ΑΞΙΟΛΟΓΗΣΗ ΑΣΦΑΛΕΙΑΣ ΓΙΑ ΕΞΥΠΝΑ ΣΥΜΒΟΛΑΙΑ ETHEREUM

Από
Χρήστο Δαβιλά.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΦΕΒΡΟΥΑΡΙΟΣ 2021

Διπλωματική Εργασία υποβλήθηκε στο Τμήμα Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς για την απόκτηση Μεταπτυχιακού Διπλώματος Ειδίκευσης στην Ασφάλεια Ψηφιακών Συστημάτων

Christos Xenakis, Ph.D., Chair

Ευχαριστίες

Με την περάτωση της παρούσας μεταπτυχιακής εργασιάς θα ήθελα να ευχαριστήσω τον κ. Χρήστο Ξενάκη για την αγαστή συνεργασία που είχαμε και για την καθοδήγησή του.

Επιπλέον, θα ήθελα να ευχαριστήσω την οικογένεια μου και τους φίλους μου για τη στήριξη που μου παρείχαν.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΠΙΝΑΚΩΝ

ΠΕΡΙΛΗΨΗ

Με την παρούσα μεταπτυχιακή εργασία θα μελετήσουμε τις βασικές έννοιες του blockchain, των smart contracts καθώς και την ασφάλεια των τελευταίων. Η τεχνολογία blockchain, τα smart contracts καθώς και τα κρυπτονομίσματα είναι από τα πεδία της βιομηχανίας που παρουσίαζουν μεγάλη δυναμική και σχεδόν εκθετική αύξηση χρόνο με το χρόνο. Τα smart contracts καταλαμβάνουν πολλούς τομείς της βιομηχανίας προκειμένου να αυτοματοποιηθούν οι διαδικάσιες και να εξαλειφθεί ο εξωτερικός διαμεσολαβητής. Συνεπώς, η ασφάλεια τους λαμβάνει όλο και μεγαλύτερη αξία και προσοχή. Θα αναλύσουμε γνωστές ευπάθειες των smart contracts και τρόπους αποφυγής τους με έμφαση στα εργαλεία στατικής ανάλυσης.

1. Εισαγωγη

1.1. Σκοπός της μεταπτυχιακής εργασίας

Με την παρούσα μεταπτυχιακή εργασία θα μελετήσουμε τις βασικές έννοιες του blockchain, των smart contracts καθώς και την ασφάλεια των τελευταίων. Η blockchain τεχνολογία λαμβάνει όλο και μεγαλύτερη απήχηση στο ευρύτερο κοινό λόγω των Bitcoins. Τα Bitcoins που έχουν συγκεντρώσει και το μεγαλύτερο ενδιαφέρον λόγω των επενδυτικών κεφαλαίων που συσωρεύονται όπως έγινε πρόσφατα με την μεγαλή αγορά Bitcoins από τη Tesla. [1]

Εν τούτοις, θα ήταν λάθος να εστιάζαμε μόνο σε μία εφαρμογή αυτής της τεχνολογίας. Πέρα από τις ψηφιακές συνναλλαγές, μέσω των έξυπνων συμβολαίων ανοίγει η πόρτα για πληθώρα αγορών όπως στην εφοδιαστική αλυσίδα, για τις δημόσιες υπηρεσίες, για την υγεία, για τα πνευματικά δικαιώματα, για την ενέργεια.

Συνεπώς, η ανάλυση της ασφάλειας για τις παραπάνω εφαρμογές είναι εξέχουσας σημασίας. Ήδη, πλήθος επιθέσεων έχει σημειωθεί σε έξυπνα συμβόλαι που είχαν σαν αποτέλεσμα την απώλεια εκαταντάδων εκατομμυρίων δολαρίων. [2] [3] Συνεπώς, κάθε έξυπνο συμβόλαιο που αναρτάται στο blockchain θα πρέπει να αναλύεται διεξοδικά και να ελέγχεται για τυχόν γνωστές ευπάθειες που δύναται να έχει. Στην παρούσα διπλωματική αφού μελετήσουμε τις γνωστές ευπάθεις θα παρουσιάσουμε τα πιο γνωστά εργαλεία και τα αποτελέσματα αυτών.

1.2. Δομή εργασίας

Κεφάλαιο 2: Αναλύουμε βασικές έννοιες της κρυπτογραφίας, του blockchain και της τεχνολογίας Ethereum. Μέσα από αυτό το κεφάλαιο ο χρήστης θα μπορέσει να κατανοήσει το υπόβαθρο των smart contracts.

Κεφάλαιο 3: Αναλύουμε γνωστές επιθέσεις που έχουν γίνει σε Ethereum smart contracts καθώς και πως μπορούμε να τις αποτρέψουμε. Επίσης, υπάρχει μια ταξινόμηση των επιθέσεων.

Κεφάλαιο 4: Συζητάμε για την αναγκαιότητα χρήσης εργαλείων στατικής ανάλυσης κατά την ανάπτυξη του συμβολαίου. Επιπλέον, μέσω των εργαλείων καταδεικνύουμε γνωστές ευπάθειες και εξάγουμε συμβόλαια από το Ethereum blockchain για να τα

ελέξουμε. Τέλος, συζητάμε για καλές πρακτικές που πρέπει να εφαρμόζουν οι προγραμματιστές κατά τη σύνταξη των συμβολαίων.

Κεφάλαιο 5: Συζητάμε για τα αποτελέσματα της μεταπτυχιακής εργασίας καθώς και για τα συμπεράσματα που εξήγαμε.

2. ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ

2.1. Βασικές αρχές κρυπτογραφίας

Οι τρεις βασικές αρχές είναι η εμπιστευτικότητα των δεδομένων, η ακεραιότητα των δεδομένων, η διαθεσιμότητα των δεδομένων. Η εμπιστευτικότητα των δεδομένων αναφέρεται στην προστασία των πληροφοριών από την πρόσβαση από μη εξουσιοδοτημένα μέρη. Η ακεραιότητα των δεδομένων αναφέρεται στη διασφάλιση της αυθεντικότητας των δεδομένων και ότι τα δεδομένα δε τροποποιούνται καθώς και η πηγή των πληροφοριών είναι γνήσια. Τέλος, η διαθεσιμότητα των δεδομένων όπου διασφαλίζει ότι τα δεδομένα είναι προσβάσιμα από εξουσιοδοτημένους χρήστες. [4]

2.2. Υβριδική κρυπτογραφία

Υπάρχουν διάφοροι αλγόριθμοι κρυπτογραφίας που επιτυγχάνουν μία ή περισσότερες από τις βασικές αρχές. Χωρίζονται σε δύο κατηγορίες. Η πρώτη κατηγορία είναι οι συμμετρικοί αλγόριθμοι όπως το AES-256 και η δεύτερη κατηγορία είναι οι ασύμμετροι αλγόριθμοι όπως ο Diffie-Hellman. Κυρίως, χρησιμοποιούνται σε συνδυασμό για να ικανοποιούν ταυτόχρονα τις βασικές αρχές. Στο υβριδικό κρυπτοσύστημα χρησιμοποιείται ο ασύμμετρος αλγόριθμος για την κρυπτογράφηση του κλειδιού και ο συμμετρικός αλγόριθμος για την κρυπτογράφηση των δεδομένων.

2.3. Συνάρτηση κατατεμαχισμού (Hash Function)

Η συνάρτηση κατακερματισμού είναι μια συνάρτηση που είναι πρακτικά ανέφικτο να αντιστραφεί και έχει τρεις κύριες ιδιότητες. Το αποτέλεσμά της ονομάζεται σύνοψη. Πρώτον, δεν είναι εφικτό να βρείς την είσοδο από τη σύνοψη. Δεύτερον, δεν είναι εφικτό να τροποποιηθεί η είσοδος χωρίς να τροποποιηθεί η σύνοψη. Τέλος, είναι ανέφικτο να βρεθούν δύο τιμές που θα έχουν την ίδια έξοδο. [5]

2.4. Δείκτης Κατακερματισμού (Hash pointer)

Ένας δείκτης κατακερματισμού (hash pointer) είναι ένας δείκτης που περιέχει εκτός από μια διεύθυνση το κρυπτογραφικό κατακερματισμό των δεδομένων που υποδεικνύουν. Μια συνδεδεμένη λίστα δεικτών κατακερματισμού μπορεί να ονομαστεί blockchain, έτσι διασφαλίζεται η ακεραιότητα της αλυσίδας. Οι λίστες έχουν πολλά μειονεκτήματα, για παράδειγμα την πολυπλοκότητα της διαγραφής εισαγωγής O (N). Αντ 'αυτού, χρησιμοποιούνται ευρέως Merkle trees. Merkle tree, είναι ένα δυαδικό δέντρο με δείκτες κατακερματισμού. Τα φύλλα είναι μπλοκ δεδομένων και οι κόμβοι είναι οι κατακερματισμοί των αντίστοιχων παιδιών τους. Έτσι, παρέχει έναν αποτελεσματικό τρόπο απόδειξης ότι ένα συγκεκριμένο μπλοκ δεδομένων περιέχεται (Proof of Membership) και ότι ένα συγκεκριμένο μπλοκ δεδομένων δεν περιέχεται (Proof of Non Membership). [6]

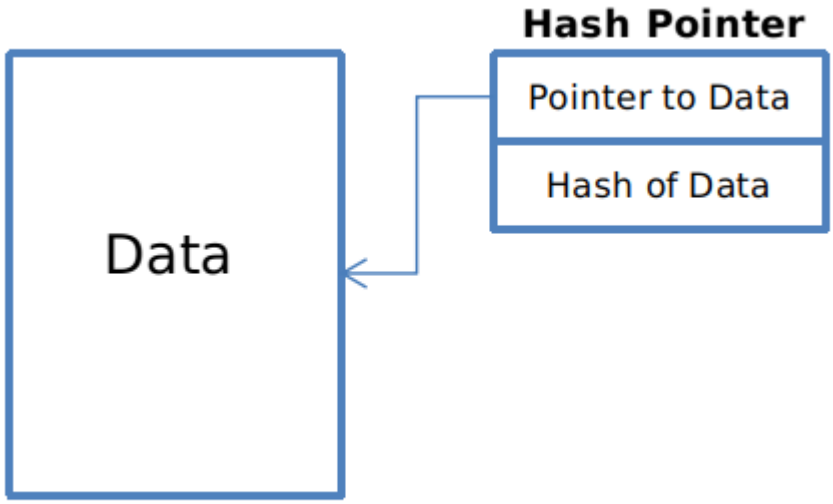

Figure 1:Hash pointer [6]

2.5. Bitcoin

Το Bitcoin είναι ένα κρυπτονόμισμα που εφευρέθηκε από τον Satoshi Nakamoto το 2008. [7] Ο κύριος στόχος ήταν μια καθαρά έκδοση peer-to-peer ηλεκτρονικών μετρητών χωρίς να περάσει από ένα χρηματοπιστωτικό ίδρυμα. Το κύριο πρόβλημα που έλυσε το bitcoin είναι ο έλεγχος των διπλών δαπανών χωρίς κεντρική αρχή. Αυτό επιτυγχάνεται καθώς όλες οι συναλλαγές δημοσιεύονται δημοσίως και είναι αδύνατο να χειραγωγηθούν.

Ο Satoshi Nakamoto, ορίζει ότι ένα ηλεκτρονικό νόμισμα είναι μια αλυσίδα ψηφιακής υπογραφής. Προκειμένου να αποφευχθεί η χρήση μιας κεντρικής αρχής, χρησιμοποιεί

ένα δίκτυο ψηφιακών χρονοσφραγίδων. Κατά συνέπεια, ένα σύστημα Proof-of-Work χρησιμοποιείται για έναν κατανεμημένο διακομιστή χρόνο σφραγίδων. Συγκεκριμένα, αυξάνεται ένα nonce στο μπλοκ έως ότου βρεθεί μια τιμή που δίνει στο hash του μπλοκ με τα απαιτούμενα μηδενικά bit. Επί της ουσίας το Proof-of-Work είναι ένας επεξεργαστής μία ψήφους. [8]

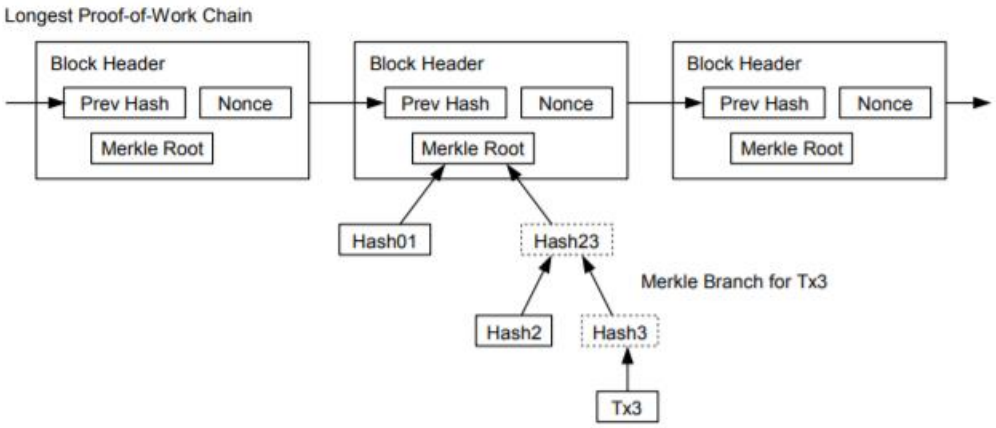

Figure 2:Bitcoin transaction. [7]

Το βασικό κίνητρο των κόμβων για συμμετοχή στο δίκτυο είναι ότι η πρώτη συναλλαγή σε ένα μπλοκ που ξεκινά ένα νέο νόμισμα ανήκει στον δημιουργό του μπλοκ. Επίσης, οι κόμβοι ανταμείβονται για τις υπηρεσίες τους με τέλη συναλλαγής. Το τέλος συναλλαγής εξαρτάται από το μέγεθος της συναλλαγής. Ένας κόμβος έχει πολλούς ρόλους μέσα στο δίκτυο, όπως wallet owner, light node, full node, miner. Ένας πλήρης κόμβος (full node) ενσωματώνει όλες αυτές τις υπηρεσίες. Οι ελαφροί κόμβοι (ligt node) χρησιμοποιούν τη μέθοδο απλοποιημένης επαλήθευσης πληρωμής για την επαλήθευση συναλλαγών. Οι κόμβοι εξόρυξης (miner) ανταγωνίζονται για τη δημιουργία νέων μπλοκ που επιλύουν τον αλγόριθμο proof-of-work. Τέλος, ο κάτοχος του πορτοφολιού (wallet owner) υπογράφει και δημοσιεύει τις συναλλαγές.

Η διαδικασία που ακολουθείται για να εισαχθεί μία συναλλαγή σε ένα block είναι η ακόλουθη:

- Η συναλλαγή δημιουργείται και υπογράφεται από το κάτοχο του πορτοφολιού.
- Η συναλλαγή στέλνεται σε ένα πλήρη κόμβο, ο οποίος την επικυρώνει.
- Η συναλλαγή διαδίδεται στο διαδίκτυο και αποθηκεύεται στο memory pool.
- Ο κόμβος εξόρυξης χτίζει τη συναλλαγή μέσα σε ένα μπλοκ και το μεταφέρει στο διαδίκτυο.
- Οι υπόλοιποι κόμβοι επικυρώνουν το μπλοκ, και ανανεώνουν το memory pool.

Το bitcoin έχει σα βάση του, τις συναλλαγές οι οποίες διαχωρίζονται σε εισερχόμενες και εξερχόμενες. [9] Οι συναλλαγές δε σχετίζονται με λογαριασμούς και θεωρούνται

σα κλάσματα από bitcoin. Οι συναλλαγές διαθέτουν και τα αντίστοιχα scripts, sciptSig για τις εισερχόμενς συναλλαγές και scriptPubKey για τις εξερχόμενες συναλλαγές. Όπου η δομή της συναλλαγής χωρίζεται στα εισερχόμενα δεδομένα που περιέχουν την προηγούμενη συναλλαγή και τα εξερχόμενα όπου περιέχονται το scriptPubkey και το μεταβιβασθέν ποσό.

```
Input:
Previous tx: f5d8ee39a430901c91a5917b9f2dc19d6d1a0e9cea205b009ca73dd04470b9a6
Index: 0
scriptSig: 304502206e21798a42fae0e854281abd38bacd1aeed3ee3738d9e1446618c4571d10
90db022100e2ac980643b0b82c0e88ffdfec6b64e3e6ba35e7ba5fdd7d5d6cc8d25c6b241501

Output:
Value: 5000000000
scriptPubKey: OP_DUP OP_HASH160 404371705fa9bd789a2fcd52d2c580b65d35549d
OP_EQUALVERIFY OP_CHECKSIG
```

Figure 3:Transaction structure. [10]

Ένα από τα θεμελιώδη στοιχεία του blockchain είναι του πως επιτυγχάνει την ομοφωνία.

- Κάθες κόμβος που λαμβάνει ή δημιουργεί μία συναλλαγή, το μεταδίδει στο υπόλοιπο δίκτυο.
- Κάθε κόμβος, συλλέγει τις έγκυρες συναλλαγές, και δημιουργεί ένα νέο block που τις περιέχει.
- Το δίκτυο επιλέγει τυχαία ένα κόμβο και προτείνει το block προς επικύρωση από το δίκτυο.
- Οι υπόλοιποι κόμβοι, λαμβάνουν τον κόμβο και τον ελεγχουν εάν είναι αυθεντικός.
- Οι υπόλοιποι κόμβοι εφόσον τον αποδεχτούν, χτίζουν τα υπόλοιπα blocks πάνω από το «τυχαίο» block.

Συνεπώς, είναι αδύνατο κάποιος κακόβουλος κόμβος να μπλοκάρει τις συναλλαγές κάποιου χρήστη καθώς κάθε φορά θα επιλέγεται ένας διαφορετικός κόμβος. Επίσης, θα αναφερθούμε στα updates που χρειάζεται το δίκτυο των bitcoin και πως αυτό επιτυγχάνεται. Κατ'αρχάς, το update μπορεί να στοχευεί διαφορετικές λειτουργικότητες (Consensus Layer/Peer Service Layer/API Layer, Application Layer), όλες οι προτάσεις αναφέρονται σαν Bitcoin improvement proposals (BIP). [11]

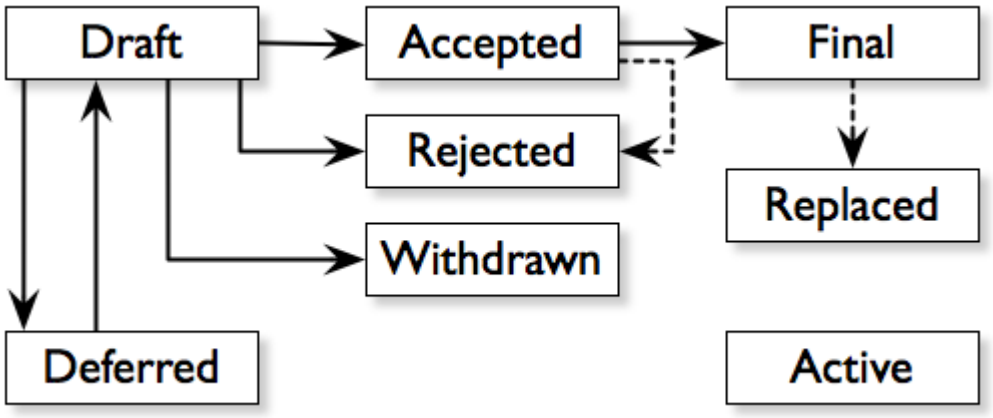

Figure 4:Bitcoin improvement proposals flow. [11]

Το bitcoin [7] είναι ένα PoW μηχανισμός ομοφωνίας(1 CPU – 1 Vote), συνεπώς το χειρότερο είδος επίθεσης, είναι κάποιος κακόβουλος να έχεις το 51% της επεξεργαστικής ισχύος. Θα κατάφερνε να μπλοκάρει άλλους χρήστες, να συλλέγει όλα τα mining rewards και επί της ουσίας θα ξαναέγραφε την ιστορία της αλυσίδας.

Τέλος, ένα από τα μεγαλύτερα μειονεκτήματα του bitcoin είναι το περιβαλλοντικό αποτύπωμα που αφήνει, είνα σχεδόν εφάμιλο με τις ενεργειακές απαιτήσεις της Ελβετίας. [12] Σε συνδιασμό με το μικρό throughput συναλλαγών που μπορεί να επιτύχει το καθιστά εξαιρετικά δύσχρηστο.

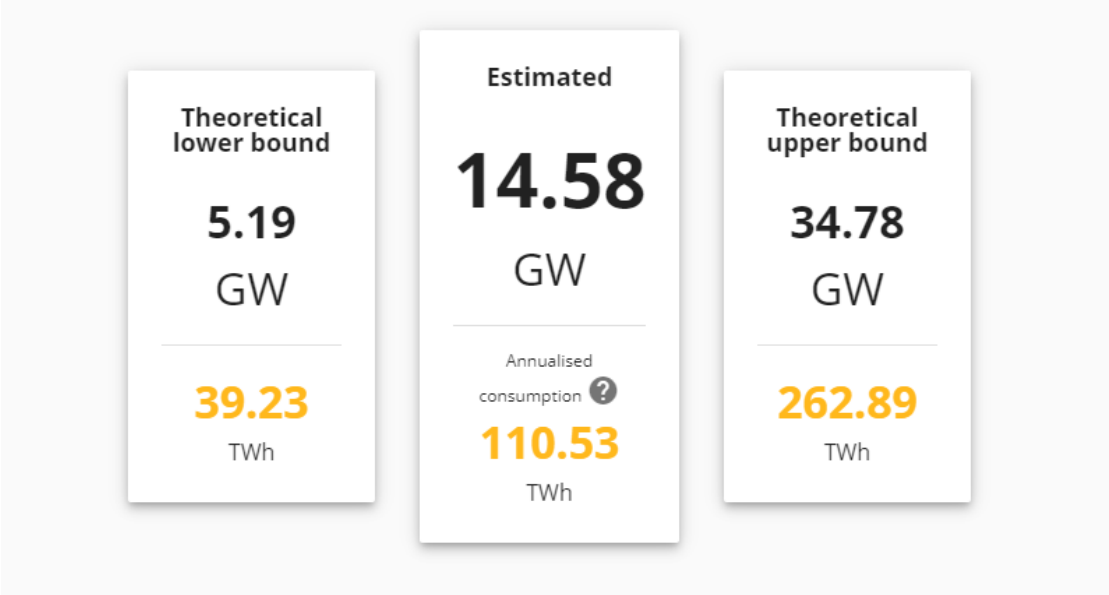

Figure 5:Bitcoin energy consumption. [13]

2.6. Ethereum

Το Ethereum είναι μία αποκεντρωμένη blockchain πλατφόρμα, η οποία προτάθηκε από τον Vitalik Buterin το 2013. [14] Η μεγάλη διαφορά μεταξύ του Bitcoin και του Ethereum, έγκειται στο γενονός ότι το Ethereum δεν περιορίζεται μόνο σε συναλλαγές αλλά μέσω των έξυπων συμβολαίων έχει μία πληθώρα χρήσεων όπως η υλοποίηση συμβολαίων για τις μεταφορές, τη ναυτιλία, την εφοδιαστική αλυσίδα κ.ο.κ. Η υλοποίηση των συμβολαίων επιτυγχάνεται μέσω μία Turing-complete γλώσσας. Όπως αναφέρει και ο ιδρυτής της συγκεκριμένης πλατφόρμας η βασική του φιλοσοφία έγκειται σε 5 άξονες: [14]

- Στην απλότητα του προκειμένου κάθε προγραμματιστής να μπορεί να το χειριστεί και να το υλοποιήσει.
- Στην καθολικότητα του καθώς βασίζεται σε μία Turing-complete γλώσσα και ο καθένας να μπορεί να υλοποιεί οποιαδήποτε συναλλαγή ή συμβόλαιο επιθυμεί.
- Στην ευελιξία του καθώς υπάρχει πρόνοια για τυχόν updates ή αλλαγές στον πρωτόκολλο που θα βελτιώνουν την επεκτασιμότητα και την ασφάλεια της πλατφόρμας.
- Στην έλλειψη λογοκρισίας καθώς το πρωτόκολλο δε θα επιτρέπει τον περιορισμό οποιασδήποτε χρήσης.

Μία απο τις ουσιώδεις διαφορές μεταξύ Bitcoin και Ethereum είναι ότι το δεύτερο, μπορεί να κρατάει την κατάσταση μίας συναλλαγής, πιο συγκερκριμένα ονομάζεται Ethereum Account όπου περιέχει τέσσερα πεδία. [14]

- Nonce, ένας μετρητής για να επιβεβαίωσει ότι κάθε συναλλαγή εκτελείται μία φορά.
- Το τρέχον υπόλοιπο του λογαριασμού.
- Ο κώδικας του συμβολαίου του λογαριασμού.
- Η αποθήκευση του λογαριασμού.

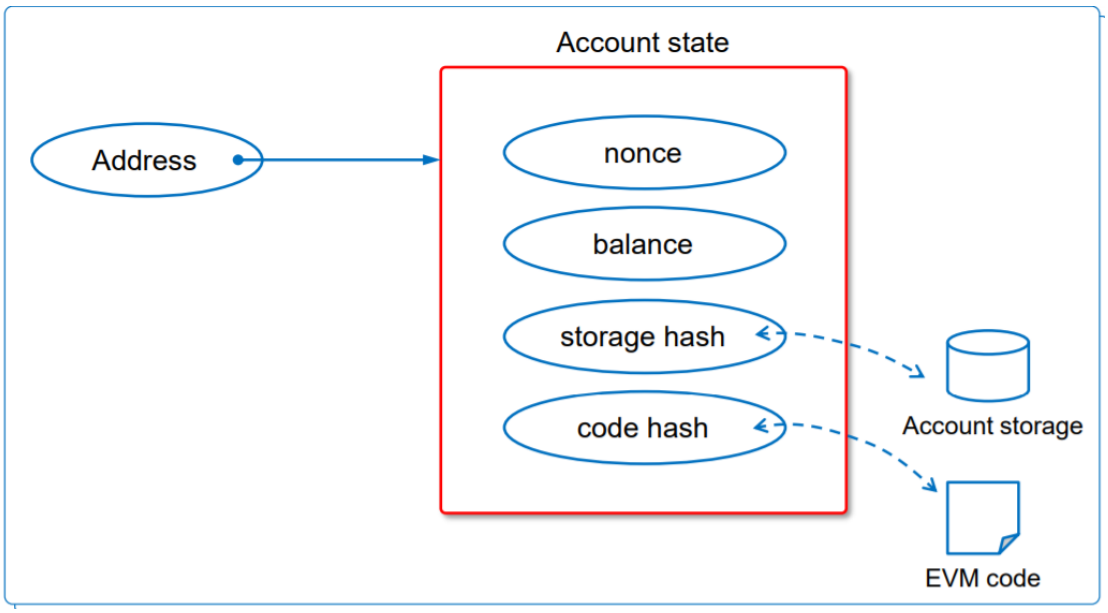

Figure 6:Ethereum account. [15]

Το Ether είναι επί της ουσίας το νόμισμα με το οποίο πληρώνονται τα κόμιστρα συναλλαγών. Στο Ethereum υπάρχουν δύο είδη Accounts, τα externally owned accounts και τα contract accounts. Τα πρώτα ελέγχονται με ένα ιδιωτικό κλειδί και δεν περιέχουν κώδικα. Τα δεύτερα κάθε φορά που λαμβάνουν ένα μήνυμα ο κώδικας τους ενεργοποιείται επηρεάζοντας την εσωτερική κατάσταση του contract.

Με τον όρο συναλλαγή ορίζουμε τα υπογεγραμμένα πακέτα που περιέχουν ένα μήνυμα που στέλνεται από ένα external owned account. Μία συναλλαγή περιλαμβάνει: [14]

- Τον παραλήπτη του μηνύματος.
- Μία υπογραφή που πιστοποιεί τον αποστολέα.
- Το ποσό του ether που μεταφέρεται από τον αποστολέα στον παραλήπτη.
- STARTGAS, ο μέγιστος αριθμών υπολογιστικών βημάτων.
- GASPRICE, το κόμιστρο που πληρώνει ο αποστολέας για τα υπολογιστικά βήματα.

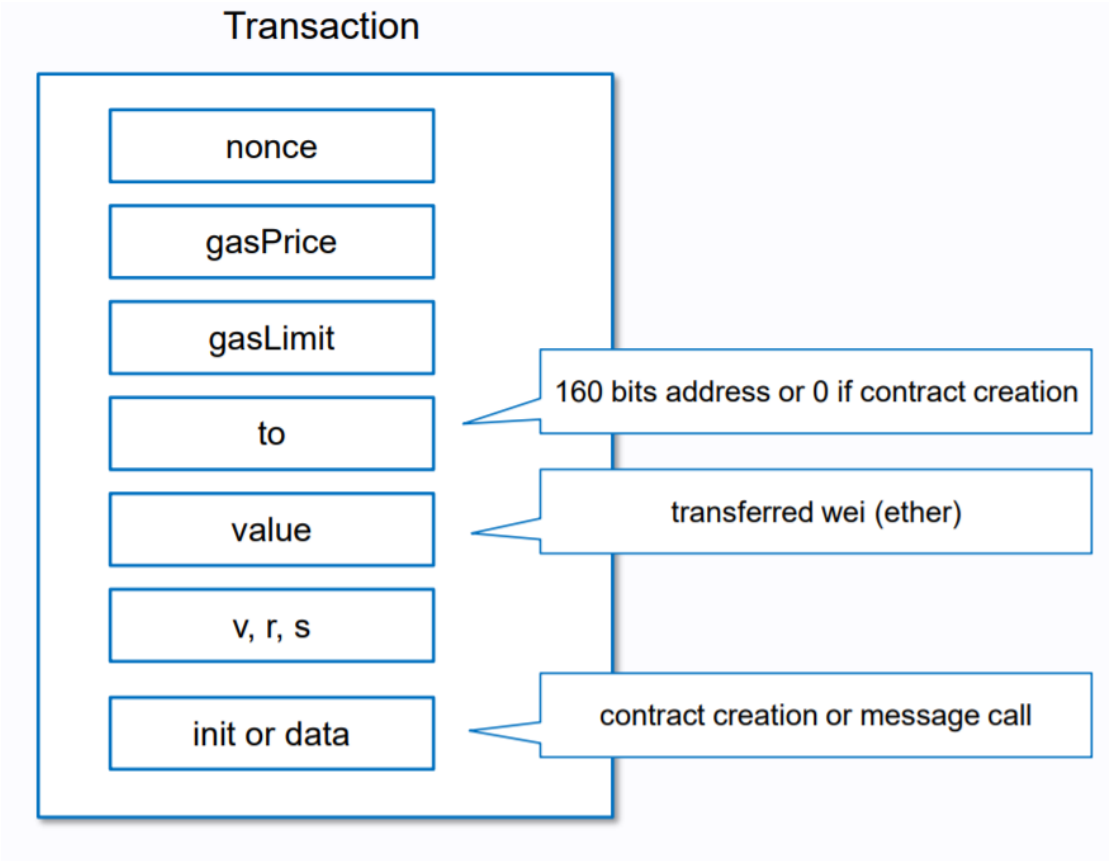

Figure 7:Ethereum transaction. [15]

Τα contracts έχουν τη δυνατότητα να στέλνουν μήνυματα σε άλλα contracts και περιέχουν : [14]

- Τον αποστολέα του μηνύματος.
- Τον αποδέκτη του μηνύματος.
- Το ποσό του ether που μεταφέρεται στο μήνυμα.
- STARTGAS.

Οι συναρτήσεις μετάβασης κατάστασης (Ethereum transition functions), ελέγχουν τα κάτωθι:

- Καταρχάς, ελέγχει εάν η συναλλαγή είναι συντακτικά σωστή, η υπογραφή είναι έγκυρη και το Nonce ταιριάζει με αυτό του αποστολέα.
- Υπολογίζει, το κόμιστρο που ισούται με το STARGAS * GASPRICE.
- Μεταφέρει το ποσό συναλλαγής από το λογαριασμό του αποστολέα, στο λογαριασμό του αποδέκτη.
- Εάν για οποιοδήποτε λόγο αποτύχει η συναλλαγή, τότε επανέρχεται στην αρχική κατάσταση η συναλλαγή αλλά το κόμιστρο πληρώνεται κανονικά στο miner.

- Εάν επιτύχει η συναλλαγή, το GAS που περίσσεψε επιστρέφεται στον αποστολέα και το υπόλοιπο δίνεται σα κόμιστρο στον miner.

Ethereum Virtual Machine, είναι μία οντότητα που αποτελείται από πλήθος συνδεμένων υπολογιστικών συστημάτων που τρέχουν σαν Ethereum Clients. Επί της ουσίας είναι το περιβάλλον όπου τα Ethereum contracts υπάρχουν και σε κάθε block στην αλυσίδα, υπάρχει ένα μοναδικό ένα καθολικό και μοναδικό state.

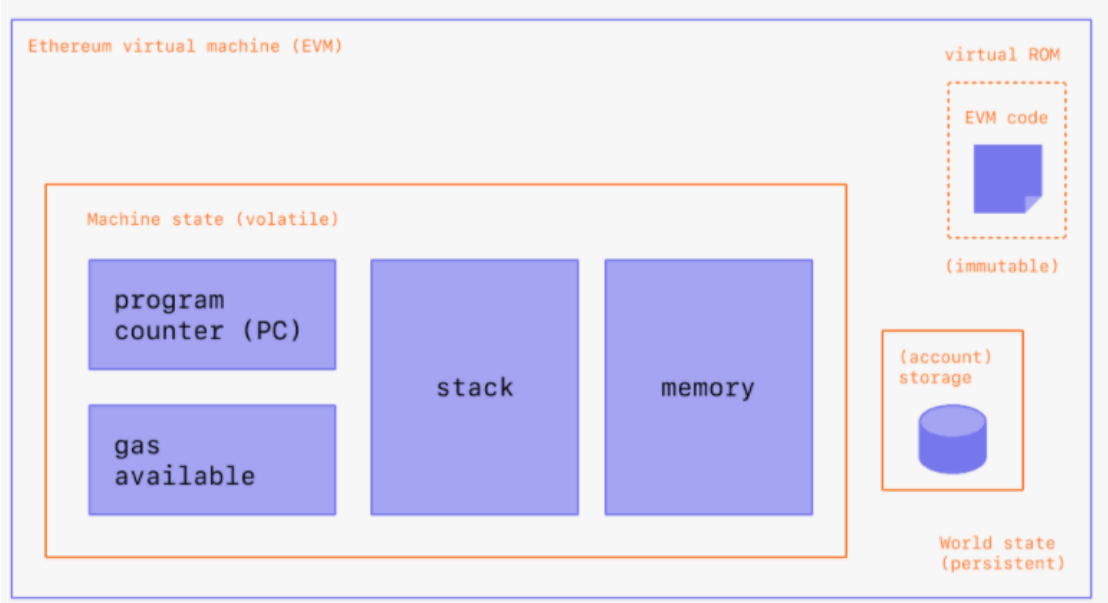

Figure 8:Ethereum EVM illustrated. [16]

Ο κώδικας που εκτελείται στα contracts, ονομάζεται σαν EVM code. Ο κώδικας αποτελείται από μία ακολουθία από bytes, όπου κάθε byte αναπαριστά ένα operation. Τα operation έχουν πρόσβαση σε τρεις τύπους στο EVM, όπως φαίνεται και στο Figure 8:Ethereum EVM illustrated., στο stack, στη memory και στο storage. Υπάρχουν, πληθώρα EVM υλοποιήσεων γραμμένες σε διάφορες γλώσσες, όπως σε python [17], C++ [18], GO [19].

2.7. Ethereum Smart contract

Ένα smart contract, είναι ένα πρόγραμμα που σκοπό έχει να εκτελείται αυτόματα εφόσον οι όροι του συμβολαίου έχουν ικανοποιηθεί. Ο σκοπός του είναι να εξαλείψει την ανάγκη για μεσολαβητές και βασίζεται στο μηχανισμό ομοφωνίας που προσφέρει το blockchain. Ο όρος πρώτο εισήχθει το 1994 από τον Nick Szabo, ο οποίος εφηύρε το Bit Gold το 1998 [20]. Συγκρεκριμένα τα Ethereum smart contracts είναι μία συλλογή από συναρτήσεις και δεδομένα τα οποία βρίσκονται σε μια συγκεκριμένη διεύθυνση στο Ethereum blockchain.

Τα Ethereum smart contracts, έχουν ένα balance και μπορούν να στείλουν συναλλαγές πάνω στο δίκτυο. Εδώ, πρέπει να δώσουμε έμφαση στην ασφάλεια των smart contracts καθώς, εφόσον αναπτυχθούν στο δίκτυο είναι προσβάσιμα από τον καθένα και δεν μπορούν να τροποποιηθούν. Ο εκάστοτε χρήστης μπορεί να αλληλεπιδράσει με το smart contract μέσω των συναλλαγών προκειμένου να εκτελεστεί μία συγκεκριμένη συνάρτηση του συμβολαίου. [16]

Ο καθένας μπορεί να γράψει ένα smart contract και να το ανεβάσει στο δίκτυο. Αυτό επτυγχάνεται μέσω της Solidity η της Vyper σαν γλώσσας προγραμματισμού, όπου αργότερα θα δούμε τα κενά ασφαλείας που έγκειται σε αυτές τις γλώσσες. Επιπλέον, υπάρχουν και οι smart contract libraries προκειμένου να διευκολύνουν τη δημιουργία των smart contracts. Χρησιμοποιώντας blocks από τα libraries μπορείς να χρησιμοποιήσεις συμπεριφορές που έχουν ξανα υλοποιηθεί είτε κάποια standards που έχουν υλοποιηθεί. Το ανέβασμα στο δίκτυο είναι τεχνικά μία συναλλαγή, όπου πρέπει να πληρώσεις GAS. Επίσης, τα smart contracts δεν έχουν πρόσβαση σε εξωτερικά δεδομένα και ο μόνος τρόπος για πρόσβαση σε εξωτερικά δεδομένα είναι μέσω των oracles.

Τα oracles, συνδέουν το smart contract, με τον εξωτερικό κόσμο, όπως προαναφέραμε. Συνεπώς, τα oracles είναι υπεύθυνα να φέρνουν την εξωτερική πληροφορία και να την αναρτούν στο blockchain. Έτσι, κάθε κόμβος θα μπορεί να αναπαράξει τη συναλλαγή και να έχει ακριβώς το ίδιο αποτέλεσμα. Τα oracles είναι από τα κύρια συστατικά που θα αναλύσουμε για την ασφάλεια τους.

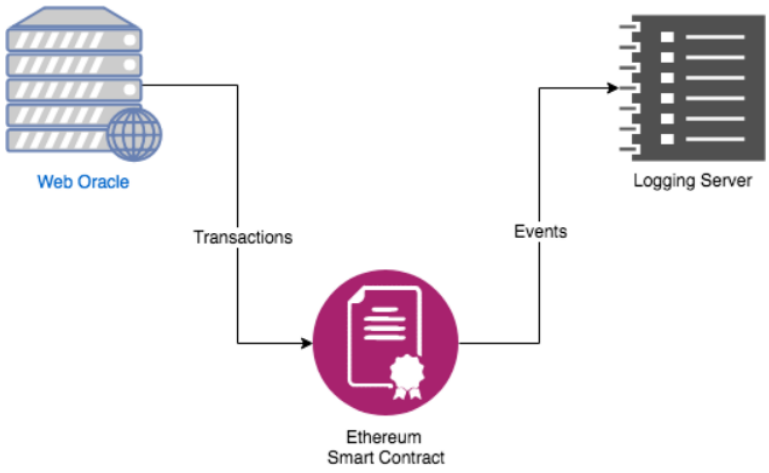

Figure 9:Oracles basic flow. [21]

Τα δεδομένα του εκάστοτε contract, αποθηκεύονται είτε στο storage είτε στη memory. Τα δεδομένα που αποθηκεύονται στο storage, αποθηκεύονται εν γένει στο blockchain μόνιμα. Εν αντιθέσει, τα δεδομένα που αποθηκεύονται στη memory, έχουν διάρκεια ζωής όσο και η διάρκεια ζωής της συνάρτησης που τα χρησιμοποιεί.

2.8. Gas limit

Όπως αναφέραμε, το Gas είναι από τα κύρια συστατικά στοιχεία του Ethereum. Επί της ουσίας το Gas αναφέρεται στην υπολογιστική προσπάθεια που απαιτείται προκειμένου να εκτελεστεί μία συναλλαγή στο δίκτυο. Συνεπώς το Gas είναι το κόμιστρο που πληρώνεται για να εκτελεστεί αυτή η συναλλαγή.

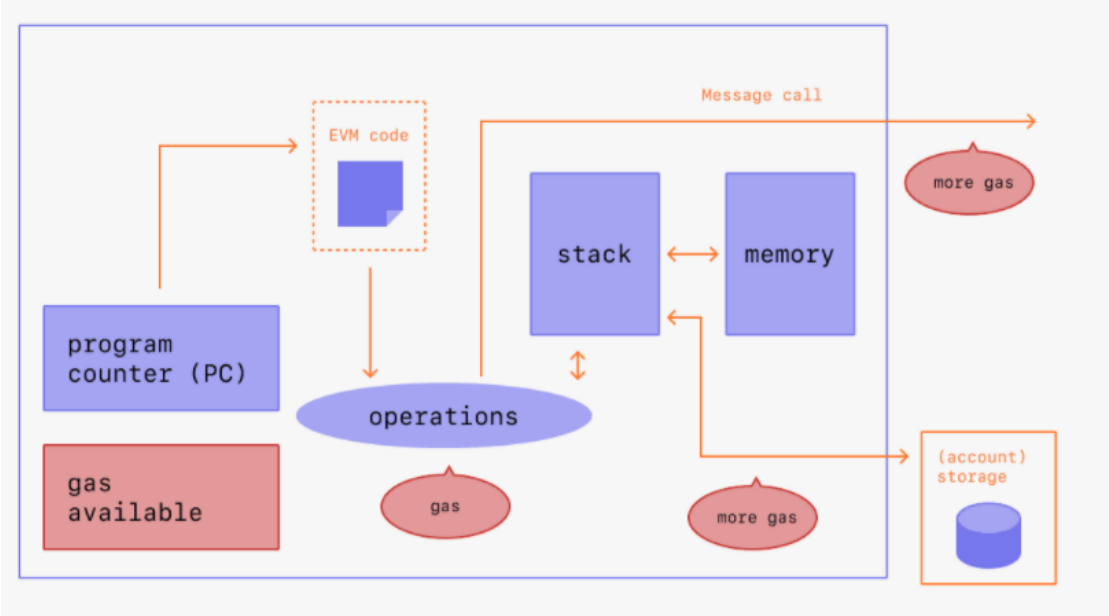

Figure 10:Ethereum Gas [16]

Ένας από τους κύριους λόγους, που υπάρχει το Gas είναι για την ασφάλεια όπως αναφέρεται και από τον ιδρυτή του [14], καθώς οι κακόβουλοι χρήστες θα πρέπει να πληρώσουν μεγάλο κόμιστρο προκειμένου να κάνουν μία επίθεση. Επίσης, σε περίπτωση που ολοκληρωθεί μια συναλλαγή και παραμείνει Gas, επιστρέφεται στον αποστολέα.

3. Γνωστες επιθεσεις σε Ethereum smart contracts και vulnerabilities

3.1. Εισαγωγή

Τα ethereum smart contracts όπως και κάθε προιόν που βασίζεται σε κώδικα, μπορεί να έχεις bugs, τα οποία στην περίπτωση μας οδήγησαν και θα οδηγούν στην απώλεια χρήματων. Υπάρχει πληθώρα περιπτωσέων κατά το παρελθόν όπου εκλάπησαν μεγάλα χρηματικά ποσά, με πιο γνωστή την επίθεση που έγινε κατά της DAO και είχε σα συνέπεια την απώλεια 50 εκατομμυρίων δολλαρίων. [2] Οι επιθέσεις κατα των smart contracts μπορούν να διαχωριστούν σε 3 κατηγορίες [22] :

- Σε ευπάθειες που οφείλονται στη γλώσσα προγραμματισμού που χρησιμοποιείται για τη σύνταξη συμβολαίων, τη Solidity.
- Σε ευπάθειες που οφείλονται στην πλατφόρμα που χρησιμοποιείται, δηλαδή στο Ethereum Virtual Machine.
- Σε ευπάθειες που παρατηρούνται στο Blockchain.
- Σε εγγενείς ευπάθειες των Oracles.

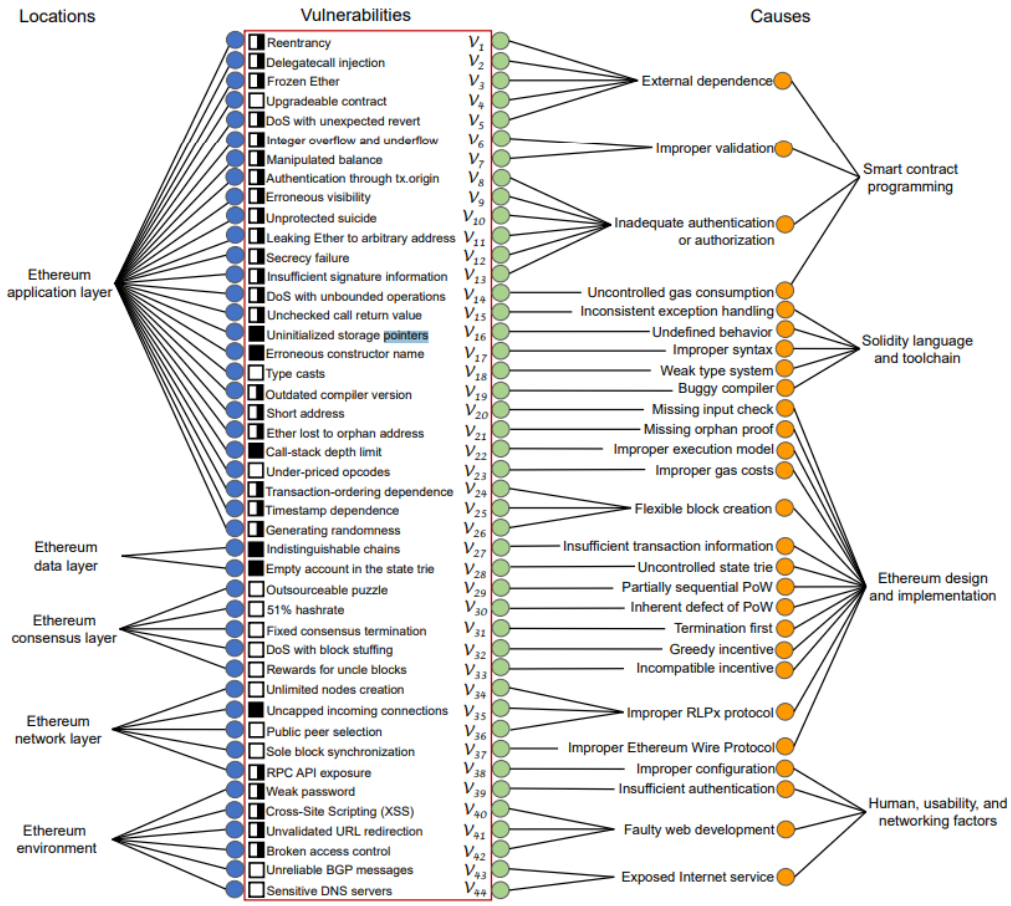

Figure 11:Ethereum vulnerabilities [23]

3.2. Ανατομία της Solidity

Η solidity είναι μία αντικειμενοστραφής γλώσσα προγραμματισμού συνεπώς διαρθρώνεται και αντίστοιχα: [24]

- Όταν ένα contract δημιουργείται καλείται ο constructor που ορίζεται με το keyword constructor και είναι προαιρετικός. Αφού, εκτελεστεί ο constructor το συμβόλαιο εισέρχεται στο blockchain.
- Το visibility [24] [25] :
    - External - Μπορούν να κληθούν από άλλα contracts είτε μέσω transactions.

    - Public – Μπορούν να κληθούν μέσω των messages.
    - Internal – Είναι functions που μπορούν να κληθούν μόνο από το ίδο contract.
    - Private - Είναι functions που μπορούν να κληθούν μόνο από το ίδο contract και όχι από τα παιδιά του contract.
- Ο compiler δημιουργεί αυτόματα getters functions για όλες τις public μεταβλητές.
- Function modifiers όπου επιτρέπουν να επηρεάσουν μία συνάρτηση πριν κληθεί.
- Constant / Immutable καταστάσεις των μεταβλητών που δεν επιττρέπεται να αλλάξουν. Η διαφορά έγκειται στο γεγονός ότι οι immutable μεταβλήτες δύναται να αλλαχθούν από τον constructor, εν αντιθέσει με τις constant μεταβλητες (compile time).
- Pure functions, ονομάζονται οι συναρτήσεις που δεν διαβάζουν / τροποποιούν την κατάσταση του contract.
- Receive ether functions, κάθε contract έχει το πολύ μία, δε μπορούν να έχουν arguments ούτε να επιστρέφουν και έχουν external visibility.
- Fallback functions, εκτελείται εφόσον καμία αλλή συνάρτηση δε ταιριάζει είτε δεν έχει δηλωθεί καμία ether function. Για να λάβει ether θα πρέπει να χρησιμοποιηθεί το keyword payable.
- Events, είναι inherited από το EVM logging functionality κατά αντιστοιχία χρησιμοποιούνται από τα contracts για το transaction’s log.
- Κληρονομικότητα, για να υποστηριχθεί θα πρέπει να χρησιμοποιούνται τα keywords virtual και override. Συνεπώς, ένα συμβόλαιο μπορεί να κληρονομεί από ένα άλλο.

3.3. Ευπάθειες στη Solidity

Καταρχάς, τα smart contracts γίνονται δημόσια από τη στιγμή που εισέρχονται στο blockchain. Συνεπώς, ο καθένας μπορεί να δει τον κώδικα του contract και να προσπαθήσει να το εξαπατήσει και να εξάγει χρήματα προς όφελος του. [24]

3.3.1. Re-entrancy

Τα contracts μπορούν να καλέσουν άλλα contracts, αυτή η κλήση προς τα εξωτερικά contracts, μπορεί να προκαλέσει την ευπάθεια. Συνεπώς, δια μέσου της fallback συνάρτησης μπορεί ο επιτιθέμενος να καλέσει κώδικα του αρχικού συμβολαίου.(re-entrancy) Αυτό το είδος της επίθεσης χρησιμοποιήθηκε για την περίφημη επίθεση στο DAO [2] καθώς και για την επίθεση SpankChain [26]. Όπως φαίνεται και στο κάτωθι παράδειγμα, ο επιτιθέμενος θα μπορεί να πάρει τον έλεγχο του contract δια μέσου του call back καθώς η γραμμη 4i δεν θα εκτελεστεί μέχρι να τελειώσει όλο το gas του αποστελέα.

```
1.  // SPDX-License-Identifier: GPL-3.0
2.  pragma solidity >=0.6.0 <0.9.0;

3.  // THIS CONTRACT CONTAINS A BUG - DO NOT USE
4.  contract Fund {
        a.  /// @dev Mapping of ether shares of the contract.
        b.  mapping(address => uint) shares;
        c.  /// Withdraw your share.
        d.  function withdraw() public {
        e.  if (payable(msg.sender).send(shares[msg.sender])) [24]
                i.  shares[msg.sender] = 0;
        f.  }
5.  }
```

Για να αντιμετωπίσουμε αυτή την ευπάθεια, υπάρχουν τρεις τεχνικές που χρησιμοποιούνται:

- Πρώτον, θα μπορούσε να χρησιμοποιηθεί η συνάρτηση transfer() κατά το αρχικό call που έχει πεπερασμένο αριθμό gas, συνεπώς δε θα επιτυγχανόταν κλήσεις περεταίρω συναρτήσεων.
- Δεύτερον, θα μπορούσε να αλλάξουν οι τοπικές μεταβλητές πριν την εκτέλεση της εξωτερικής κλήσης.
- Τρίτον, θα μπορούσε να χρησιμοποιηθεί mutex, που δε θα επέτρεπε το re-entrancy call.

3.3.2. Arithmetic underflows / overflows

Η πλατφόρμα Ethereum Vitural Machine ορίζει συγκεκριμένους τύπους δεδομένων. Συνεπώς, εάν ο προγραμματιστής δε προσέξει με το contract μπορεί να υποπέσει σε arithmetics uderflows / overflows. Όπως βλέπουμε και στο κάτωθι παράδειγμα είναι δυνατόν μέσω της πρόσθεσης ή της αφαίρεσεις να έχουμε την προαναφερθείσα ευπάθεια.

```
1. pragma solidity ^0.4.16;
2. contract TimeLock {
3.
4.      mapping(address => uint) public balances;
5.      mapping(address => uint) public lockTime;
6.
7.      function deposit() public payable {
8.          balances[msg.sender] += msg.value;
9.          lockTime[msg.sender] = now + 1 weeks;
10.       }
11.
12.       function increaseLockTime(uint _secondsToIncrease) public
    {
13.           lockTime[msg.sender] += _secondsToIncrease;
14.       }
```

```
15.
16.      function withdraw() public {
17.          require(balances[msg.sender] > 0);
18.          require(now > lockTime[msg.sender]);
19.          uint transferValue = balances[msg.sender];
20.          balances[msg.sender] = 0;
21.          msg.sender.transfer(transferValue);
22.      }
23.  } [27]
```

Για να αντιμετωπίσουμε τη συγκερκιμένη ευπάθεια ακολουθούμε τις εξής τεχνικές :

- Η solidity διαθέτει ένα mode, όπου αναγνωρίζει τα overflows / underflows.
- Επίσης, μπορούν να χρησιμοποιηθούν μαθηματικές βιβλιοθήκες που έχουν επιλύσει αυτά τα ζητήματα όπως τη OpenZepelin [28].

3.3.3. Default visibilities

Οι συναρτήσεις στη Solidity έχουν διάφορα επίπεδα visibility, το οποίο μας δείχνει ποιος επιτρέπεται να τις καλέσει και κατηγοριοποιούνται ως εξής [24] [25] :

- External - Μπορούν να κληθούν από άλλα contracts είτε μέσω transactions.
- Public – Μπορούν να κληθούν μέσω των messages.
- Internal – Είναι functions που μπορούν να κληθούν μόνο από το ίδο contract.
- Private - Είναι functions που μπορούν να κληθούν μόνο από το ίδο contract και όχι από τα παιδιά του contract.

Η ευπάθεια έγκειται στο γεγονός ότι το default visibility είναι Public, συνεπώς, οι προγραμματιστές μπορεί να ξεχάσουν να αλλάξουν το visibility. Για να αποτρέψουμε αυτή την ευπάθεια, θα πρέπει πάντα να ορίζουμε το visibility ακόμα και εάν είναι Public. [27]

3.3.4. Entropy illusion

Όπως έχουμε αναλύσει το Ethereum είναι μια πλατφόρμα που τα transactions έχουν μια ντετερμινιστική κατάσταση και αντίστοιχα είναι η καθολική κατάσταση του Ethereum. [27] Συνεπώς, δεν υπάρχει πηγή τυχαιότητας η εντροπία είναι αρκετά χαμηλή. Τα block hashes είναι τυχαία, αλλά ο κακόβουλος χρήστης μπορεί να τροποποιήσει τα blocks για να επηρεάσει τα hashes.

Αυτή η ευπάθεια μπορεί να οδηγήσει σε κακόβουλη εκμετάλευση των συμβολαίων που χρησιμοποιούν τη τυχαιότητα, όπως τα gambling contracts σε μία lottery. Για να επιλυθεί αυτή η ευπάθεια θα πρέπει η πηγή της τυχαιότητας να είναι εξωτερική, μέσω oracles.

3.3.5. *tx origin*

Η solidity έχει μία καθολική μεταβλητή που επιστρέφει τη διεύθυνση του λογαριασμού που έστειλε αρχικά τη κλήση. Χρησιμοποιώντας, αυτή τη μεταβλητή για αυθεντικοποίηση το contract, είναι ευπαθές για phising – like επιθέσεις.

Για να αποτρέψουμε αυτή την επίθεση δε, θα πρέπει να χρησιμοποιούμε το tx.origin για αυθεντικοποίηση του αποστολέα. Αλλά, θα μπορούσε να βάλει έναν έξτρα έλεγχο:

Require(tx.origin == msg.sender)

```
1. // SPDX-License-Identifier: GPL-3.0
2. pragma solidity >=0.7.0 <0.9.0;
3. contract TxUserWallet {
4.     address owner;
5.
6.     constructor() {
7.         owner = msg.sender;
8.     }
9.
10.      function transferTo(address payable dest, uint amount)
   public {
11.          require(tx.origin == owner);
12.          dest.transfer(amount); [22]
```

3.3.6. *Race condition*

Στη συγκεκριμένη επίθεση ο επιτιθέμενος εκμεταλεύεται το χρονικό περιθώριο μεταξύ της δημιουργίας της συναναλλαγής και της στιγμής που γίνεται αποδεκτό στο blockchain. Επί της ουσίας, ο miner που επιλύει το γρίφο, επιλέγει ποια συναλλαγή θα εισαχθεί στο block, συνήθως επιλέγεται αυτή με το μεγαλύτερο gas price. Συνεπώς ο επιτιθέμενος μπορεί να παρακολουθεί το transaction pool, να φτιάξει ένα καινούριο συμβόλαιο που θα έχει μεγαλύτερο gasPrice και έτσι θα εισαχθεί πρώτα το συμβόλαιο του επιτιθέμενου. [25] [24]

Για να αποτρέψουμε αυτή την επίθεση, θα μπορούσαμε να θέσουμε ένα άνω όριο στο gasPrice, συνεπώς οι κακόβουλοι χρήστες δε θα μπορούσαν να το εκμεταλευτούν. Η καλύτερη τεχνική για να αποτρέψουμε τέτοιο είδους επιθέσεων θα ήταν η απόκρυψη της λύσης μέχρι να εισαχθεί το συμβόλαιο στο blockchain. Εφόσον το συμβόλαιο εισαχθεί στο blockchain ο χρήστης στέλνει μία συναλλαγή που αποκαλύπτει τη λύση.

3.3.7. *Wrong constructor name*

Μία συνάρτηση που προοριζόταν να είναι ο constructor του συμβολαίου, από λάθος είχε άλλο όνομα. Συνεπώς οποιοσδήποτε κακόβουλος θα μπορούσε να καλεί το όνομα της συνάρτησης που προοριζόταν να είναι ο constructor και να αλλάζει την κατάσταση του συμβολαίου. Σαν λύση για τη συγκεκριμένη ευπάθεια, πρέπει να χρησιμοποείται το keyword Constructor αντί για το όνομα της κλάσης. [25]

3.3.8. Denial of Service *(DOS)*

Η γνωστή επίθεση DOS, στα συμβόλαια έχει την έννοια ότι ο επιτιθέμενος, μπορεί να αφήσει τα συμβόλαια σε μία κατάσταση όπου το συμβόλαιο δε θα μπορεί να λειτουργήσει. Υπάρχουν μία πληθώρα από τεχνικές που μπορεί να επιτευχθεί το παραπάνω σενάριο: [29] [27]

- Όταν μία συναλλαγή ανακαλείται καθώς ένα σφάλμα συμβαίνει κατά μία εξωτερική κλήση. Όταν χρησιμοποιείται το CALL opcode, το οποίο δεν ανακαλεί τη συναλλαγή όταν αποτυγχάνει. Για να αποτρέψουμε αυτού του είδους τις επιθέσεις θα πρέπει να ορίσουμε ένα όριο στο gas που θα χρησιμοποιήσει η συνάρτηση.
- Όταν σε ένα συμβόλαιο, υπάρχει μία loop σε ένα array όπου το ορίζει ο επιτιθέμενος, συνεπώς το gas που απαιτείται ξεπερνάει το gas limit. Αυτή η ευπάθεια μπορεί να αποτραπεί εάν τα contracts δε χρησιμοποιούν loops για δομές δεδομένων που δίδονται από το χρήστη.
- Κάποια συμβόλαια, των οποίων η κατάσταση εξαρτάται από κάποια εξώτερική κλήση μπορεί να δεχθούν DOS- attack καθώς μπορεί να αποτραπεί αυτή η πληροφορία.

3.3.9. Delegate Call to Untrusted Callee

Υπάρχει μία ξεχωριστή κατηγορία για την κλήση του μηνύματος η οποία ονομάζεται Delegate Call και έχει το opcode DELEGATECALL. Η διαφορά έγκειται στο γεγονός ότι ο κώδικας εκτελείται στο πλαίσιο του καλούντος συμβολαίου και msg.sender και msg.value δεν αλλάζουν τις τιμές τους. Η προαναφερθείσα λειτουργηκότητα μας δίνει τη δυνατότητα της υλοποίησης βιβλιοθηκών όπου ο προγραμματιστής μπορέι να δημιουργήσει κώδικα για μελλοντικά συμβόλαια. Στο παρακάτω συμβόλαιο μπορούμε να δούμε ένα χαρακτηριστικό παράδειγμα όπου ένας κακόβουλος μπορεί να πάρει τον έλεγχο ενός συμβολαίου. Το ευπαθές συμβόλαιο έχει ένα συνδιασμό απο 2 ευπάθειες, η πρώτη αφορά το visibility των συναρτήσεων και το δεύτερο το delegate call. Συνεπώς, στο msg.data υπάρχει το signature της συνάρτησης που επιθυμεί να καλέσει.

```
1. pragma solidity ^0.4.11;
```

```
2.
3. // Credits to OpenZeppelin for this contract taken from the
   Ethernaut CTF
4. //
   https://ethernaut.zeppelin.solutions/level/0x68756ad5e1039e4f3b
   895cfaa16a3a79a5a73c59
5. contract Delegate {
6.
7.   address public owner;
8.
9.   function Delegate(address _owner) {
10.      owner = _owner;
11.    }
12.
13.    function pwn() {
14.      owner = msg.sender;
15.    }
16.  }
17.
18.  contract Delegation {
19.
20.    address public owner;
21.    Delegate delegate;
22.
23.    function Delegation(address _delegateAddress) {
24.      delegate = Delegate(_delegateAddress);
25.      owner = msg.sender;
26.    }
27.
28.    function() {
29.      if(delegate.delegatecall(msg.data)) {
30.        this;
31.      }
32.    }
33.  }
```

Η ευπάθεια έγκειται στο γεγονός ότι ο κώδικας στη διεύθυνση στόχου μπορεί να αλλάξει οποιαδήποτε μεταβλητή του καλούντος και να αποκτήσει πρόσβαστη στο Wallet του καλούντος. [30] Για να αποφύγουμε τη συγκεκριμένη ευπάθεια θα πρέπει όταν γίνεται χρήση του Delegate Call να χρησιμοποιείται μια whitelist από συμβόλαια. Επίσης, η Solidity μέσω του keyword library μας δίνει τη δυνατότητα για την υλοποίηση stateless library συμβολαίων συνεπώς αποφεύγουμε την παραπάνω επίθεση. Η πιο γνωστή επίθεση που συνέβη είναι The Party Wallet Hack [31] όπου εκλάπησαν γύρω στα 30 εκ. $.

3.3.10. Use of Deprecated Solidity Functions

Αρκετές συναρτήσεις είτε operators στη Solidity μπορεί να προκαλέσουν ευπάθειες στα συμβόλαια καθώς είναι ξεπερασμένες συνεπώς η χρήση τους καλό είναι να αποφεύγεται. Επίσης, με νεότερες εκδόσες του compiler η συμπεριφόρα τους δε θα είναι ντετερμινιστική και μπορεί να έχουμε βυζαντινές βλάβες. [32]

DEPRECATED	ALTERNATIVE
SUICIDE	Selfdestruct
BLOCK.BLOCKHASH	Blockhash
SHA3	Keccak256
CALLCODE	Delegatecall
THROW	Revert
MSG.GAS	Gasleft
CONSTANST	View
VAR	Type name

Table 1:List of deprecated functions/operations.

```
1. pragma solidity ^0.4.24;
2. contract DeprecatedSimple {
3.     // Do everything that's deprecated, then commit suicide.
4.     function useDeprecated() public constant {
5.         bytes32 blockhash = block.blockhash(0);
6.         bytes32 hashofhash = sha3(blockhash);
7.         uint gas = msg.gas;
8.
9.         if (gas == 0) {
10.             throw;
11.          }
12.          address(this).callcode();
13.
14.          var a = [1,2,3];
15.
16.          var (x, y, z) = (false, "test", 0);
17.
18.          suicide(address(0));
19.      }
20.      function () public {}
21.
22.  } [32]
```

3.3.11. Assert Violation

Η συνάρτηση της Solidity assert έχει σαν σκοπό να ελέγχει τη τιμή invariants μεταβλητών. Συνεπώς δε θα πρέπει να χρησιμοποιείται για την εγκυρότητα των δεδομένων διαφορετικά μπορεί να εισάγει bugs. Για να αποφύγουμε τη συγκεκριμένη πιθανότητα θα πρέπει να χρησιμοποιείται η require για τον εγκυρότητα δεδομένων που ενδέχεται να μεταβληθούν. [33]

```
1. /*
2.  * @source: ChainSecurity
3.  * @author: Anton Permenev
4.  */
5. pragma solidity ^0.4.21;
6.
7. contract GasModel{
8.     uint x = 100;
```

```
9.      function check(){
10.           uint a = gasleft();
11.           x = x + 1;
12.           uint b = gasleft();
13.           assert(b > a);
14.       }
15.   }
16.
```

3.3.12. Unprotected SELFDESTRUCT Instruction

Η συνάρτηση selfdestruct(address), αφαιρεί το bytecode και στέλνει όλο το ether που υπήρχε στο συμβόλαιο στην διεύθυνση που είναι δοσμένη σαν όρισμα. Κάποιος κακόβολους θα μπορεί να καταστρέψει ένα συμβόλαιο σε περίπτωση που η self-destruct δεν έχει χρησιμοποιηθεί σωστά. Συνεπώς, ο προγραμματιστής θα πρέπει να είναι ιδιαίτερα επιφυλακτικός με τη χρήση της συγκεκριμένης συνάρτησης και σε περίπτωση που είναι απολύτως αναγκαία η χρήση της να χρησιμοποιείται ένα mulitisig σχήμα όπου θα πρέπει πολλαπλά μέρη να εγκρίνουν την συγκεκριμένη εντολή. [34]

```
1. pragma solidity ^0.4.22;
2.
3. contract SimpleSuicide {
4.
5.   function sudicideAnyone() {
6.     selfdestruct(msg.sender);
7.   }
8.
9. }
```

3.3.13. Write to arbitrary Storage Location / Unitialized Storage pointer

Τα ευαίσθητα δεδομένα του συμβολαίου (π.χ ο ιδιοκτήτης του συμβολαίου) αποθηκεύονται σε κάποιο συγκεκριμένο location σε επίπεδο EVM. Το συμβόλαιο είναι υπεύθυνο για να διασφαλίζει ότι μόνο οι εξουσιοδοτημένοι χρήστες μπορούν να γράψουν στο συγκεκριμένο location. Η ευπάθεια έγκειται στο γεγονός ότι κάποιος κακόβουλος θα μπορεί να γράψει στο συγκεκριμένο location και να αλλοιώσει τα ευαίσθητα δεδομένα. Για να αντιμετωπιστεί η συγκεκριμένη ευπάθεια θα πρέπει ο προγραμματιστής να διασφαλίζει ότι τα δεδομένα που θα γράφονται σε μία δομή δεδομένων δε θα επηρέαζουν άλλες δομές. [35] Η συγκεκριμένη ευπάθεια γίνεται αντιληπτή από το compiler (v.0.5.0) και προκαλεί compilation error. [24] [27]

```
1. // A Locked Name Registrar
2. contract NameRegistrar {
3.
```

```
4.      bool public unlocked = false;  // registrar locked, no name
   updates
5.
6.      struct NameRecord { // map hashes to addresses
7.          bytes32 name;
8.          address mappedAddress;
9.      }
10.
11.       mapping(address => NameRecord) public
   registeredNameRecord; // records who registered names
12.       mapping(bytes32 => address) public resolve; // resolves
   hashes to addresses
13.
14.       function register(bytes32 _name, address _mappedAddress)
   public {
15.           // set up the new NameRecord
16.           NameRecord newRecord;
17.           newRecord.name = _name;
18.           newRecord.mappedAddress = _mappedAddress;
19.
20.           resolve[_name] = _mappedAddress;
21.           registeredNameRecord[msg.sender] = newRecord;
22.
23.           require(unlocked); // only allow registrations if
   contract is unlocked
24.       }
25.   }  [35]
```

3.3.14. Requirement Violation

Όπως αναλύσαμε και στο Section 3.3.11 η συνάρτηση require(), χρησιμοποιείται για να ελέγχει την εγκυρότητα των δεδομένων τα οποία συνήθως δίνοντας από τους καλούντες της συνάρτησης. Επί της ουσίας, χρησιμοποιείται για να ελέξει ότι ικανοποιούνται τα invariants του contract. Σε περίπτωση που η require δεν ικανοποιείται υπάρχουν δύο ενδεχόμενα:

- Υπάρχει bug στο συμβόλαιο που έδωσε το όρισμα.
- Η συνθήκη που ελέγχει το require είναι πολύ αυστηρή.

Για να αποφύγουμε τη δεύτερη περίπτωση θα πρέπει να κάνουμε πιο χαλαρή τη συνθήκη, ενώ για την πρώτη περίπτωση το bug πρέπει να διορθωθεί στο αρχικό συμβόλαιο. [36]

```
1. pragma solidity ^0.4.25;
2.
3. contract Bar {
4.      Foo private f = new Foo();
5.      function doubleBaz() public view returns (int256) {
6.          return 2 * f.baz(0);
7.      }
8. }
9.
10.  contract Foo {
11.       function baz(int256 x) public pure returns (int256) {
12.           require(0 < x);
13.           return 42;
```

```
14.         }
15.   }
```

Όπως βλέπουμε και στο παραπάνω κομμάτι του κώδικα, η συνθήκη του δε θα ικανοποιηθεί ποτέ στη γραμμή 12.

3.4. Ευπάθειες στο EVM

3.4.1. Stack Size limit

Το call stack ξεπερνιέται συνεπώς ένα exception προκαλείται, όταν ένα συμβόλαιο καλεί ένα άλλο συμβόλαιο η ακόμα και τον εαυτό του. Το όριο του stack είναι 1024 και το όριο των κλήσεων ξεπερνιέται ένα exception προκαλείται. Ο κακόβουλος μπορεί να προκαλέσει τη κλήση του δικού του συμβολαίου μέχρι να φτάσει στο όριο και έπειτα να καλέσει το ευπαθές συμβόλαιο προκειμένου να προκληθεί το exception.

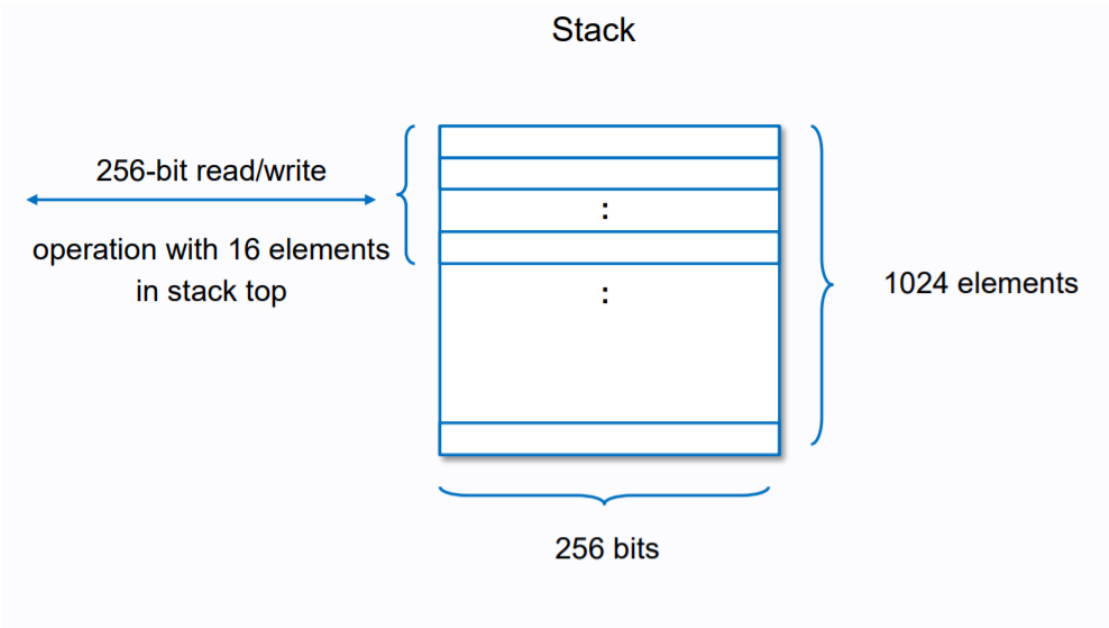

Figure 12:EVM stack illustration. [16]

3.4.2. Ether lost in transfer

Προκειμένου να στείλουμε ένα ether πρέπει να ορίσουμε και τη διεύθυνση του παραλήπτη. Σε περίπτωση που η διεύθυνση είναι ορφανή δε συνδέεται με κάποιο χρήστη / συμβόλαιο τότε το ether χάνεται για πάντα.

3.5. Ευπάθειες στο Blockchain

3.5.1. Unpredictable state

Η κατάσταση ενός συμβολαίου αλλάζει δυναμικά αναλόγως και τα πεδία του συνεπώς δεν υπάρχει εγγύηση ότι όταν κάποιος στέλνει μια συναλλαγή σε ένα συμβόλαιο ότι θα έχει την ίδια κατάσταση.

3.5.2. Consensus mechanism vulnerability

Έγκειται στην εν γένει αδυναμία των μηχανισμών ομοφωνίας, άρα άμα κάποιος συγκεντρώσει το 51% της υπολογιστικής ισχύς τότε αποκτάει και τη δύναμη να ξαναγράψει τα blocks.

3.6. Ευπάθειες στο Oracle

Τα oracles είναι άρρηκτα συνδεδεμένα με το blockchain, καθώς είναι η διεπαφή μεταξύ του blockchain και του πραγματικού κόσμου. Επί της ουσίας το oracle είναι ένας third-party οργανισμός για τον οποίο είναι αδιαπραγμάτευτη η εγκυρότητα του. Επίσης, τα Oracles δεν εισάγουν την πληροφορία στο blockchain αλλά αποθηκεύουν όλα τα δεδομένα από τον εξωτερικό κόσμο προκειμένουν κάποιος κόμβος να μπορεί να εξάγει την πληροφορία που επιθυμεί. [37] Όπως είχαμε αναφέρει μία από τις μεγαλύτερες δυνανατότητες των Oracles είναι η εισαγωγή τυχαιότητας.

Κάποια από τα δεδομένα που μπορεί να συλλέξουν τα oracles είναι νικητές λοτταρίας, φυσικές καταστροφές, καιρικές συνθήκες, πολιτικά γεγονότα, αθλητικά γεγονότα, ατυχήματα. Επίσης, υπάρχουν πολλά είδη Oracles :

- Hardware που μπορεί να είναι σένσορες όπως IoT συσκεύες.
- Software που εξάγουν τα δεδομένα μέσω διεπαφών.
- Consensus όταν επιτυγχάνεται ομοφωνία από τα αποκεντρωμένα oracles.
- Inbound που πρέπει να ικανοποιηθεί μία συνθήκη για στείλουν δεδομένα.
- Outbound που τα συμβόλαια στέλνουν δεδομένα προς τον εξωτερικό κόσμο.

Η αδυναμία επιβεβαίωσης των δεδομένων που παρέχονται από τα oracles ονομάζεται “Oracles Problem”. Μία λύση για το συγκεκριμένο πρόβλημα μπορεί να είναι ένα σύστημα αποκεντρωμένων oracles, βασισμένο στη φήμη προκειμένου να αναπαραχθεί το σύστημα ομοφωνίας του blockchain. Επιθέσεις που έχουν στηριχθεί σε χειραγώγηση των oracles DeFi liquidations, έχουν επιφέρει ζημίες των 100 εκ. Δολαρίων . [3]

4. ΕΡΓΑΛΕΙΑ STATIC ANALYSIS ΠΟΥ ΑΝΙΧΝΕΥΟΥΝ BUGS

Όπως είδαμε στο προηγούμενο κεφάλαιο τα smart contracts, από τη στιγμή που εισέρχονται στο blockchain δεν επιδέχονται αλλαγών, συνεπώς η ποιότητα του κώδικα είναι βαρύνουσας σημασίας. Για να επιτευχθεί η μέγιστη ποιότητα ο προγραμματιστής των smart contract θα πρέπει να χρησιμοποιεί static analysis tools προκειμένου να βεβαιώνει ότι το smart contract που θα ανεβάσει στο blockchain δεν έχει κάποια από την ευπάθεια όπως αυτές που μελετήσαμε προηγουμένως.

Επί της ουσίας με τη στατική ανάλυση ελέγχουμε τον κώδικα χωρίς αυτός να εκτελείται και βρίσκουμαι προγραμματιστικά λάθη, συντακτικά λάθη, παραβιάσεις ασφαλείας, παραβιάσεις των standards ακόμα και κώδικα που δε χρησιμοποιείται. [38] Τα εργαλεία στατικής ανάλυσης χρησιμοποιούν διαφορετικές μεθόδους για να υλοποιήθουν και μπορούν να κατηγοριοποιηθούν ακολούθως [39]:

- Model Checking: Κατά το οποίο το πρόγραμμα μοντελοποιείται σα μια μηχανή πεπερασμένων καταστάσεων.
- Control flow analysis: Κατά το οποίο κατασκευάζεται ένας κατευθυνόμενος γράφος(CFG), που αναπαριστά μπλόκς του κώδικα προκειμένου να είναι εμφανής οι εξαρτήσεις του κώδικα. Ο προαναφερθέν γράφος για τη κατασκευή του βασίζεται στο Abstract Syntax Tree.

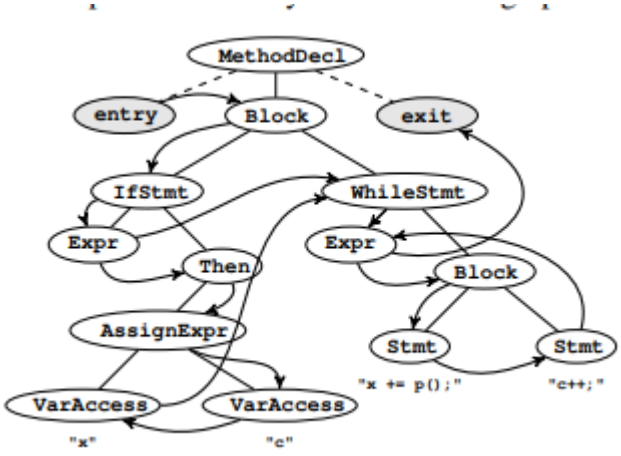

Figure 13:Control Flow graph [40]

- Data-flow analysis: Αντίστοιχα κατασκευάζεται ένας CFG, αλλά η κύρια του λειτουργία είναι να δείξει της εξαρτήσεις που έχουν τα δεδομένα ενός συστήματος.
- Symbolic analysis: Κατά την οποία το εργαλείο λαμβάνει υπόψιν τις μεταβλητές του προγράμματος.

Τα εργαλεία static analysis που θα αναλύσουμε είναι το Slither, το Ethlint, το Mythril, SmartCheck που παρέχουν πληθώρα επιλογών. Πέρα από τις γνωστές επιθέσεις που θα

αναλύσουμε θα πάρουμε και contracts από το Ethereum blockchain προκειμένου να επεξεργαστούμε κατά πόσο έχουν ευπάθειες.

4.1. Αξιολόγηση των εργαλείων

Εκτός από τις γνωστές ευπάθειες που θα μελετήσουμε, θα εξάγουμε και τον κώδικα ενός smart contract που υπάρχει στο ethereum blockchain προκειμένου να το αναλύσουμε. Προκειμένου να εξάγουμε ένα συμβόλαιο που θα ικανοποιεί κάποιες προυποθέσεις χρησιμοποιήσαμε την υπηρεσία smart corpus [41]. Συνεπώς έχουμε τη δυνατότητα να επιλέξουμε την έκδοση του compiler, τον αριθμό των γραμμών του συμβολαίου, τον αριθμό των συναρτήσεων, τον αριθμό των modifiers, τον αριθμο των payables.

Το smart corpus [41] είναι μία πρωτοποριακή πλατφόρμα που μαζεύει τα συμβόλαια, αφαιρεί τα διπλότυπα που έχουν τον ίδιο κώδικα αλλά διαφορετική address, μοντελοποιεί τα συμβόλαια βάση των metadata και εν τέλει παρουσίαζει τα αποτέλεσματα σε ένα GUI περιβάλλον.

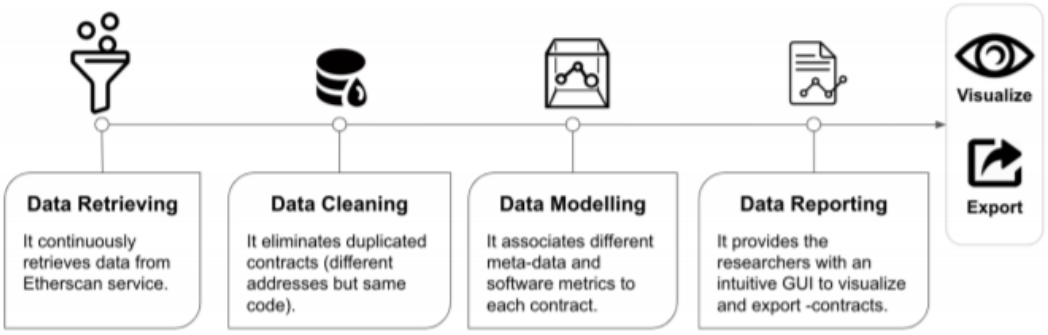

Figure 14: Smart corpus pipeline. [41]

Στο σύνολο υπάρχουν πάνω από 30 εργαλεία τα οποία βρίσκουν ευπάθειες πάνω σε συμβόλαια και αναλύουν τον κώδικα της Solidity. Όπως έχει εξεταστεί σε μία πληθώρα συμβολαίων, σε σχεδόν 48.000 συμβόλαια, το 97% εξαυτών εντοπίστηκε με κάποιου είδους ευπάθεια. [42]

4.2. Slither

Το Slither είναι ένα εργαλείο στατικής ανάλυσης του κώδικα μέσω του οποίου μπορούμε να επιτύγχουμε εύρεση ευπαθειών, πιθανές βελτιστοποιήσεις του κώδικα, κατανόηση του κώδικα και μπορεί να χρησιμοποιηθεί για code review. [43] Το Slither είναι ένα εργαλείο που έχει υλοποιηθεί με Python3 και η υλοποίηση του Slither μπορεί να σπάσει σε τρία βήματα :

- Αρχικά, χρησιμοποιεί σαν όρισμα το Abstract Syntax Tree που παράγεται από το compiler του Solidity.
- Έπειτα ο κώδικας μετατρέπεται σε SlithIR για ενδιάμεση αναπαράσταση.
- Στο τελευταίο στάδιο πραγματοποιείται η πραγματική ανάλυση του κώδικα που παρέχει και την πληροφορία στα υπόλοιπα μέρη του συστήματος.

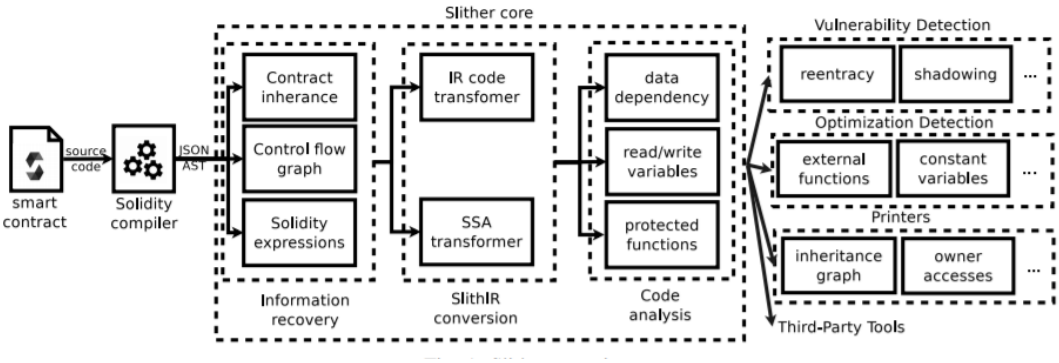

Figure 15: Slither overview [43]

Με το Slither ο προγραμματιστής θα μπορεί να ελέξει τον κώδικα του και συγκεκριμένα το Slither θα πρέπει να τηρεί κάποιες συγκεκριμένες προυποθέσεις [43]:

- Ευρωστία, θα πρέπει να λειτουργεί κάτω υπό οποιεσδήποτε συνθήκες.
- Ακρίβεια, θα πρέπει να βοηθά το πρόγραμμα στατικής ανάλυσης να βρίσκει τυχόν ευπάθειες με λίγα false positive.
- Απόδοση, θα πρέπει είναι γρήγορο και εύκολα να εγκαθίσταται σε εργαλέια ανάπτυξης λογισμικού.
- Σωστό επίπεδο abstraction, καθώς θα πρέπει να είναι εύκολα επεκτάσιμο για τυχόν νέους detector ή λειτουργικότητες.
- Να περιλαμβάνει ένα σέτ από detectors που θα είναι κοινοί για την πλεοψηφία των συμβολαίων.

Το slither διαθέτει πληθώρα από bugs detectors που συνέχεια εμπλουτίζονται, προς το παρόν υπάρχουν παραπάνω από 70 κάποια από αυτά είναι τα εξής : [44]

- Shadowing: Για τις μεταβλητές που είναι είτε τοπικές είτε καθολικές και μπορεί να προκάλεσουν δυσεύρετα bugs.
- Unitialized variables: Για τις μεταβλητες που δεν αρχικοποιούνται.
- Reentrancy: Η ευπάθεια που περιγράψαμε και παραπάνω και που ευθύνεται και το DAO attack. [2]

Επίσης, μέσω του Slither μπορούμε να επιτύγχουμε βελτιστοποιήσεις στον κώδικα οι οποίες θα βελτιώσουν το performance του κώδικα κάποιες από αυτές είναι οι εξης:

- Μεταβλητές που θα έπρεπε να είχαν δηλωθεί σαν constant καθώς οι constant μεταβλητές βοηθούν το compiler να βελτιστοποιήσει τον κώδικα.

- Συναρτήσεις που θα έπρεπε να είχαν δηλωθεί σαν externals. Αντίστοιχα, ο compiler έχει τη δυνατότητα να βελτιστοποιήσει τον κώδικα.

Ένα από τα πιο σηματικά εργαλεία που μας δίνεται από το Slither είναι οι printers καθώς έτσι επιτυγχάνουμε καλύτερη κατανόηση του κώδικα του συμβολαίου. Καταρχάς, μπορούμε να εξάγουμε διάφορα γραφήματα όπως το inheritance graph είτε το Control Glow Graph. Επίσης, έχουμε τη δυνατότητα να δούμε μία σύνοψη των ευπαθειών είτε των βελτιστοποιήσεων που απαιτούνται από το contract καθώς και μία σύνοψη των authorization accesses.

Επίσης, μπορεί να χρησιμοποιηθεί για codes reviews καθώς μπορούν να εισαχθούν συγκεκριμένοι περιορισμοί που θα πρέπει να ικανοποιούνται στον κώδικα του συμβολαίου.

4.2.1. Απόκριση του Slither σε γνωστές ευπάθειες

Στην παρούσα ενότητα θα δούμε κατά πόσον το Slither μπορεί να ανιχνεύσει τις ευπάθειες που παρουσιάμε στην ενότητα 3.3, όπως επίσης και τις συνόψεις του εργαλείου.

Re-entrancy:

Το Slither επιτυχώς αναγνωρίζει τη συγκεκριμένη ευπάθεια και εκτυπώνει μηνύματα λάθους για το re-entrancy. Εκτός από τη συγκεκριμένη ευπάθεια, το εργαλείο στατικής ανάλυσης μέσω των μηνυμάτων που εκτυπώνει μας ενημερώνει ότι η έκδοση του compiler που χρησιμοποιούμε είναι παρωχημένη, ότι το timestamp comparison δεν είναι καλή τεχνική, σχετικά με τα low level calls, όπως επίσης για το γεγονός ότι δε χρησιμοποιούμε constanst μεταβλητές.

```
INFO:Detectors:
Reentrancy in EtherStore.withdrawFunds(uint256) ():
        External calls:
        - require(bool)(msg.sender.call.value(_weiToWithdraw)()) ()
        State variables written after the call(s):
        - balances[msg.sender] -= _weiToWithdraw ()
        - lastWithdrawTime[msg.sender] = now ()
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
EtherStore.withdrawFunds(uint256) () uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(now >= lastWithdrawTime[msg.sender] + 604800) ()
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Pragma version^0.4.10 () allows old versions
solc-0.4.10 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in EtherStore.withdrawFunds(uint256) ():
        - require(bool)(msg.sender.call.value(_weiToWithdraw)()) ()
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter EtherStore.withdrawFunds(uint256)._weiToWithdraw () is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
EtherStore.withdrawalLimit () should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
depositFunds() should be declared external:
        - EtherStore.depositFunds() ()
withdrawFunds(uint256) should be declared external:
        - EtherStore.withdrawFunds(uint256) ()
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:re-entrancy.sol analyzed (1 contracts with 72 detectors), 9 result(s) found
```

Figure 16:Output of Slither for re-entrancy.

Πέρα από την ανίχνευση ευπαθειών το Slither μας δίνει τη δυνατότητα για περεταίρω ανάλυση του contract δίνοντας μία σύνοψη της συνάρτησης με τα εξής δεδομένα:

- Function
- Visibility
- Modifiers
- Read
- Write
- Internal Calls
- External Calls

```
alonso@ubuntu:~/thesis/known_attacks$ slither re-entrancy.sol  --print function-summary
INFO:Printers:
Contract EtherStore
Contract vars: ['withdrawalLimit', 'lastWithdrawTime', 'balances']
Inheritance:: []

+-------------------------------+------------+-----------+-----------------------------------+-----------------------------------+-------------------+------------------------------------------------------------
-------------------------------+
|            Function           | Visibility | Modifiers |                Read               |               Write               |   Internal Calls  |                             External Calls
                               |
+-------------------------------+------------+-----------+-----------------------------------+-----------------------------------+-------------------+------------------------------------------------------------
-------------------------------+
|         depositFunds()        |   public   |     []    |    ['balances', 'msg.sender']     |            ['balances']           |         []        |                                   []
                               |
|                               |            |           |           ['msg.value']           |                                   |                   |
                               |
|     withdrawFunds(uint256)    |   public   |     []    | ['balances', 'lastWithdrawTime'] | ['balances', 'lastWithdrawTime'] | ['require(bool)'] | ['msg.sender.call.value(_weiToWithdraw)()', 'msg.sende
r.call.value(_weiToWithdraw)'] |
|                               |            |           | ['withdrawalLimit', 'msg.sender'] |                                   |                   |
                               |
|                               |            |           |              ['now']              |                                   |                   |
                               |
| slitherConstructorVariables() |  internal  |     []    |                 []                |        ['withdrawalLimit']        |         []        |                                   []
                               |
+-------------------------------+------------+-----------+-----------------------------------+-----------------------------------+-------------------+------------------------------------------------------------
-------------------------------+

+-----------+------------+------+-------+----------------+----------------+
| Modifiers | Visibility | Read | Write | Internal Calls | External Calls |
+-----------+------------+------+-------+----------------+----------------+
+-----------+------------+------+-------+----------------+----------------+

INFO:Slither:re-entrancy.sol analyzed (1 contracts)
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Figure 17:Function summary

<u>Arithmetic underflows/overflows:</u>

Όπως περιγράψαμε και στο section Ευπάθειες στη Solidity τα arithmetic overflows είναι ένα σύνηθες λάθος, στο συγκεκριμένο σενάριο ο compiler v0.4.16 χρησιμοποιείται. Παρομοίως, το Slither μας προειδοποιεί για τον παρωχημένο compiler που χρησιμοποιούμε καθώς και για για το timestamp comparison και για το external visibility, αλλά δε μας προειδοποιεί για arithmetic overflows.

```
alonso@ubuntu:~/thesis/known_attacks$ slither arithmetic_underflow_overflow.sol
INFO:Detectors:
TimeLock.withdraw() (arithmetic_underflow_overflow.sol#16-22) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(now > lockTime[msg.sender]) (arithmetic_underflow_overflow.sol#18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Pragma version^0.4.16 (arithmetic_underflow_overflow.sol#1) allows old versions
solc-0.4.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter TimeLock.increaseLockTime(uint256)._secondsToIncrease (arithmetic_underflow_overflow.sol#12) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
deposit() should be declared external:
        - TimeLock.deposit() (arithmetic_underflow_overflow.sol#7-10)
increaseLockTime(uint256) should be declared external:
        - TimeLock.increaseLockTime(uint256) (arithmetic_underflow_overflow.sol#12-14)
withdraw() should be declared external:
        - TimeLock.withdraw() (arithmetic_underflow_overflow.sol#16-22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:arithmetic_underflow_overflow.sol analyzed (1 contracts with 72 detectors), 7 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Figure 18:Output of Slither for arithmetic overflows.

Επιπλέον, μπορούμε να επιλέξουμε μέσω του –print human-summary το slither να παράξει μία σύνοψη για το έλεγχο που έτρεξε στο συμβόλαιο.

```
alonso@ubuntu:~/thesis/known_attacks$ slither arithmetic_underflow_overflow.sol --print human-summary
INFO:Printers:
Compiled with solc
Number of lines: 23 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 1 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 3
Number of low issues: 1
Number of medium issues: 0
Number of high issues: 0


+----------+-------------+------+------------+--------------+-------------+
|   Name   | # functions | ERCS | ERC20 info | Complex code |   Features  |
+----------+-------------+------+------------+--------------+-------------+
| TimeLock |      3      |      |            |      No      | Receive ETH |
|          |             |      |            |              |   Send ETH  |
+----------+-------------+------+------------+--------------+-------------+
INFO:Slither:arithmetic_underflow_overflow.sol analyzed (1 contracts)
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Figure 19:Summary of contract with arithmetic overflow.

Default visibilities:

Όπως είδαμε και στα δύο προηγούμενα παραδείγματα ο detector μπόρεσε να αναγνωρίσει δύο περιπτώσεις που το function θα έπρεπε να είχε δηλωθεί σαν external. Επίσης, επιτυγχάνει να αναγνωρίσει ότι το contract που του δίνουμε μπορεί να στείλει σε οποιονδήποτε το ποσό.

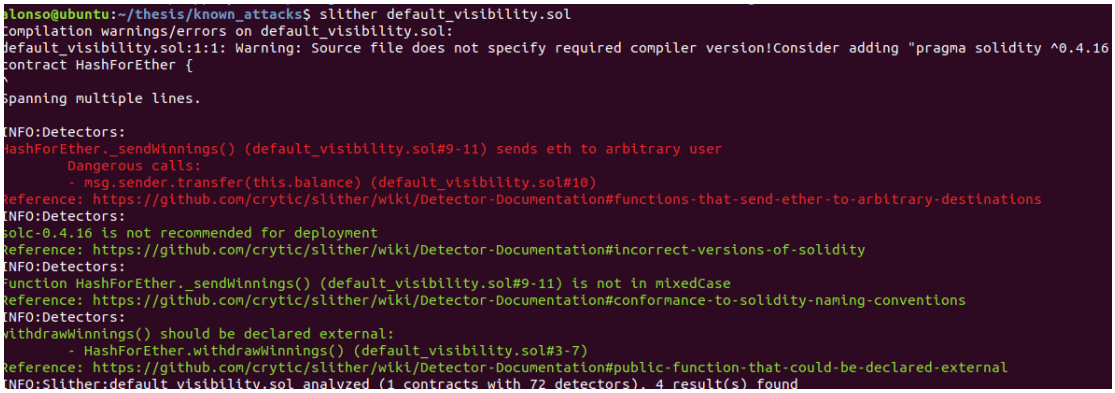

Figure 20:Output of Slither for default visibilities.

tx origin:

Χρησιμοποιώντας τον κώδικα που δείξαμε στο section 3.3.5 και έχοντας ορίσει το compiler στην έκδοση v0.7.4, το Slither αναγνωρίζει αρκέτα vulnerabilities καθώς και το πρωτεύον που είναι το tx.origin. Το Slither αναγνωρίζει 3 ευπάθειες στο συγκεκριμένο κώδικα:

- Send ether to arbitrary destinations.
- Dangerous usage of tx.origin.
- Missing zero address validation. (καθώς το address δεν αρχικοποιείται.)

Figure 21:Summary of Slither for tx.origin vulnerability.

Denial of Service:

Για αυτό το σενάριο χρησιμοποιούμε τον κώδικα που βρίσκεται στο repository not-so-smart-contract [45] καθώς και το compiler v0.4.10. Το Slither δεν ανιχνεύει ότι το

συμβόλαιο έχει ευπάθεια για DOS επίθεση όπως μπορούμε να δούμε και στο αποτέλεσμα.

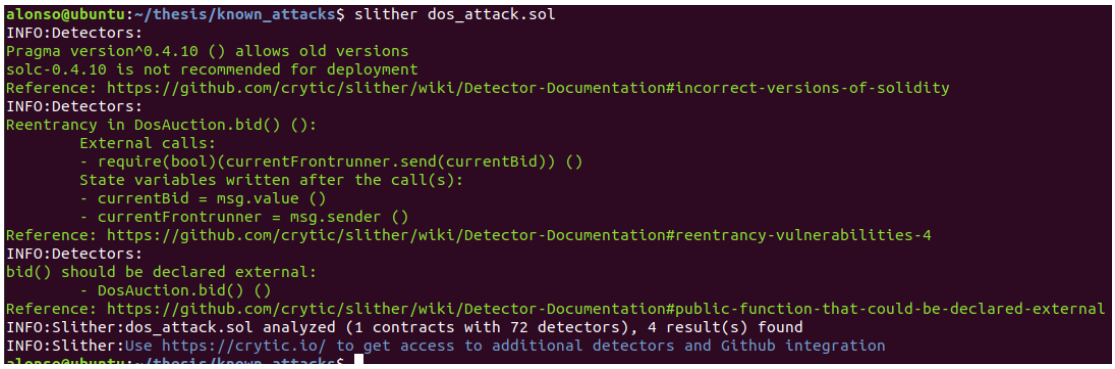

Figure 22:Output of Slither for DOS vulnerability.

Write to arbitrary Storage Location / Unitialized storage pointer:

Για να αξιολογήσουμε αυτή την ευπάθεια, χρησιμοποιούμε εκ νέου το compiler v0.4.10. Το εργαλείο Slither επιτυχώς βρίσκει την ευπάθεια καθώς διαθέτει και τον αντίστοιχο detector για unitialized storage variable. Όπως διακρίνουμε και από τον κώδικα το newRecord δεν έχει αρχικοποιηθεί συνεπώς μπορεί να αλλοιώσει τη τιμή του unlocked αναλόγως του ορίσματος που έχει δοθεί σα _name. Εφόσον και τα δύο έχουν αποθηκευτεί στο Slot1.

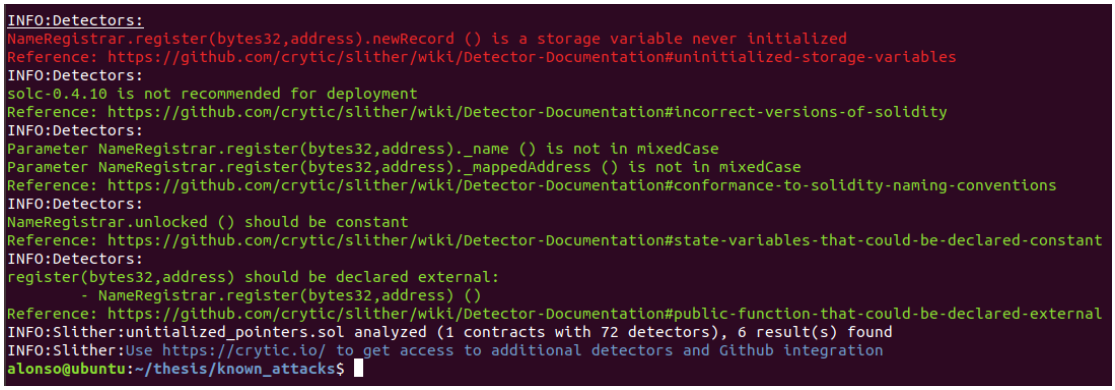

Figure 23: Output of Slither for unitialized pointes vulnerability.

Delegate Call to Untrusted Callee

Το Slither επιτυγχώς αναγνωρίζει ότι το συμβόλαιο που του δώσαμε σαν όρισμα έχει ευπάθεια στο Delegate Call. Ο compiler του συμβολαίου που χρησιμοποιήσαμε είναι ο v.0.4.11 συνεπώς μας προειδοποιεί για την παλαιότητα του compiler

χρησιμοποιούμε. Τέλος, αναγνωρίζει ότι το function pwn έπρεπε να είχε δηλωθεί σαν external καθώς δεν καλείται από το συμβόλαιο που ανήκει, έτσι εξοικωνομούμε gas.

```
alonso@ubuntu:~/thesis/known_attacks$ slither delegate_call.sol
INFO:Detectors:
Delegation.fallback() (delegate_call.sol#31-35) uses delegatecall to a input-controlled function id
        - delegate.delegatecall(msg.data) (delegate_call.sol#32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall
INFO:Detectors:
Delegate.Delegate(address)._owner (delegate_call.sol#9) lacks a zero-check on :
                - owner = _owner (delegate_call.sol#10)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Pragma version^0.4.11 (delegate_call.sol#1) allows old versions
solc-0.4.11 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Delegation.fallback() (delegate_call.sol#31-35):
        - delegate.delegatecall(msg.data) (delegate_call.sol#32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Redundant expression "this (delegate_call.sol#33)" inDelegation (delegate_call.sol#18-37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
pwn() should be declared external:
        - Delegate.pwn() (delegate_call.sol#13-15)
fallback() should be declared external:
        - Delegation.fallback() (delegate_call.sol#31-35)
```

Figure 24:Output of Slither for delegateCall

Assert Violation

Στο συγκεκριμένο σενάριο το Slither αποτυγχάνει να αναγνωρίσει την ευπάθεια σχετικά με το assert violation και μας ενημερώνει για την παλαιά έκδοση του compiler.

```
alonso@ubuntu:~/thesis/known_attacks$ slither assert_violation.sol
Compilation warnings/errors on assert_violation.sol:
assert_violation.sol:9:6: Warning: No visibility specified. Defaulting to "public".
            function check(){
     ^ (Relevant source part starts here and spans across multiple lines).

INFO:Detectors:
Pragma version^0.4.21 (assert_violation.sol#5) allows old versions
solc-0.4.21 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
check() should be declared external:
        - GasModel.check() (assert_violation.sol#9-14)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Figure 25:Output of Slither for assert violation.

Unprotected SELFDESTRUCT Instruction

Το Slither αναγνωρίζει επιτυγχώς την ευπάθεια.

```
onso@ubuntu:~/thesis/known_attacks$ slither self_destruct.sol
mpilation warnings/errors on self_destruct.sol:
lf_destruct.sol:5:4: Warning: No visibility specified. Defaulting to "public".
        function sudicideAnyone() {
 ^ (Relevant source part starts here and spans across multiple lines).

FO:Detectors:
mpleSuicide.sudicideAnyone() (self_destruct.sol#5-7) allows anyone to destruct the contract
ference: https://github.com/crytic/slither/wiki/Detector-Documentation#suicidal
FO:Detectors:
agma version^0.4.22 (self_destruct.sol#1) is known to contain severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
lc-0.4.22 is not recommended for deployment
ference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
FO:Detectors:
dicideAnyone() should be declared external:
        - SimpleSuicide.sudicideAnyone() (self_destruct.sol#5-7)
ference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Figure 26:Output of Slither for selfdestruction

Requirement Violation

Για τη συγκεκριμένη ευπάθεια το Slither αποτυγχάνει να την αναγνωρίσει.

```
lonso@ubuntu:~/thesis/known_attacks$ slither requirement_violation.sol
NFO:Detectors:
ragma version^0.4.25 (requirement_violation.sol#1) allows old versions
olc-0.4.25 is not recommended for deployment
eference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
NFO:Detectors:
oubleBaz() should be declared external:
        - Bar.doubleBaz() (requirement_violation.sol#5-7)
az(int256) should be declared external:
        - Foo.baz(int256) (requirement_violation.sol#11-14)
eference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
NFO:Slither:requirement_violation.sol analyzed (2 contracts with 72 detectors), 4 result(s) found
NFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Figure 27:Output of Slither for Requirement Violation.

ΕΥΠΑΘΕΙΑ	ΑΝΑΓΝΩΡΙΣΗ
RE-ENTRANCY	ΝΑΙ
ARITHMETIC OVERFLOWS	ΟΧΙ
DEFAULT VISIBILITY	ΝΑΙ
TX.ORIGIN	ΝΑΙ
DOS	ΟΧΙ
ARBITARY STORAGE	ΝΑΙ
DELEGATE CALL	ΝΑΙ
ASSERT VIOLATION	ΟΧΙ
UNPROTECTED SELFDESTRUCT	ΝΑΙ
REQUIREMENT VIOLATION	ΟΧΙ

Table 2:Vulnerabilities recognized by Slither,60 % accuracy.

4.2.2. Απόκριση του Slither σε ένα πραγματικό smart contract.

Χρησιμοποιώντας το smart corpus [41], εξάγαμε ένα συμβόλαιο που ικανοποιούσε τα κριτήρια που είχαμε επιλέξει (https://etherscan.io/address/0x2b3f2887c697b3f4f8d9f818c95482e1a3a759a5#code):

- Pragma version: 0.5.*
- Source lines of code: Greater than 10
- Number of functions: Greater than 10
- Number of modifiers: Greater than 10
- Number of payable: Greater than 10

Αρχικά, εξάγουμε το summary προκειμένου να έχουμε μία γρήγορη αποτίμηση της κατάστασης του συμβολαίου. Αξίζει να αναφέρουμε ότι ο compiler που χρησιμοποιούμε τρέχοντας το Slither είναι ο v.0.5.16. Από τη σύνοψη του συμβολαίου διακρίνουμε:

- Optimization issues : 32
- Informational issues : 12
- Low issues : 5
- Medium issues : 1
- High issues : 0

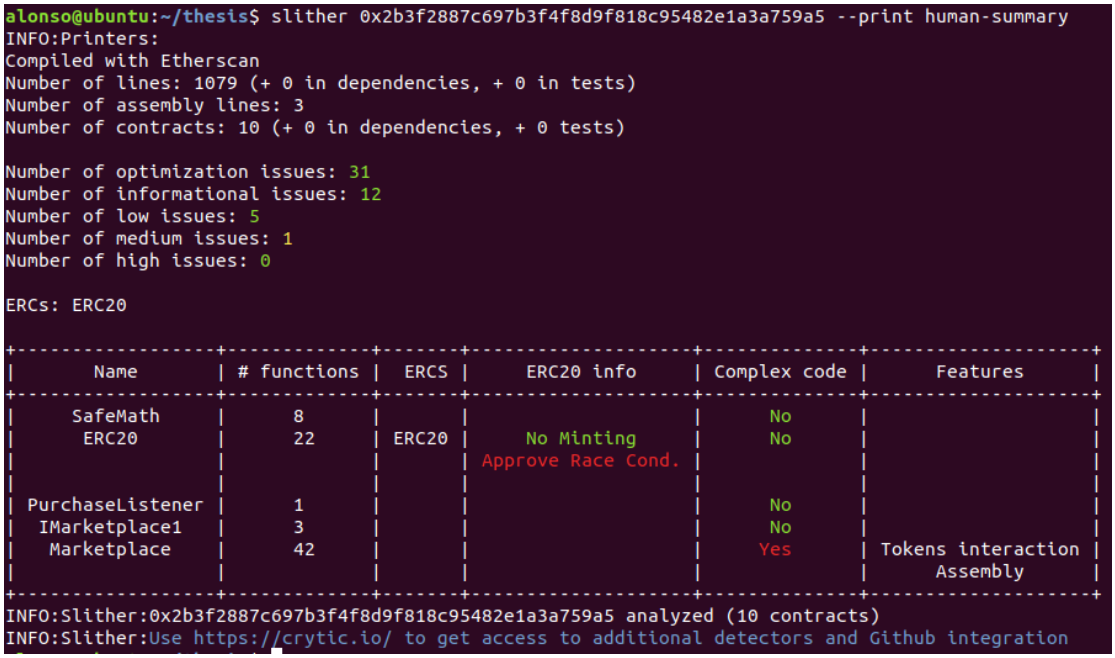

Figure 28: Summary of Slither for 0x2b3f2887c697b3f4f8d9f818c95482e1a3a759a5 contract.

Αντίστοιχα θα εξετάσουμε τα medium / low issues :

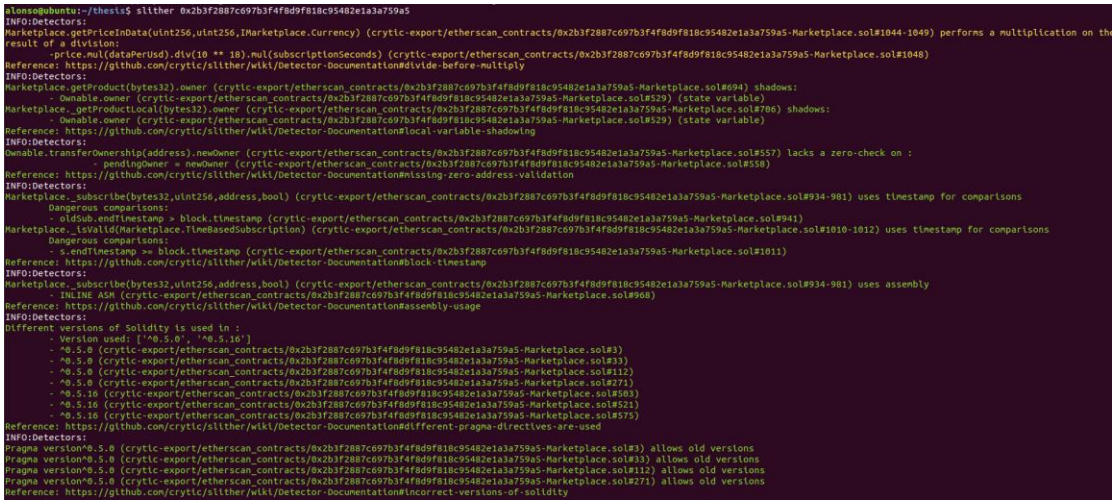

Figure 29:Output of Slither for 0x2b3f2887c697b3f4f8d9f818c95482e1a3a759a5 contract.

- Divide before multiply, που μπορεί να οδηγήσει σε απώλεια της ακρίβειας συνεπώς και σε λάθος απότελεσμα. Συνεπώς, είναι προτιμότερο να χρησιμοποιούμε πρώτα τον πολλαπλασιασμό πριν τη διαίρεση.

```
    * Helper function to calculate (hypothetical) subscription cost for given seconds and price, using current exchange rates.

function getPriceInData(uint subscriptionSeconds, uint price, Currency unit) public view returns (uint datacoinAmount) {

    if (unit == Currency.DATA) {

        return price.mul(subscriptionSeconds);

    }

    return price.mul(dataPerUsd).div(10**18).mul(subscriptionSeconds);

}
```

- Local Variable Shadowing [46], που μας δείχνει ότι υπάρχουν δύο μεταβλητές με το ίδιο όνομα στο ίδιο scope ή σε εμφωλευμένο. Όπως βλέπουμε και στον κώδικα η μεταβλητή owner δηλώνεται στην κλάση Ownable και αντίστοιχα χρησιμοποιείται και στην κλάση Marketplace που κληρονομεί τη κλάση

```
contract Ownable {
   address public owner;
   address public pendingOwner;

……
……
contract Marketplace is Ownable, IMarketplace2 {
……

function getProduct(bytes32 id) public view returns (string memory name, address owner, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds, ProductState state, bool requiresWhitelist) {
}
function _getProductLocal(bytes32 id) internal view returns (string memory name, address owner, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds, ProductState state, bool requiresWhitelist) {{
```

Ownable.

- Missing zero address validation, που μπορεί να οδηγήσει στην απώλεια του συμβολαίου για τον ιδιοκτήτη του, όπως βλέπουμε και στο παρακάτω κομμάτι του κώδικα σε περίπτωση που το newOwner είναι empty.

```
/**
 * @dev Allows the current owner to set the pendingOwner address.
 * @param newOwner The address to transfer ownership to.
 */
function transferOwnership(address newOwner) public onlyOwner {
   pendingOwner = newOwner;
```

- Use timestamps for comparison, το οποίο είναι επικίνδυνο καθώς μπορεί χειραγωγηθεί από τους miners καθώς όπως εξηγήσαμε στο section 3.3.4 το Ethereum blockchain δε διαθέτει πηγή τυχαιότητας και πρέπει να χρησιμοποιηθεί κάποιο oracle για να εισαχθεί από τον εξωτερικό κόσμο. Για να μπορέσει ένας miner να χειραγωγήσει το timestamp θα πρέπει να ικανοποιεί δύο συνθήκες: [47]

    - Δε μπορεί να χρησιμοποιήσει νωρίτερο χρόνο από του πατέρα του.
    - Δε μπορεί να έχει μεγάλη χρονική απόσταση από τη σύγχρονη ώρα.

```
if (oldSub.endTimestamp > block.timestamp) {
    require(addSeconds > 0, "error_topUpTooSmall");
    endTimestamp = oldSub.endTimestamp.add(addSeconds);
    oldSub.endTimestamp = endTimestamp;
    emit SubscriptionExtended(p.id, subscriber, endTimestamp);
}
```

- Assembly usage, καθώς είναι erroneous να αναμιγνύουμε high-level language(Solidity) με low-level language(Assembly).

```
assembly { codeSize := extcodesize(recipient) }  // solium-disable-line
security/no-inline-assembly
```

- Unchecked low level calls, καθώς άμα αποτύχει η κλήση το Ether θα κλειδωθεί στο contract. Συνεπώς, πρέπει πάντα να ελέγχουμε το τι επιστρέφει ένα low level call είτε να το logάρουμε. Εν προκειμένω, το contract ελέγχει το αποτέλεσμα της low-level κλήσης συνεπώς έχουμε false positive. [48]

```
        (bool success, bytes memory returnData) = recipient.call(

abi.encodeWithSignature("onPurchase(bytes32,address,uint256,uint256,uint256)",
            productId, subscriber, oldSub.endTimestamp, price, fee)
        );
        if (success) {
            (bool accepted) = abi.decode(returnData, (bool));
            require(accepted, "error_rejectedBySeller");
        }
```

Πέρα από τα issues / optimizations που μας προσέφερε το static analysis tool όπως είδαμε και παραπάνω μας προσφέρει επιπλέον εργαλεία για να κατανοήσουμε τον κώδικα.

- Κατά αρχάς ένα από τα πιο σημαντικά στοιχεία για την κατανόηση του κώδικα είναι η κατανόηση της ιεραρχίας. Με την ακόλουθη εντολή –print inheritance-graph μπορούμε να εξάγουμε την ιεραρχία του contract. Όπως, βλέπουμε και από το Figure 30 η κλάση Marketplace κληρονομεί τη κλάση Ownable και την κλάση IMarketplace2. Επίσης, από το γράφημα βλέπουμε ξεκάθαρα κάθε κλάση τα functions που έχει με το visibility αντίστοιχα καθώς και τα variables που υπάρχουν. Οι συναρτήσεις που έχουν πορτοκαλί χρώμα overrides μία συνάρτηση της parent's class. e.g getPriceInData. Στην περίπτωση που είχαμε μία μεταβλητή στην Derived Class που έκανε shadow μεταβλητή της Parent Class τότε θα η μεταβλητή θα είχε χρώμα κόκκινο.

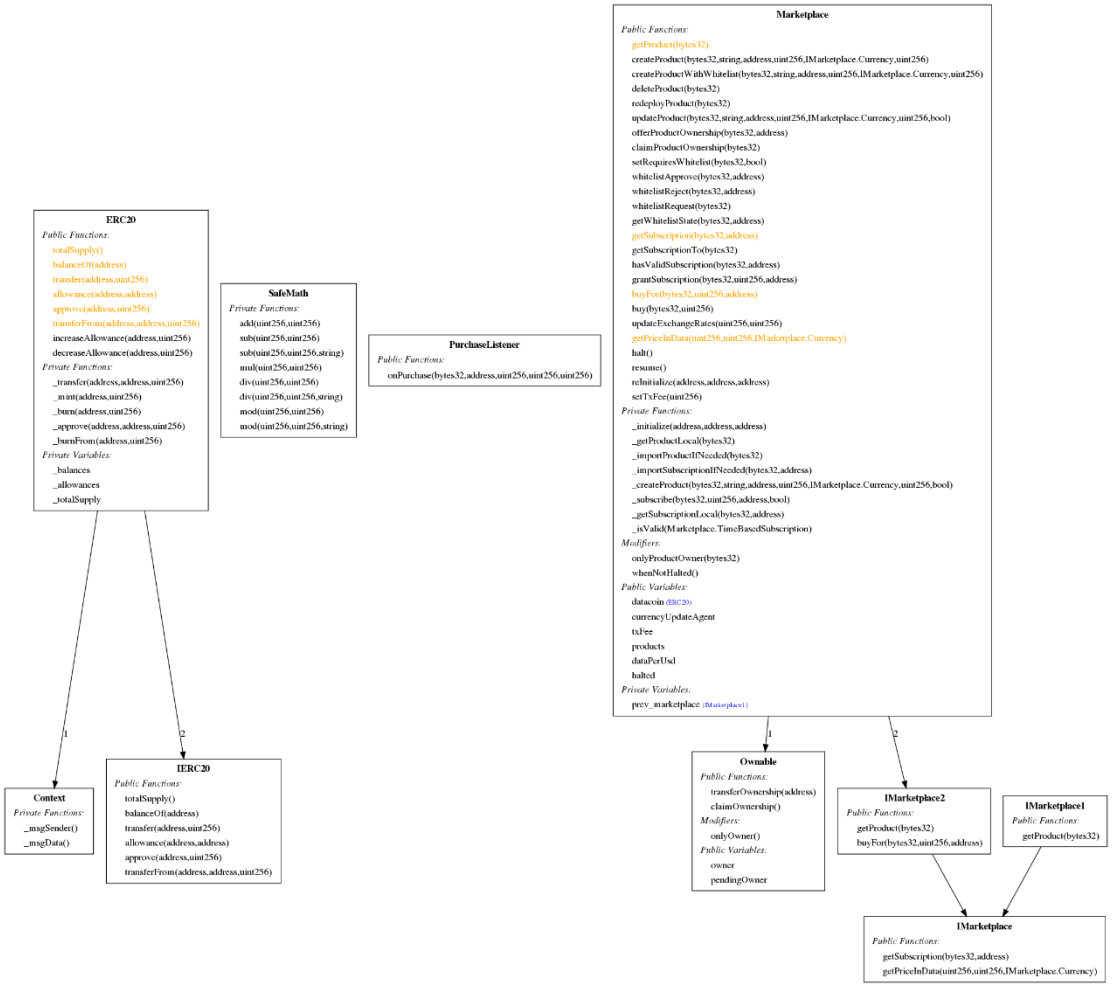

Figure 30:Class hierarchy for 0x2b3f2887c697b3f4f8d9f818c95482e1a3a759a5 contract.

- Αντίστοιχα, μπορούμε με την εντολη –print call-graph να δούμε τις κλήσεις των συναρτήσεων και πως αλληλεπιδρουν μεταξύ τους. Έτσι, επιτυγχάνουμε να έχουμε πλήρη κατανόηση της αρχιτεκτονικης του συμβολαιού.

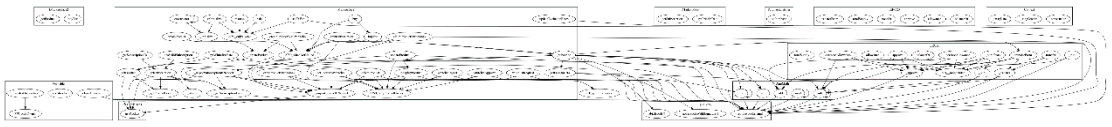

Figure 31:Call graph for 0x2b3f2887c697b3f4f8d9f818c95482e1a3a759a5 contract.

- Τέλος, με την εντολή –print modifiers μπορούμε να δούμε όλους τους modifiers που καλούνται για κάθε συνάρτηση για να εντοπίσουμε πιθανές ευπάθειες.

Figure 32: Modifiers of contract.

Επίσης δειγματοληπτικά επιλέξαμε άλλα δύο συμβόλαια 0xD81975505034f9a8C3618Faf65E9e9F06A9D698D, 0xc6c97d38CE7589c0881f6F845aC035042f088650 τα οποία εμφάνισαν πληθώρα ευπαθειών και πιθανών βελτιστοποιήσεων. Συνεπώς, παρατηρούμε ότι και τα τρία συμβόλαια που εξετάσαμε έχουν ευπάθειες και αναδεικνύει τη χρησιμότητα των εργαλείων στατικής ανάλυσης.

Για το συμβόλαιο 0xD81975505034f9a8C3618Faf65E9e9F06A9D698D έχουμε τις κάτοθι ευπάθειες:

- Local variable shadowing (Low risk)
- Missing zero address validation (Low risk)
- Assembly usage (Low risk)

Figure 33:Summary report for 0xd81975505034f9a8c3618faf65e9e9f06a9d698d contract.

Αντίστοιχα το συμβόλαιο 0xc6c97d38CE7589c0881f6F845aC035042f088650 παρουσιάζει τις κάτοθι ευπάθειες:

- Array length assignment (High risk)
- Dangerous strict equalities (Medium risk)
- Re-entrancy vulnerabilities (Medium risk)
- Un-used return (Medium risk)
- Re-entrancy vulnerabilities (2/3) (Low risk)
- Timestamp comparisons (Low risk)

```
alonso@ubuntu:~/thesis$ slither 0xc6c97d38CE7589c0881f6F845aC035042f088650 --print human-summary
INFO:Printers:
Compiled with Etherscan
Number of lines: 622 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 7 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 20
Number of informational issues: 25
Number of low issues: 11
Number of medium issues: 4
Number of high issues: 1
ERCs: ERC20

+-----------+-------------+-------+--------------------+--------------+--------------------+
|    Name   | # functions |  ERCS |     ERC20 info     | Complex code |      Features      |
+-----------+-------------+-------+--------------------+--------------+--------------------+
|  SafeMath |      8      |       |                    |      No      |                    |
|   ERC20   |      7      | ERC20 |     No Minting     |      No      |                    |
|           |             |       | Approve Race Cond. |              |                    |
|           |             |       |                    |              |                    |
| FixedSwap |      37     |       |                    |      No      |    Receive ETH     |
|           |             |       |                    |              |      Send ETH      |
|           |             |       |                    |              | Tokens interaction |
+-----------+-------------+-------+--------------------+--------------+--------------------+
INFO:Slither:0xc6c97d38CE7589c0881f6F845aC035042f088650 analyzed (7 contracts)
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Figure 34:Summary of contract 0xc6c97d38CE7589c0881f6F845aC035042f088650 which its value is ~$51,000.

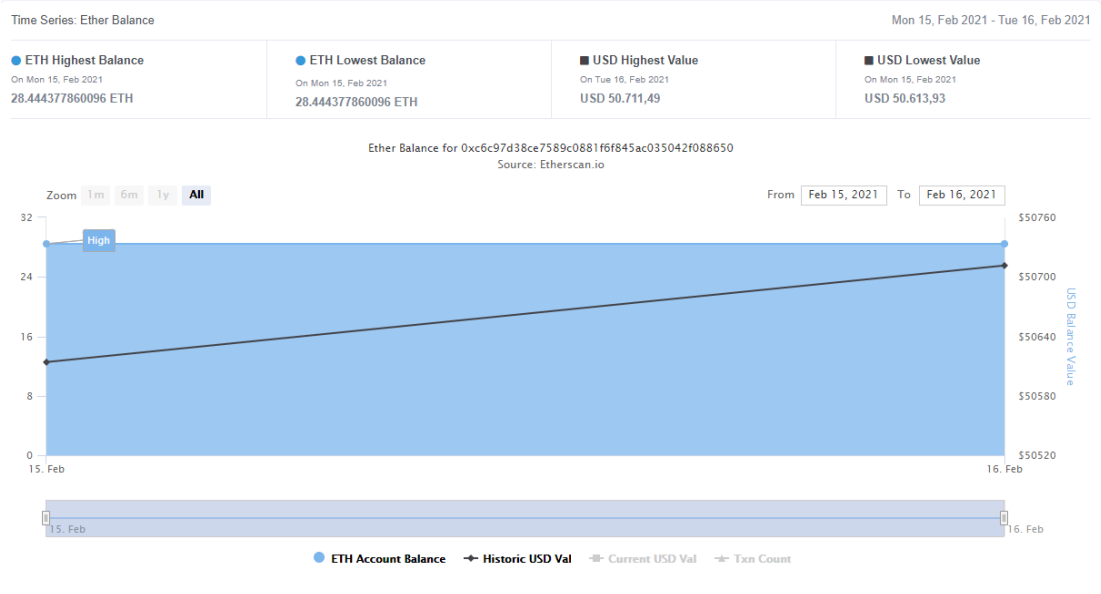

Figure 35:Balance of the contract 0xc6c97d38CE7589c0881f6F845aC035042f088650

4.3. Ethlint

Το ethlint [49] (ex called Solium) είναι ένα από τα πιο γνωστά εργαλεία στατικής ανάλυσης για smart contracts. Εκτός από την εύρεση ευπαθειών μπορεί να χρησιμοποιηθεί και για παρατηρήσεις για το στυλ του κώδικα βασισμένες στο Solidity Style Guide [50].

Επίσης, σου δίνει την ευελιξία μέσω του `.soliumrc.json` να παραμετροποιήσεις τους κανόνες στατικής ανάλυσης που θα τρέξεις για τον κώδικα σου. Το Ethlint

εγκαθίσταται έχοντας το security plugin το οποίο περιέχει τους κανόνες για τις καλύτερες πρακτικές ασφαλείας βασισμένες το Consensys Best Practices [51]. Στη σελίδα Security Plugin for Solium [52] ο αναγνώστης μπορεί να βρει αναλυτική περιγραφή των κανόνων.

Εν προκειμένω, για να αναλύσουμε τα προαναφερθέντα συμβόλαια είχαμε το κάτοθι configuration:

```
{
  "extends": "solium:recommended",
  "plugins": [
    "security"
  ],
  "rules": {
    "quotes": [
      "error",
      "double"
    ],
    "security/enforce-explicit-visibility": [
      "error"],
    "security/no-throw": [
      "error"],
    "security/no-tx-origin": [
      "error"],
    "security/enforce-explicit-visibility": [
      "error"],
    "security/no-block-members": [
      "error"],
    "security/no-call-value": [
      "error"],
    "security/no-fixed": [
      "error"],
    "security/no-inline-assembly": [
      "error"],
    "security/no-low-level-calls": [
      "error"],
    "security/no-modify-for-iter-var": [
      "error"],
    "security/no-send": [
      "error"],
    "security/no-sha3": [
      "error"],
    "security/no-unreachable-code": [
      "error"],
    "security/no-call-value": [
      "error"],
    "security/no-fixed": [
      "error"],
    "security/no-inline-assembly": [
      "error"]

  }
}
```

Figure 36:soliumrc.json configuration.

Αρχικά, αναλύουμε τις γνωστές επιθέσεις που παρουσιασάμε στην ενότητα Ευπάθειες στη Solidity.

ΕΥΠΑΘΕΙΑ	**ΑΝΑΓΝΩΡΙΣΗ**
RE-ENTRANCY	ΟΧΙ
ARITHMETIC OVERFLOWS	ΟΧΙ
DEFAULT VISIBILITY	ΝΑΙ
TX.ORIGIN	ΝΑΙ
DOS	ΟΧΙ

ARBITARY STORAGE	OXI
DELEGATE CALL	NAI
ASSERT VIOLATION	OXI
UNPROTECTED SELFDESTRUCT	OXI
REQUIREMENT VIOLATION	OXI

Table 3:Vulnerabilities recognized by Ethlint, 30 % accurnacy.

Επίσης, αναλύουμε το συμβόλαιο 0x2b3f2887c697b3f4f8d9f818c95482e1a3a759a5 για τις ευπάθειες που έχει και για τις προτάσεις πάνω στο στυλ του κώδικα.

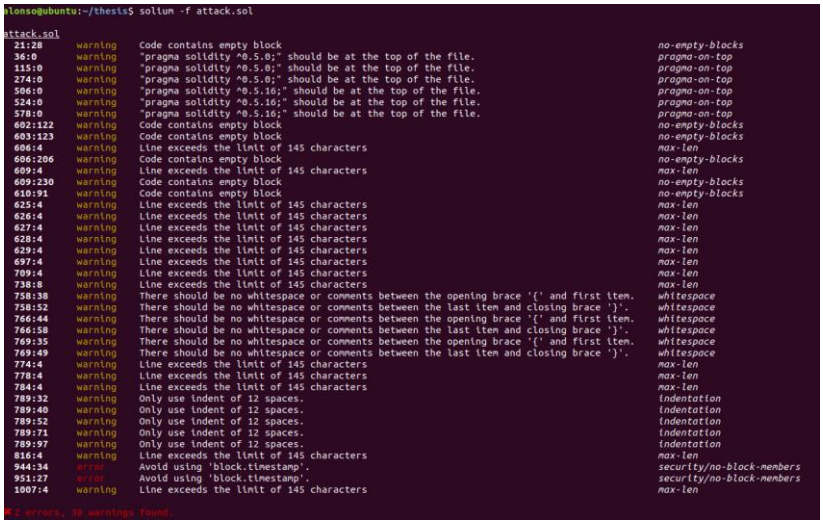

Figure 37:Summary of Ethlint for contract 0x2b3f2887c697b3f4f8d9f818c95482e1a3a759a5..

Όπως φαίνεται και από την εικόνα, υπάρχουν πολλά hints για διορθώσεις στο στυλ του κώδικα καθώς και δύο λάθη σχετικά με το block timestamp όπως αναφέραμε και στο section 4.2.2. Παρόλα αυτά παρατηρούμε ότι πολλές ευπάθειες που είχαμε εντοπίσει με το Slither δεν κατέστη δυνατό να τις εντοπίσουμε στο Ethlint, αυτό έγκειται σε δύο λόγους :

- Ο συντάκτης του συμβολαίου έχει ορίσει comment directives προκειμένου κάποιες ευπάθειες να μην εμφανίζονται με το Linter. π.χ

```
if (codeSize > 0) {
    // solium-disable-next-line security/no-low-level-calls
    (bool success, bytes memory returnData) = recipient.call(
        abi.encodeWithSignature("onPurchase(bytes32,address,uint256,uint256,uint256)",
        productId, subscriber, oldSub.endTimestamp, price, fee)
    );
```

Στην προκειμένη περίπτωση ο συντάκτης γνωρίζει ότι κάνει έλεγχο του αποτελέσματος του συμβολαίου, συνεπώς ο έλεγχος θα ήταν false positive.

- Κάποιες ευπάθειες που μπορούσε να τις εντοπίσει το Slither, αδυνατεί να τις βρει το Solium. π.χ divide before multiplication.

4.4. Mythril

Το Mythril είναι ένα εργαλείο στατικής ανάλυσης το οποίο λειτουργεί σε επίπεδο EVM bytecode. Το Mythril έχει την χαρακτηριστική ονομασία «swiss army knife of smart contracts security» και πρωτό εισήχθη το 2018. [53] Η διαδικασία που χρησιμοποιεί το Mythril προκειμένου να κάνει την ανάλυση είναι η ακόλουθη: [54]

- Εξάγει τον κώδικα του EVM bytecode κάνοντας compile τον κώδικα.
- Αρχικοποιεί την κατάσταση του λογαριασμού εκτελώντας το bytecode είτε εξάγοντας τα δεδομένα απευθείας από τον κόμβο.
- Εκτελεί τον κώδικα symbolically, προκειμένου να εξερευνήσει όλες τις πιθανές καταστάσεις του προγράμματος για n συναλλαγές.
- Όταν συναντά μία απροσδόκητη κατάσταση, βάση των λογικών καταστάσεων διαπιστώνει εάν η προαναφερθείσα είναι έγκυρη.

Το Mythril χρησιμοποιεί το back-end symbolic execution Laser Ethereum. [55] Το Symbolic Virtual Machine τρέχει Mythril Disassembly objects αντί για Ethereum Bytecode. Το Laser επιστρέφει ένα Object το οποίο περιέχει την κατάσταση του συμβολαίου και το αναπαριστά με ένα γράφο.

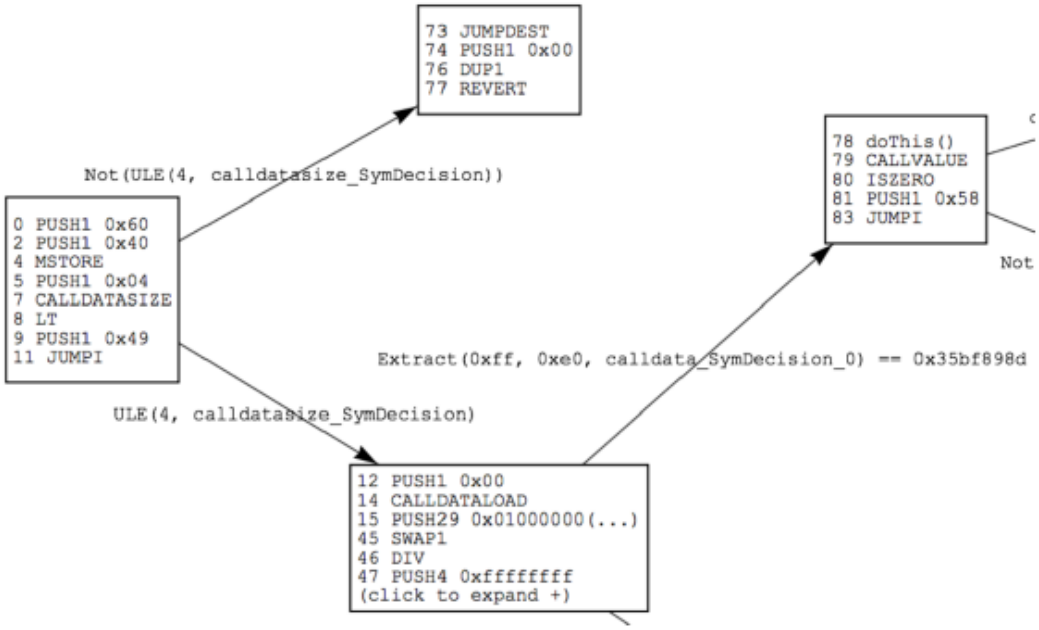

Figure 38:Control flow graph of Laser. [56]

Κάθε κόμβος στο γράφο αναπαριστά και ένα βασικό μπλοκ του κώδικα που εκτελείται και περιέχει και μία λίστα από καθολικές μεταβλητές. Μία κατάσταση ισοδυναμεί με τη θέση που έχει ο program counter. Επίσης, μία λίστα των ακμών μεταξύ των κόμβων παρέχεται. Ακόμα, για να επιλυθούν τα προβλήματα λογικής χρησιμοποιούνται SMT solvers προκειμένου να αποσαφηνιστεί εάν ικανοποιείται η λογική συνθήκη για την ευπάθεια του κώδικα. Εν προκειμένω, ο solver που χρησιμοποιείται είνναι ο Z3 Solver.

4.4.1. Ανάλυση του Mythril

Το Mythril υποστηρίζει τα εξής security analysis modules:

- Delegate Call to Untrusted Contract
- Dependence on Predictable Variables
- Deprecated Opcodes
- Ether Thief
- Exceptions
- External Calls
- Integer (Overflows)
- Multiple Sends
- Suicide
- State Change External Calls
- Unchecked Retval
- User supplied assertion
- Arbitrary Storage Write
- Arbitrary Jump

Συντακτικό των εντολών που τρέχουν στο Mythril:

- *Produce Control Flow Graph: myth -g example.html example.sol*
- *Produce Report: myth -x example.sol –verbose-report*
- *Select specific security analysis module: myth – x example.sol – m integer s*
- *Select max depth : myth -x example.sol –max-depth 24*

Ένα στιγμιότυπο κατά το run του Mythril που μπορεί να αναλυθεί εώς εξής:

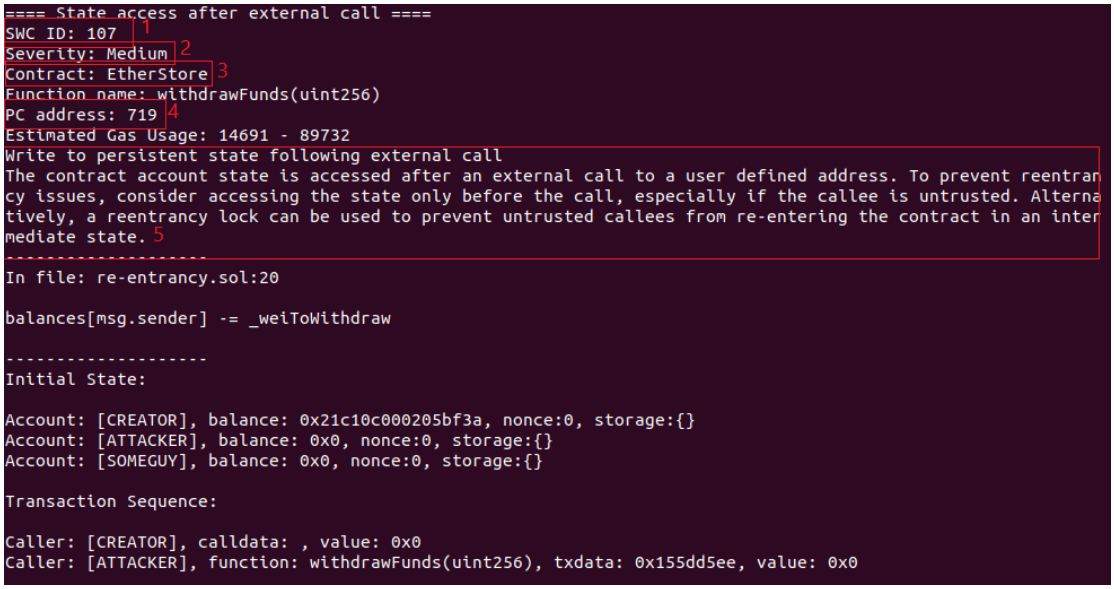

Figure 39:Mythril output.

1. SWC_ID: Δηλώνει το είδος της ευπάθειας που αναγνώρισε swcregistry.io. π.χ https://swcregistry.io/docs/SWC-107
2. Τη σοβαρότητα της ευπάθειας.
3. Το όνομα του συμβολαίου.
4. PC address, του συμβολαίου.
5. Μία αναλυτική περιγραφή της ευπάθειας.

4.4.2. Απόκριση του Mythril σε γνωστές ευπάθειες της Solidity

ΕΥΠΑΘΕΙΑ	ΑΝΑΓΝΩΡΙΣΗ
RE-ENTRANCY	NAI
ARITHMETIC OVERFLOWS	NAI
DEFAULT VISIBILITY	OXI
TX.ORIGIN	NAI
DOS	OXI
ARBITARY STORAGE	OXI
DELEGATE CALL	NAI
ASSERT VIOLATION	NAI
UNPROTECTED SELFDESTRUCT	NAI
REQUIREMENT VIOLATION	OXI

Table 4:Vulnerabilities recognized by Mythril, 60 % accuracy.

4.5. Smart Check

Το Smart Check [57] είναι εργαλείο στατικής ανάλυσης, γραμμένο σε Java. Χρησιμοποιεί XML δέντρα σαν ενδιάμεση αναπαράσταση τα οποία τα συγκρίνει με τα

XPath μοτίβα. Αυτή η μέθοδος στατικής ανάλυσης έχει αρκετούς περιορισμούς καθώς δεν υποστηρίζει περίπλοκους κανόνες αφού δεν υποστηρίζονται από τα XPath. Οι ευπάθειες που αναγνωρίζονται από το Smart Check είναι οι ακόλουθες :

- Balance equality.
- Unchecked external call.
- DOS.
- Send instead of transfer.
- Re-entrancy.
- Malicious libraries.
- Tx.origin.
- Integer division.
- Locked money.
- Unchecked math.
- Timestamp pendence.
- Unsafe type inference.
- Transfer forwards all gas.
- Byte array.
- Costly loop.
- Compiler version fixed.
- Private modifier.
- Redundant fallback function.
- Style guide violation.
- Implicit visibility level.

4.5.1. Απόκριση του SmartCheck σε γνωστές επιθέσεις

ΕΥΠΑΘΕΙΑ	ΑΝΑΓΝΩΡΙΣΗ
RE-ENTRANCY	ΟΧΙ
ARITHMETIC OVERFLOWS	ΟΧΙ
DEFAULT VISIBILITY	ΝΑΙ
TX.ORIGIN	ΝΑΙ
DOS	ΟΧΙ
ARBITARY STORAGE	ΟΧΙ
DELEGATE CALL	ΟΧΙ
ASSERT VIOLATION	ΟΧΙ
UNPROTECTED SELFDESTRUCT	ΟΧΙ
REQUIREMENT VIOLATION	ΟΧΙ

Table 5:Vulnerabilities recognized by Smart Check 20% accuracy.

Το smart check είναι ένα πολύ εύχρηστο εργαλείο το οποίο παρέχει τη δυνατότητα να τεστάρεις το κομμάτι του κώδικα που επιθυμείς μέσω Web σελίδας. Παρόλα αυτά κυμαίνεται στα επίπεδα του Ethlint με 20% ακρίβειας στις ευπάθειες που το τεστάραμε.

4.6. SmartBugs

Το SmartBugs [58] είναι μία πρωτοποριακή πλατφόρμα που υποστηρίζει 10 εργαλεία στατικής ανάλυσης προκειμένου ο προγραμματιστής να έχει την ευχαίρεια να αξιολογήσει το συμβόλαιο του και με τα 10 εργαλεία ταυτόχρονα. Επίσης, διαθέτει 2 datasets, το πρώτο είναι σχετικά μικρό με 143 συμβόλαια και το δεύτερο είναι μία μεγάλη συλλογή με 45,518 συμβόλαια. Αυτό δίνει την ευκολία σε κάποιον ερευνήτη να συγκρίνει τα εργαλεία με έναν εύκολο τρόπο. Τα εργαλεία που υποστηρίζει το SmartBugs είναι τα εξής HoneyBadger, Maian, Manticore, Mythril, Osiris, Oyente, Securify, Slither, SmartCheck, Solhint.

4.7. Σύγκριση των εργαλείων

Επίσης, με την μελέτη των παραπάνω εργαλείων στατικής ανάλυσης διαπιστώσαμε τα πλεονεκτήμα και τα μειονεκτήματα του εκάστοτε εργαλείου. Το Slither ήταν το εργαλείο που ξεχωρίσαμε καθώς είναι πάρα πολύ γρήγορο από άποψη απόδοσης και επίσης διαθέτει μία πληθώρα detectors που ανανεώνεται τακτικά. Το Mythril είναι ένα εργαλείο στατικής ανάλυσης σε επίπεδο EVM bytecode και είναι αρκετά αργό σε σχέση με τα προηγούμενα εργαλεία. Παρόλα αυτά είναι το πιο αναλυτικό και μπορούμε να εξάγουμε αρκετά περισσότερα δεδομένα σε σχέση με τα προηγούμενα εργαλεία. Το Ethlint και το Smart Check ήταν τα εργαλεία με τη χαμηλότερη απόδοση καθώς δεν εντόπισαν τις βασικές ευπάθειες παρόλα αυτά είναι χρήσιμα βοηθητικά εργαλεία.

	Mythril	Ethlint	Slither	Smart Check
RE-ENTRANCY	X	-	X	-
ARITHMETIC OVERFLOWS	X	-	-	-
DEFAULT VISIBILITY	-	X	X	X
TX.ORIGIN	X	X	X	X
DOS	-	-	-	-
ARBITARY STORAGE	-	-	X	-

DELEGATE CALL	X	X	X	-
ASSERT VIOLATION	X	-	-	-
UNPROTECTED SELFDESTRUCT	X	-	X	-
REQUIREMENT VIOLATION	-	-	-	-

Table 6:Comparison of Static analysis tools.

Τα ευρήματα μας συμφωνούν και με μία έρευνα που έχει γίνει σε περισσότερα από 47,000 συμβόλαια [42] και το εργαλείο με την καλύτερη ακρίβεια ήταν το Mythril 27%, ακολουθούσε το Slither με 17 % και τέλος ήταν το SmartCheck με 11%.

4.8. Βελτιστες τεχνικες συνταξης εξυπνων συμβολαιων

Όπως έγινε κατανοητό τα έξυπνα συμβόλαια πρέπει να πληρούν τις αυστηρότερες των προδιαγραφών προκειμένου να ενταχθούν στο blockchain. [59] Κατά αρχάς θα πρέπει να λαμβάνουν υπόψιν πιθανά λάθη και να τερματίζουν ομαλά. Επίσης, θα ήταν φρόνιμο πριν ανέβει ένα συμβόλαιο στο blockchain να έχει ανέβει σαν test συμβόλαιο. Μία από τις βασικές αρχές του προγραμματισμού είναι το Single Responsibility Principle, που σημαίνει ότι κάθε function θα πρέπει να εκτελεί μόνο μία λειτουργία κάτι αντίστοιχο θα πρέπει να συμβαίνει και με τα συμβόλαια και με το πως είναι διαρθρωμένος ο κώδικας. Επιπλεόν, θα πρέπει να είναι updated ο compiler στην τελευταία του έκδοση καθώς και με την υιοθέτηση ενός πλέγματος από Tools.

Μία από τις κατηγορίες που θα πρέπει να λάβει υπόψιν του ο προγραμματιστής είναι οι ιδιότητες του blockchain και πως μπορούν να επηρεάσουν το συμβόλαιο του.

- Θα πρέπει να είναι ιδιαίτερα προσκεκτικός με τις εξωτερικές κλήσεις καθώς μπορεί να εκτελέσει κακόβουλο κώδικα και αλλαχθεί η ροή εκτέλεσης.
- Θα πρέπει να λαμβάνει υπόψιν το κόστος του gas που θα χρησιμοποιηθεί.
- Τα timestamps μπορεί να αλλαχθούν συνεπώς θα πρέπει να είναι πολύ προσεκτικός όταν τα χρησιμοποιεί.
- Στα blockchains δεν υπάρχει τυχαιότητα συνεπώς για να την εισάγει θα πρέπει να χρησιμοποιήσει Oracles.

Από προγραμματιστικής άποψης, το συμβόλαιο θα πρέπει να είναι αρθρωτό, προκειμένου κομμάτια του κώδικα είναι επαναχρησιμοποιήσιμα και να τηρεί τις βασικές προγραμματιστικές προυποθέσεις:

- Ο προγραμματιστής θα πρέπει να είναι επιφυλακτικός όταν κάνει εξωτερικές κλήσεις καθώς κλήσεις σε μη έμπιστα συμβόλαια μπορούν να προκαλέσουν σφάλματα στην εκτέλεση του κώδικα του είτε να εκτελέσουν κακόβουλο κώδικα.
- Αποφυγή αλλαγής κατάστασης μετά από εξωτερικές κλήσεις καθώς υπάρχει ο κίνδυνος κακόβουλος κώδικας να έχει εκτελεστεί.
- Αποφυγή χρησιμοποίησης των συναρτήσεων transfer(), send() αντιθέτως να χρησιμοποιείται η συνάρτηση call().
- Χερισμός των σφαλμάτων κατά την κλήση εξωτερικών συναρτήσεων όπως των exceptions. Επίσης, χρήσιμο θα ήταν κάθε εξωτερική κλήση να απομονώνεται με μία ξεχωριστή συναλλαγή.
- Η μη χρησιμοποίηση delegatecall σε μη έμπιστο κώδικα.
- Τα δεδομένα που υπάρχουν τα συμβόλαια είναι public, συνεπώς ο προγγραμματιστής θα πρέπει να είναι ιδιαίτερα προσεκτικός στο τι ανεβάζει.
- Κατά τη σύνταξη των συμβολαίων θα πρέπει να λαμβάνει υπόψιν την πιθάνοτητα ένα από τα δύο μέρη μπορεί να αποσυρθεί. Παρόλα αυτά θα πρέπει το συμβόλαιο να συνέχισει κανονικά την εκτέλεση του.
- Θα πρέπει να λαμβάνει υπόψιν του πιθανά underflows/overflows.

Τέλος, σε περίπτωση που ο προγραμματιστής χρησιμοποιεί τη Solidity, θα πρέπει να λάβει υπόψιν του της εξής πρακτικές :

- Να χρησιμοποιεί την assert(), προκειμένου να ελέγχει δυναμικά ότι ικανοποιούνται οι συνθήκες που έχει θέσει.
- Η χρησιμοποίηση των modifiers μόνο για τους ελέγχους και όχι για εξωτερικές κλήσεις.
- Προσοχή με τη χρησιμοποίηση της διαίρεσης ακεραίων.
- Οι fallback συναρτήσεις θα πρέπει να παραμένουν απλές και να ελέγχεται το μέγεθος των δεδομένων.
- Κάθε συνάρτηση που δέχεται ether θα πρέπει να έχει το αναγνωριστικό payable.
- Σε κάθε συνάρτηση θα πρέπει να δηλώνεται το visibility της.
- Θα πρέπει να χρησιμοποιούνται Events για να αποθηκεύεται σε Logs η δραστηριότητα του συμβολαίου.
- Ο προγραμματιστής θα πρέπει να προσέχει τις shadowed μεταβλητές.
- Η αποφυγή της χρήσης του tx.origin για να επιβεβαίωσει την ταυτότητα του owner.
- Η αποφυγή της χρήσης των timestamps.
- Ιδιαίτερη προσοχή με τη χρήση πολλαπλής ιεραρχίας καθώς μπορεί να οδηγήσει σε Diamond Problem. [60]
- Η αποφυγή της χρήσης του extcodesize για τον έλεγχο εξωτερικών λογαριασμών.

5. Επιλογος

5.1. Σύνοψη

Με την ολοκλήρωση της μεταπτυχιακής εργασίας φάνηκε η αναγκαιότητα χρήσης εργαλείων στατικής ανάλυσης από τους συντάκτες των smart contract. Συλλέξαμε τρία συμβόλαια από το blockchain και τα τρία παρουσίασαν ευπάθειες. Συνεπώς, γίνεται έκδηλο ότι οι προγραμματιστές κάνουν λάθη και λάθη που μπορεί να επιφέρουν την απώλεια χρημάτων. Μοναδική λύση για να μειώσουμε τα λάθη είναι η ένταξη στα coding practices και στο pipeline που χρησιμοποιούν εργαλεία στατικής ανάλυσης. Επίσης, ο συνδιασμός τέτοιων εργαλείων μπορεί να αυξήσει και την εύρεση ευπαθειών όπως κατέδειξε και η παρούσα μεταπτυχιακή εργασία.

5.2. Μελλοντική εργασία

Τα επόμενα βήματα θα είναι η ανάλυση περισσότερων εργαλείων όπως εργαλειών που αναλύουν το EVM code. Επίσης, η υλοιποίηση υπηρεσίας που θα τραβάει αυτόματα τα contracts που πρώτο εισέρχονται στο blockchain, θα τα αναλύει και θα βρίσκει πιθανές ευπάθειες.

ΠΗΓΕΣ

[1] «Tesla buys $1.5 billion in bitcoin, plans to accept it as payment,» [Ηλεκτρονικό]. Available: https://www.cnbc.com/2021/02/08/tesla-buys-1point5-billion-in-bitcoin.html.

[2] «DAO Attack,» [Ηλεκτρονικό]. Available: https://www2.deloitte.com/ie/en/pages/technology/articles/DAO-Attack-Analysis.html.

[3] S. o. a. c. $. i. E. D. liquidations. [Ηλεκτρονικό]. Available: https://cryptoslate.com/seeming-oracle-attack-causes-100m-in-ethereum-defi-liquidations/.

[4] Mozzila. [Ηλεκτρονικό]. Available: https://developer.mozilla.org/en-US/docs/Archive/Security/Confidentiality,_Integrity,_and_Availability.

[5] «Wikipedia,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Cryptographic_hash_function.

[6] H. Zhao. [Ηλεκτρονικό]. Available: https://medium.com/@zhaohuabing/hash-pointers-and-data-structures-f85d5fe91659.

[7] «Satoshi Nakamoto,» [Ηλεκτρονικό]. Available: https://bitcoin.org/bitcoin.pdf.

[8] «OREILLY,» [Ηλεκτρονικό]. Available: https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch06.html#full_node_reference.

[9] OREILY, «Chapter 5. Transactions,» [Ηλεκτρονικό]. Available: https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch05.html.

[10] «Bitcoin Wiki,» [Ηλεκτρονικό]. Available: https://en.bitcoin.it/wiki/Transaction.

[11] «Bitcoin Improvement Proposals Wiki,» [Ηλεκτρονικό]. Available: https://en.bitcoin.it/wiki/Bitcoin_Improvement_Proposals.

[12] «BBC,» [Ηλεκτρονικό]. Available: https://www.bbc.com/news/technology-48853230.

[13] «Cambridge,» [Ηλεκτρονικό]. Available: https://cbeci.org/.

[14] «Vitalik Buterin,» [Ηλεκτρονικό]. Available: https://ethereum.org/en/whitepaper/.

[15] T. T.. [Ηλεκτρονικό]. Available: https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf.

[16] «Ethereum Devs,» [Ηλεκτρονικό]. Available: https://ethereum.org/en/developers/docs/evm/.

[17] «EVM py,» [Ηλεκτρονικό]. Available: https://github.com/ethereum/py-evm.

[18] «EVM C,» [Ηλεκτρονικό]. Available: https://github.com/ethereum/evmone.

[19] «EVM GO,» [Ηλεκτρονικό]. Available: https://github.com/hyperledger/burrow.

[20] «Nick Szabo,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Nick_Szabo.

[21] «Ethereum Oracles Medium,» [Ηλεκτρονικό]. Available: https://medium.com/decentlabs/building-your-first-ethereum-oracle-1ab4cccf0b31.

[22] M. B. a. T. C. Nicola Atzei, «A survey of attacks on Ethereum smart contracts».

[23] «A Survey on Ethereum Systems Security:,» [Ηλεκτρονικό]. Available: https://arxiv.org/pdf/1908.04507.pdf.

[24] Solidity. [Ηλεκτρονικό]. Available: https://docs.soliditylang.org/en/latest/security-considerations.html.

[25] [Ηλεκτρονικό]. Available: https://github.com/crytic/not-so-smart-contracts.

[26] «https://medium.com/spankchain/we-got-spanked-what-we-know-so-far-d5ed3a0f38fe,» [Ηλεκτρονικό]. Available: We Got Spanked: What We Know So Far.

[27] «Solidity Security,» [Ηλεκτρονικό]. Available: https://blog.sigmaprime.io/solidity-security.html#reentrancy.

[28] «openzeppelin-contracts,» [Ηλεκτρονικό]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol.

[29] «An investigation into a denial of service attack on an ethereum,» [Ηλεκτρονικό]. Available: https://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1219&context=ism.

[30] «Delegatecall to Untrusted Callee,» [Ηλεκτρονικό]. Available: https://swcregistry.io/docs/SWC-112.

[31] «The Parity Wallet Hack Explained,» [Ηλεκτρονικό]. Available: https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/.

[32] «Use of Deprecated Solidity Functions,» [Ηλεκτρονικό]. Available: https://swcregistry.io/docs/SWC-111.

[33] «Assert Violation,» [Ηλεκτρονικό]. Available: https://swcregistry.io/docs/SWC-110#gas_modelsol.

[34] «Unprotected SELFDESTRUCT Instruction,» [Ηλεκτρονικό]. Available: https://swcregistry.io/docs/SWC-106.

[35] «Write to Arbitrary Storage Location,» [Ηλεκτρονικό]. Available: https://swcregistry.io/docs/SWC-124#arbitrary_location_write_simplesol.

[36] «Requirement Violation,» [Ηλεκτρονικό]. Available: https://swcregistry.io/docs/SWC-123.

[37] «What are Oracles? Smart Contracts, Chainlink & “The Oracle Problem”,» [Ηλεκτρονικό]. Available: https://blockonomi.com/oracles-guide.

[38] «What are “static analysis” tools?,» [Ηλεκτρονικό]. Available: https://proandroiddev.com/what-are-static-analysis-tools-48ccff8135d4.

[39] Z. Z. Y. Roudier, «Static Code Analysis for Software Security Verification: Problems and Approaches,» σε *COMPSAC*, July 2014.

[40] E. S. ¨, «Extensible Intraprocedural Flow Analysis at the».

[41] G. A. P. R. T. M. Marchesi, «An Organized Repository of Ethereum Smart Contracts' Source Codes and Metrics,» 2020.

[42] T. D. J. F. F. P. C. R. Abreu, «Empirical Review of Automated Analysis Tools on 47,587».

[43] J. F. G. G. A. Groce, «Slither: A Static Analysis Framework For Smart,» [Ηλεκτρονικό]. Available: http://barbie.uta.edu/~xlren/SmartContract/slither.pdf.

[44] «Bugs and Optimizations Detection Slither,» [Ηλεκτρονικό]. Available: https://github.com/crytic/slither#bugs-and-optimizations-detection.

[45] «not-so-smart-contract,» [Ηλεκτρονικό]. Available: https://github.com/crytic/not-so-smart-contracts/blob/master/denial_of_service/auction.sol.

[46] «Shadowing variables,» [Ηλεκτρονικό]. Available: https://github.com/ethereum/solidity/issues/2563.

[47] «Block Timestamp Manipulation,» [Ηλεκτρονικό]. Available: https://solidity-by-example.org/hacks/block-timestamp-manipulation/.

[48] «Insecure Use of Low-Level Call,» [Ηλεκτρονικό]. Available: https://docs.guardrails.io/docs/en/vulnerabilities/solidity/use_of_low_level_call.

[49] «ETHLINT,» [Ηλεκτρονικό]. Available: https://ethlint.readthedocs.io/en/latest/.

[50] «Style Guide,» [Ηλεκτρονικό]. Available: https://docs.soliditylang.org/en/latest/style-guide.html.

[51] «Consensys Best Practices,» [Ηλεκτρονικό]. Available: https://consensys.github.io/smart-contract-best-practices/recommendations/.

[52] «The official Security Plugin for Solium,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/solium-plugin-security#list-of-rules.

[53] «Mythril Documentation,» [Ηλεκτρονικό]. Available: https://readthedocs.org/projects/mythril-classic/downloads/pdf/master/.

[54] «Practical Smart Contract Security Analysis and Exploitation,» [Ηλεκτρονικό]. Available: https://hackernoon.com/practical-smart-contract-security-analysis-and-exploitation-part-1-6c2f2320b0c.

[55] «Laser Ethereum,» [Ηλεκτρονικό]. Available: https://github.com/b-mueller/laser-ethereum.

[56] «Laser Ethereum,» [Ηλεκτρονικό]. Available: https://github.com/b-mueller/laser-ethereum.

[57] S. T. E. V. I. Ivanitskiy, «SmartCheck: Static Analysis of Ethereum Smart Contracts».

[58] P. C. T. D. R. A. João F. Ferreira, «SmartBugs: A Framework to Analyze Solidity Smart Contracts,» [Ηλεκτρονικό]. Available: https://arxiv.org/abs/2007.04771.

[59] «Ethereum Smart Contract Security Best Practices,» [Ηλεκτρονικό]. Available: https://consensys.github.io/smart-contract-best-practices/.

[60] «The diamond problem,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Multiple_inheritance#The_diamond_problem.

[61] M. K. J. B. S. S. NEVILLE GRECH, «MadMax: Surviving Out-of-Gas Conditions».

[62] Wiki, «AST,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Abstract_syntax_tree.

[63] «How to Secure Your Smart Contracts: 6 Solidity Vulnerabilities and how to avoid them (Part 1),» [Ηλεκτρονικό]. Available: https://medium.com/loom-

network/how-to-secure-your-smart-contracts-6-solidity-vulnerabilities-and-how-to-avoid-them-part-1-c33048d4d17d.