εάν βρεθεί) η λανθασμένα ορισμένη Format Function η ταχύτητα της μεθόδου δεν μεταβάλλεται ριζικά.

Πιο αναλυτικά:

Αριθμός Εντολών Assembly ανά περίπτωση		
Κατάσταση	Αριθμός Εντολών Assembly	Αριθμός Εντολών Assembly με -O2 βελτιστοποίηση
Πρίν την διόρθωση	155	95
Μετά την διόρθωση	173	113
Επιπλέον συνάρτηση	293	109

Πίνακας 5.5: Πίνακας με αριθμό εντολών Assembly για το παράδειγμα του Format String Attacks με διόρθωση της λανθασμένης χρήσης των συναρτήσεων Format String .

Όπως γίνεται αντιληπτό η συγκεκριμένη μέθοδος έχει σαν αποτέλεσμα ο ασφαλής κώδικας που παράγεται από τον μηχανισμό να έχει 18 εντολές παραπάνω.Εάν βέβαια η περίπτωση είναι απλή ο ασφαλής κώδικας θα έχει σχεδόν (ελάχιστες παραπάνω) τις ίδιες εντολές με τον αρχικό. Η επιβάρυνση του προγράμματος προέρχεται κυρίως από τις 293 εντολές της περαιτέρω συνάρτησης (109 με την βελτιστοποίηση).

Κεφάλαιο 6

Σύνοψη και Συμπεράσματα

Σε αυτό το κεφάλαιο θα γίνει μια γενική αποτίμηση της παρούσας Διπλωματικής Εργασίας και θα αναφερθεί και η μελλοντική δουλειά που θα μπορούσε να γίνει σχετικά με το εργαλείο αυτό.

6.1 Σύνοψη

Είναι δεδομένο πως η ασφάλεια των δεδομένων καθώς και των συστημάτων που τα διαχειρίζονται θεωρείται απαραίτητη. Ο μηχανισμός που παρουσιάστηκε στην παρούσα διπλωματική εργασία αποτελεί έναν τρόπο αντιμετώπισης δύο επιθέσεων χαμηλού επιπέδου: του Buffer Overflow και του Format String Attack. Πρόκειται για έναν μηχανισμό ό οποίος έχει σαν σκοπό την ανάλυση προγραμμάτων σε C με σκοπό την εύρεση κάποιων τμημάτων τους που θεωρούνται επικίνδυνα και εκθέτουν τα δεδομένα του προγράμματος ή του συστήματος ολόκληρου σε κίνδυνο. Επιπρόσθετα δουλειά του μηχανισμού είναι να διορθώνει τα σημεία εκείνα που θεωρεί επικίνδυνα και να τα κάνει ασφαλή ή εναλλακτικά να σταματάει την εκτέλεση του προγράμματος πριν αυτό φτάσει στο σημείο να θεωρείται επικίνδυνο. Στα πλαίσια της προσπάθειας αυτής αναπτύχθηκαν διαφορετικές μέθοδοι αντιμετώπισης των επιθέσεων αυτών που αξιολογήθηκαν και όσον αφορά την απόδοση τους.

Το σύστημα αυτό έχει δημιουργηθεί στην γλώσσα Java ακριβώς επειδή η

συγκεκριμένη γλώσσα προγραμματισμού δίνει πολλές δυνατότητες τόσο στην δημιουργία patterns για την εύρεση εκείνων των εντολών στην C που μπορεί να προκαλέσουν κάποιο πρόβλημα, όσο και στην εύρεση του καλύτερου τρόπου αντιμετώπισης των επιθέσεων αυτών λόγω την πληθώρας επιλογών και συναρτήσεων που προσφέρει.

Όλα τα παραπάνω μας βοηθούν να εξάγουμε το συμπέρασμα ότι με την εκτέλεση ενός προγράμματος Java αυτόματα ξεκινάει η ανάλυση ενός προγράμματος σε C, η εύρεση των αδύναμων ή εκτεθειμένων τμημάτων κώδικα και η διόρθωση ή η αντιμετώπιση τους. Ο συγκεκριμένος μηχανισμός δεν μπορεί να θεωρείται πλήρως ολοκληρωμένος καθώς υπάρχουν περιπτώσεις των Format String Attacks δεν καλύπτονται. Θεωρείται όμως πως καλύπτει την επίθεση Buffer Overflow με 2 μεθόδους αντιμετώπισης και τα κάποιες σημαντικές εκδοχές των Format String Attacks.

6.2 Μελλοντική Δουλειά

Όπως αναφέρθηκε παραπάνω η γλώσσα στην οποία υλοποιήθηκε ο μηχανισμός αυτός, η Java, προσφέρει πολλές διευκολύνσεις και έχει επίσης και πολύ μεγάλη ποικιλία δυνατοτήτων γεγονός που επιτρέπει την επέκταση του μηχανισμού αυτού.Ο μηχανισμός που υλοποιήθηκε για αυτήν την Διπλωματική Εργασία καλύπτει ήδη την επίθεση Buffer Overflow και ένα μεγάλο φάσμα από τις επιθέσεις Format String Attacks.

Προφανώς ο μηχανισμός αυτός μπορεί να επεκταθεί, και να γίνει προσπάθεια για υλοποίηση παραπάνω μεθόδων αντιμετώπισης των δύο επιθέσεων που πραγματεύεται η παρούσα διπλωματική εργασία. Επίσης θα μπορούσαν να καλυφθούν και κάποιες παραπάνω ευάλωτες εντολές στις επιθέσεις που πραγματεύεται η εργασία αυτή. Τέλος, θα μπορούσε ο μηχανισμός αυτός να καλύπτει και άλλες επιθέσεις όπως, για παράδειγμα, το Stack Overflow που αποτελεί συγκεκριμένο είδος του Buffer Overflow.

Βιβλιογραφία

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *Real-World Buffer Overflow Protection for Userspace and Kernelspace*. Computer Systems Laboratory Stanford University.

[2] G. Edward Suh, Jae W. Lee, David Zhang, Srinivas Devadas. *Secure Program Execution via Dynamic Information Flow Tracking*. Computer Science and Artificial Intelligence Laboratory (CSAIL) Massachusetts Institute of Technology Cambridge, MA 02139.

[3] Lap Chung Lam, Tzi-cker Chiueh. *A General Dynamic Information Flow Tracking Framework for Security Applications*. Rether Networks, Inc. 75 Health Sciences Drive suite 111 Stony Brook, NY 11790, USA.

[4] Udit Dhawan, Catalin Hritcu, Raphael Rubin, Nikos Vasilakis, Silviu Chiricescu Jonathan M. Smith, Thomas F. Knight Jr., Benjamin C. Pierce, André DeHon. *Architectural Support for Software-Defined Metadata Processing*. University of Pennsylvania INRIA, Paris BAE Systems Ginkgo Bioworks.

[5] scut / team teso. *Exploiting Format String Vulnerabilities*. September 1, 2001 version 1.2 .

[6] MMichael Dalton, Hari Kannan, Christos Kozyrakis. *Raksha: A Flexible Information Flow Architecture for Software Security*. Computer Systems Laboratory Stanford University.

[7] Michael Dalton, Hari Kannan, Christos Kozyrakis. *Tainting is Not Pointless*. Computer Systems Laboratory Stanford University.

[8] Joël Porquet and Simha Sethumadhavan. *WHISK: An Uncore Architecture for Dynamic Information Flow Tracking in Heterogeneous Embedded*

SoCs . Department of Computer Science, Columbia University, NY, NY 10027.

[9] Buffer Overflow,
https://en.wikipedia.org/wiki/Buffer_overflow

[10] Format string attack,
https://www.owasp.org/index.php/Format_string_attack

[11] Uncontrolled format string,
https://en.wikipedia.org/wiki/Uncontrolled_format_string

[12] Buffer Overflow Attack Explained with a C Program Example,
http://www.thegeekstuff.com/2013/06/buffer-overflow/?utm_-source=feedly

[13] Format String Vulnerability,
http://www.cs.virginia.edu/ ww6r/CS4630/lectures/Format_-String_Attack.pdf

[14] Format String Bug Exploration,
http://resources.infosecinstitute.com/format-string-bug-exploration

[15] Buffer overflow protection,
https://en.wikipedia.org/wiki/Buffer_overflow_protection

[16] Address space layout randomization (ASLR),
http://searchsecurity.techtarget.com/definition/address-space-layout-randomizat

[17] Category:Input Validation,
https://www.owasp.org/index.php/Category:Input_Validation

[18] Common Types of Cybersecurity Attacks,
https://www.rapid7.com/fundamentals/types-of-attacks/

[19] Fuzzing,
https://en.wikipedia.org/wiki/Fuzzing

[20] Vector (malware)
https://en.wikipedia.org/wiki/Vector_(malware)

[21] Brute-force attack
https://en.wikipedia.org/wiki/Brute-force_attack

[22] Stack buffer overflow
https://en.wikipedia.org/wiki/Stack_buffer_overflow

[23] Computer security
https://en.wikipedia.org/wiki/Computer_security

[24] Vulnerability (computing)
https://en.wikipedia.org/wiki/Vulnerability_(computing)

[25] Buffer Overflow
https://www.owasp.org/index.php/Buffer_Overflow

[26] Buffer Overflow Attack
http://www.cse.scu.edu/ tschwarz/coen152_-
05/Lectures/BufferOverflow.html

[27] WHAT IS A BUFFER OVERFLOW? LEARN ABOUT BUFFER O-
VERRUN VULNERABILITIES, EXPLOITS AND ATTACKS
https://www.veracode.com/security/buffer-overflow

[28] Bounds checking
https://en.wikipedia.org/wiki/Bounds_checking

[29] Bounds checking
https://en.wikipedia.org/wiki/Bounds_checking

[30] Cross-site scripting
https://en.wikipedia.org/wiki/Cross-site_scripting

[31] Cross-site Scripting (XSS)
https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[32] Phishing
https://en.wikipedia.org/wiki/Phishing

[33] phishing
http://searchsecurity.techtarget.com/definition/phishing

[34] CAPEC-67: String Format Overflow in syslog()
https://capec.mitre.org/data/definitions/67.html

[35] Format String Bugs
https://msdn.microsoft.com/en-us/library/ee823826(v=cs.20).aspx

[36] Memory Corruption Attacks The (almost) Complete History
https://media.blackhat.com/bh-us-10/whitepapers/Meer/BlackHat-USA-2010-Meer-His

[37] Low-Level Software Security: Attacks and Defenses
https://link.springer.com/chapter/10.1007/978-3-540-74810-6_-
4

[38] HIGH LEVEL ATTACKS WHEN YOU ARE ON THE WEB
http://www.security-faqs.com/high-level-attacks-when-you-are-on-the-web.html

[39] What is Persistent XSS (Cross-Site-Scripting)
https://www.acunetix.com/blog/articles/persistent-xss/

[40] SQL Injection Prevention Cheat Sheet
https://www.owasp.org/index.php/SQL_Injection_Prevention_-
Cheat_Sheet