ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
Π. Μ. Σ. ΑΣΦΑΛΕΙΑ ΔΙΚΤΥΩΝ & ΑΣΥΡΜΑΤΩΝ ΣΥΣΚΕΥΩΝ

Ασφάλεια στην πλατφόρμα έξυπνων συμβολαίων Ethereum

(Security on the Ethereum Smart Contract Platform)

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κόρμπος Γεώργιος
Α.Μ: MTE1619

Επιβλέπων: Ξενάκης Χρήστος
Αναπληρωτής Καθηγητής

ΠΕΙΡΑΙΑΣ, ΙΑΝΟΥΑΡΙΟΣ 2019

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
Π. Μ. Σ. ΑΣΦΑΛΕΙΑ ΔΙΚΤΥΩΝ & ΑΣΥΡΜΑΤΩΝ ΣΥΣΚΕΥΩΝ

Ασφάλεια στην πλατφόρμα έξυπνων συμβολαίων Ethereum

(Security on the Ethereum Smart Contract Platform)

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κόρμπος Γεώργιος
Α.Μ: MTE1619

Επιβλέπων: Ξενάκης Χρήστος
Αναπληρωτής Καθηγητής

Εγκρίθηκε από την τριμελή επιτροπή στις 14 / 01 / 2019

Ξενάκης Χρήστος, Αναπληρωτής Καθηγητής, ... (Υπογραφή)
, ... (Υπογραφή)
, ... (Υπογραφή)

ΠΕΙΡΑΙΑΣ, ΙΑΝΟΥΑΡΙΟΣ 2019

Αφιερωμένο στη μητέρα μου και στο θείο μου.

Περίληψη

Ethereum είναι μια δημόσια διαδικτυακή πλατφόρμα ανοικτού κώδικα, η οποία είναι βασισμένη στην τεχνολογία blockchain και κατανεμημένων υπολογιστών. Επιπλέον, παρέχει σαν λειτουργία, τη φιλοξενία και την εκτέλεση των smart contracts, τα οποία είναι προγράμματα που λειτουργούν όπως ακριβώς έχουν φτιαχτεί. Το Ethereum δημοσιεύθηκε τον Ιούλιο του 2015 από τον ερευνητή και προγραμματιστή Vitalik Buterin.

Σε αυτή την διπλωματική εργασία θα παρουσιάσουμε την τεχνολογία blockchain, τους βασικούς πυλώνες λειτουργίας της και τις πιο σημαντικές αλλά και επικίνδυνες επιθέσεις που καλείται να αντιμετωπίσει. Για κάθε ευπάθεια της πλατφόρμας θα εξετάζουμε και τα αντίστοιχα αντίμετρα, τα οποία είτε έχουν ήδη υλοποιηθεί είτε θα υλοποιηθούν κατά την εξέλιξή της.

Έπειτα, θα στρέψουμε την προσοχή μας στην πλατφόρμα Ethereum και στα smart contracts. Τα smart contracts είναι προγράμματα τα οποία εκτελούνται στην πλατφόρμα του Ethereum και αλληλοεπιδρούν με τους χρήστες της, είτε παρέχοντας κάποια υπηρεσία σε αυτούς είτε προσφέροντας κάποιου είδους διασκέδαση. Όπως θα δούμε, αυτά τα προγράμματα, που μπορεί ο κάθε προγραμματιστής να γράψει και να ανεβάσει στην πλατφόρμα, θέλουν ιδιαίτερη προσοχή γιατί μπορεί να παρουσιάσουν διάφορες ευπάθειες. Ερευνώντας αυτές τις ευπάθειες και εξετάζοντας πραγματικά περιστατικά τα οποία έχουν συμβεί τα τελευταία χρόνια καταλήγουμε σε διάφορες καλές πρακτικές που πρέπει να ακολουθηθούν κατά την δημιουργία τους.

Στο τέλος της διπλωματικής σχολιάζουμε τα επόμενα βήματα της πλατφόρμας και πως θα εξελιχθεί ώστε να είναι πιο ασφαλής αλλά παράλληλα και πιο οικονομική στη χρήση της. Ένα από τα προβλήματα που θα πρέπει να αντιμετωπίσει είναι το πρόβλημα της κλιμάκωσης των χρηστών.

Στο τμήμα των παραρτημάτων παρατίθεται το πρόγραμμα που συντάξαμε το οποίο χρησιμοποιεί το Ethereum API για να αντλήσει πληροφορίες από την δημόσια αλυσίδα και τις συναλλαγές που αυτή περιέχει.

ΠΕΡΙΕΧΟΜΕΝΑ

1. Εισαγωγή στο blockchain

Ο όρος blockchain έκανε την πρώτη του επίσημη εμφάνιση το 2008 στην Ιαπωνία, όταν ένας προγραμματιστής, ή μια ομάδα προγραμματιστών, κάτω από το όνομα Satoshi Nakamoto, δημοσίευσε στον υπόλοιπο κόσμο το white paper του Bitcoin, ενός καινούργιου ψηφιακού νομίσματος το οποίο δεν ήταν κάτω από τον έλεγχο μιας τράπεζας ή χώρας, αλλά εξολοκλήρου κάτω από την εποπτεία των χρηστών της πλατφόρμας.

Για να χρησιμοποιήσει ένας χρήστης οποιαδήποτε πλατφόρμα blockchain και να πραγματοποιήσει μια συναλλαγή, πρώτα από όλα χρειάζεται να έχει έναν ενεργό λογαριασμό και ένα ψηφιακό πορτοφόλι το οποίο θα μπορεί να αποθηκεύσει τα ψηφιακά χρήματα που έχει στην κατοχή του. Με τη δημιουργία κάθε καινούργιου χρήστη, ο λογαριασμός αυτός συνδέεται με τρία πράγματα.

Πρώτα από όλα δημιουργείται το ιδιωτικό του κλειδί (private key), το οποίο γνωρίζει μόνο ο ίδιος. Στη συνέχεια χρησιμοποιείται ως είσοδος στην επόμενη διεργασία, που είναι η δημιουργία δημόσιου κλειδιού (public key), η οποία διαφέρει από πλατφόρμα σε πλατφόρμα. Αφότου δημιουργηθεί και το δημόσιο κλειδί του, η διεργασία δημιουργίας διεύθυνσης (address) λαμβάνει χώρα. Στο τέλος, ο νέος χρήστης έχει στην κατοχή του τρία κλειδιά διαφορετικού μήκους τα οποία χρησιμοποιούνται σε διαφορετικές περιστάσεις.

Μπορούμε να κατηγοριοποιήσουμε τους χρήστες της κάθε πλατφόρμας σε τρεις βασικές κατηγορίες. Η πρώτη κατηγορία είναι οι απλοί χρήστες, που χρησιμοποιούν την τεχνολογία αυτή για να κάνουν συναλλαγές, αγοροπωλησίες ή ανταλλαγές προϊόντων και υπηρεσιών. Η δεύτερη κατηγορία χρηστών είναι οι κόμβοι (nodes), που είναι χρήστες της πλατφόρμας. αλλά έχουν όπως θα δούμε και στη συνέχεια, μια σειρά από υποχρεώσεις που πρέπει να εκπληρώνουν. Τέλος, υπάρχουν οι εξορύκτες (miners), οι οποίοι προσπαθούν να βρουν λύσεις σε δύσκολα μαθηματικά προβλήματα ώστε να διεκδικήσουν κάποια ποσότητα από το αντίστοιχο συνάλλαγμα της κάθε πλατφόρμας και να δημιουργήσουν block τα οποία θα προστεθούν, αφού επικυρωθούν, στη βασική αλυσίδα της.

Το blockchain επιγραμματικά λειτουργεί ως εξής: ένας χρήστης πραγματοποιεί μια αγορά εκκινώντας μια σειρά από διεργασίες. Η πρώτη ενέργεια του χρήστη είναι να

πραγματοποιήσει τη συναλλαγή πληρώνοντας το κόστος αυτής. Στη συνέχεια αυτή η συναλλαγή δημοσιεύεται στους υπόλοιπους χρήστες της πλατφόρμας μέσω του δικτύου εμπιστοσύνης που έχει χτιστεί μέχρι εκείνη τη στιγμή, και καταλήγει σε μια λίστα με τις υπόλοιπες συναλλαγές που περιμένουν να επικυρωθούν. Παράλληλα, άλλοι χρήστες του δικτύου, που κατατάσσονται στην κατηγορία miners, προσπαθούν να λύσουν μια σειρά από δύσκολα μαθηματικά προβλήματα και να δημιουργήσουν νέα block τα οποία θα προστεθούν στη δημόσια αλυσίδα. Αυτή η αλυσίδα θεωρείται έμπιστη και την χρησιμοποιούν όλοι οι χρήστες της πλατφόρμας. Κάθε block που τοποθετείται στην πλατφόρμα είναι immutable, το οποίο σημαίνει ότι από τη στιγμή που θα δημιουργηθεί το περιεχόμενο του δεν μπορεί να τροποποιηθεί με κανέναν τρόπο. Αυτός ο μηχανισμός, δηλαδή η συνεχείς σύναψη των immutable block, μετατρέπει τη συνολική αλυσίδα σε ένα immutable αντικείμενο.

Όταν ένας εξορύκτης καταφέρει να λύσει το δύσκολο μαθηματικό πρόβλημα, δημοσιεύει την απάντηση του στο δίκτυο ώστε να επαληθευτεί από τους υπόλοιπους χρήστες. Έπειτα, μπορεί να επιλέξει έναν αριθμό από τις συναλλαγές που βρίσκονται στη λίστα μη επικυρωμένων και να τις τοποθετήσει μέσα στο block. Ο ίδιος χρήστης θα ανταμειφθεί με μια ποσότητα κρυπτονομίσματος για αυτή του την πράξη. Τέλος, αφού το block έχει δημιουργηθεί και έχει επισυναφθεί στην αλυσίδα, η συναλλαγή είναι πλέον επικυρωμένη και μπορεί να θεωρηθεί επιτυχής.

Με αυτόν τον τρόπο λειτουργούν όλες οι πλατφόρμες της συγκεκριμένης τεχνολογίας και οι διαφορές βρίσκονται στην υλοποίηση του κάθε βήματος, το οποίο συμβαδίζει με το επιπλέον κόστος της συναλλαγής, την επεξεργαστική ισχύ που απαιτείται για να ολοκληρωθεί και το ηλεκτρικό κόστος που προκύπτει από αυτό.

Η πλατφόρμα του Ethereum εξάλλου, δίνει τη δυνατότητα στους προγραμματιστές να αναπτύξουν τα δικά τους προγράμματα και να τα δημοσιεύσουν σε αυτή. Αυτά τα προγράμματα λέγονται smart contracts (έξυπνα συμβόλαια) και επιτρέπουν στους χρήστες να αλληλεπιδρούν μαζί τους και να τους παρέχουν επιπλέον υπηρεσίες, όπως παιχνίδια, ψηφιακά πορτοφόλια αλλά και υπηρεσίες ασφαλείας.

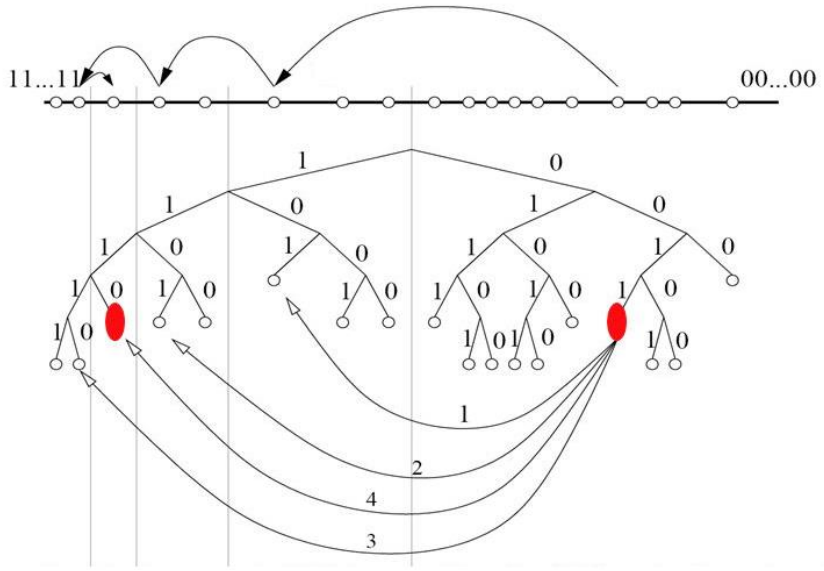
2. Το peer-to-peer δίκτυο, οι κόμβοι και οι συνδέσεις

2.1. Το peer-to-peer (P2P) δίκτυο του Ethereum

Το P2P δίκτυο του Ethereum είναι βασισμένο στο Kademlia DHT. Το Kademlia είναι σχεδιασμένο με τέτοιο τρόπο ώστε να κάνει αποδοτική αποθήκευση και αναζήτηση δεδομένων μέσα σε ένα αποκεντρωμένο δίκτυο, ενώ στο Ethereum χρησιμοποιείται για την αναζήτηση νέων χρηστών.

Ο κάθε κόμβος (node) ο οποίος είναι μέρος του δικτύου Kademlia, έχει στην κατοχή του έναν αριθμό από ευδιάκριτες λίστες, οι οποίες ονομάζονται buckets (b) και κάθε λίστα περιέχει έναν αριθμό από όμοιους χρήστες (k). Το όνομα της κάθε λίστας καθορίζει σε πόση απόσταση από αυτόν βρίσκονται οι υπόλοιποι χρήστες. Για παράδειγμα, η λίστα 10, ή αλλιώς bucket 10, ενός κόμβου θα περιέχει όλους τους χρήστες οι οποίοι βρίσκονται σε απόσταση 10. Έτσι όταν κάποιος χρήστης ζητήσει πρόσβαση σε κάποιο αρχείο (t), θα αναζητήσει μέσα στην λίστα του χρήστες οι οποίοι το έχουν, ή θα ζητήσει από αυτούς να κοιτάξουν μέσα στις δικές τους λίστες για να βρουν κάποιον χρήστη που το έχει στην κατοχή του. Αυτή η διαδικασία συνεχίζεται μέχρι να βρεθεί κάποιος χρήστης που έχει το αναζητούμενο αρχείο.

Στο Ethereum χρησιμοποιείται το ίδιο σύστημα με λίστες και χρήστες, όχι για την ανεύρεση δεδομένων αλλά για την αναζήτηση νέων χρηστών της πλατφόρμας. Στην πράξη αυτό γίνεται ως εξής: ένας κόμβος διαλέγει έναν τυχαίο στόχο, ελέγχει τις λίστες του και επιλέγει 16 χρήστες που είναι πιο κοντά σε αυτόν. Στη συνέχεια, ζητάει από τους επιλεγμένους στόχους, να του επιστρέψουν ο καθένας μια λίστα από 16 κόμβους τα οποία βρίσκονται ακόμα πιο κοντά στο στόχο, με αποτέλεσμα να έχει πλέον στην κατοχή του 16x16 νέους κόμβους, οι οποίοι θα ερωτηθούν το ίδιο ερώτημα και θα περιμένει να του επιστρέψουν περισσότερους κόμβους. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να μη βρεθούν άλλοι νέοι χρήστες.



Σχ. α. Διάγραμμα αναζήτησης κόμβων του δικτύου Kademlia.

2.2. Οι Κόμβοι (Nodes)

Σαν κόμβος ενός δικτύου ορίζεται το φυσικό εκείνο μηχάνημα το οποίο είναι συνδεδεμένο στο δίκτυο και χρησιμοποιεί έναν από τους διαθέσιμους clients της ανάλογης πλατφόρμας. Στο Ethereum μπορεί να είναι ένας από τους παρακάτω:

- Parity, γραμμένος στην γλώσσα προγραμματισμού Rust.
- Geth, γραμμένος στην γλώσσα προγραμματισμού Go.
- Ethereum(J), γραμμένος στην γλώσσα προγραμματισμού JAVA.
- Eth, γραμμένος στην γλώσσα προγραμματισμού C++.

Υπάρχουν και άλλοι clients γραμμένοι σε άλλες γλώσσες προγραμματισμού που χρησιμοποιούνται, αλλά δεν τόσο διαδεδομένοι και ίσως όχι το ίδιο ασφαλείς.

Κάθε χρήστης του δικτύου αναγνωρίζεται από την μοναδική του ταυτότητα, node ID, η οποία έχει μέγεθος 512 bit, δηλαδή 64 bytes. Για να συνδεθεί με το υπόλοιπο δίκτυο πρέπει να πραγματοποιήσει έναν αριθμό από συνδέσεις προς άλλους χρήστες. Οι συνδέσεις αυτές μπορεί να είναι είτε UDP είτε TCP.

Κάθε κόμβος είναι υπεύθυνος για τις εξής τέσσερις δραστηριότητες που πρέπει να πραγματοποιεί:

- Τοπική αντιγραφή της δημόσιας και έμπιστης αλυσίδας.
- Προώθηση συναλλαγών στη δημόσια αλυσίδα.
- Να απαντάει στις ερωτήσεις του δικτύου και κατά συνέπεια των άλλων κόμβων.
- Επαλήθευση όλων των καινούργιων block που έχουν προστεθεί στην αλυσίδα.

2.3. UDP συνδέσεις

Οι UDP συνδέσεις χρησιμοποιούνται μόνο για τη μεταφορά πληροφορίας μέσα στο δίκτυο. Κάθε κόμβος μπορεί να πραγματοποιήσει αμέτρητες UDP συνδέσεις με τον περιορισμό ότι μπορεί να έχει ενεργές ταυτόχρονα μόνο 16 από αυτές.

Υπάρχουν τέσσερις τύποι μηνυμάτων στο δίκτυο.

Ένα μήνυμα *ping,* στο οποίο αποστέλλεται ένα μήνυμα *pong* ως απάντηση.

Ένα μήνυμα *findnode* στο οποίο αποστέλλεται ένα μήνυμα *neighbor* ως απάντηση, το οποίο περιέχει 16 γειτονικούς κόμβους του ανταποκρινόμενου κόμβου. Όλα τα UDP μηνύματα έχουν χρονικές σημάνσεις και έχουν κρυπτογραφηθεί με το ID του αποστολέα. Για περιορισμό των επιθέσεων επανάληψης (replay attacks), όλα τα μηνύματα τα οποία έχουν διαφορά μεγαλύτερη των 20 δευτερολέπτων από την τοπική ώρα του παραλήπτη απορρίπτονται αυτόματα. Για την αποφυγή επιθέσεων από πλαστογραφημένες IP, κάθε μήνυμα *pong* περιέχει και την κατακερματισμένη μορφή του μηνύματος *ping* στο οποίο απαντάει.

2.4. TCP συνδέσεις

Αντίθετα από τις UTP συνδέσεις, όλη η πληροφορία που είναι σχετική με την αλυσίδα, μεταφέρεται μέσω κρυπτογραφημένων και αυθεντικοποιημένων TCP συνδέσεων. Ο μέγιστος

αριθμός συνδέσεων που μπορεί να πραγματοποιήσει ένας κόμβος κάθε δεδομένη στιγμή είναι ίσος με τον αριθμό *maxpeers*, που είναι προεπιλεγμένος στον αριθμό 25.

Μια σύνδεση TCP μπορεί να είναι είτε εξερχόμενη, δηλαδή να την έχει εκκινήσει ο ίδιος ο κόμβος προς το δίκτυο, είτε εισερχόμενη, δηλαδή να έχει δρομολογηθεί από κάποιον άλλον προς εκείνον.

2.5. Το block

Το block στο Ethereum είναι μια συλλογή από σχετικά τμήματα πληροφορίας, το οποίο ονομάζεται κεφαλίδα (block header) και περιέχει τα εξής:

- *parentHash*: Η κατακερματισμένη τιμή μεγέθους 256-bit του προηγούμενου block με όλα τα στοιχεία τα οποία περιλαμβάνει.
- *ommersHash*: Η κατακερματισμένη 256-bit τιμή του ίδιου του block, με όλα τα υπόλοιπα στοιχεία σαν είσοδο.
- *beneficiary*: Η 160-bit διεύθυνση του χρήστη που εισέπραξε όλα τα τέλη (taxes) από την επιτυχή δημιουργία του block.
- *transactionRoot*: Η κατακερματισμένη 256-bit τιμή του αρχικού κόμβου βάση της ψηφιακής δομής του δέντρου, η οποία συμπληρώνεται με κάθε συναλλαγή στη λίστα συναλλαγών.
- *logsBloom*: Το φίλτρο Bloom το οποίο απαρτίζεται από την πληροφορία που περιέχεται σε κάθε καταχώρηση του αρχείου καταγραφής.
- *difficulty*: Η τιμή της δυσκολίας του αντίστοιχου block. Υπολογίζεται από τη δυσκολία του προηγούμενου block και της χρονικής σήμανσης του.
- *number*: Ένας αριθμός ο οποίος ισούται με τον αριθμό των προηγούμενων block που έχουν προστεθεί στην αλυσίδα.
- *gasLimit*: Η τιμή η οποία ισούται με την τρέχουσα τιμή ορίου gas που μπορεί να χρησιμοποιηθεί ως δαπάνη για κάθε block.

- *gasUsed*: Η τιμή η οποία ισούται με την συνολική τιμή της ποσότητας gas που χρησιμοποιήθηκε στις συναλλαγές αυτού του block.
- *timestamp*: Η τιμή η οποία ισούται με τη λογική τιμή εξόδου της συνάρτησης time() σε χρόνο Unix κατά τη δημιουργία του block.
- *extraData*: Μια αυθαίρετου μεγέθους λίστα από byte που περιέχει δεδομένα σχετικά με αυτό το block. Αυτή η λίστα μπορεί να είναι μέχρι 32 byte.
- *mixHash*: Η κατακερματισμένη 256-bit τιμή, που αποδεικνύει σε συνδυασμό με τη *nonce*, ότι μια σημαντική ποσότητα υπολογισμών πραγματοποιήθηκαν κατά τη δημιουργία του block.
- *nonce*: Η κατακερματισμένη 64-bit τιμή που αποδεικνύει σε συνδυασμό με την *mixHash*, ότι μια σημαντική ποσότητα από υπολογισμούς πραγματοποιήθηκαν κατά τη δημιουργία του block.

2.6. Η συναλλαγή

Η συναλλαγή στο Ethereum είναι μια υπογεγραμμένη εντολή με το ιδιωτικό κλειδί του χρήστη που βρίσκεται εκτός του πεδίου εφαρμογής του Ethereum.

Υπάρχουν δύο διαφορετικές κατηγορίες συναλλαγών που μπορούν να πραγματοποιηθούν στην πλατφόρμα του Ethereum. Αυτές είναι οι συναλλαγές που έχουν σαν αποτέλεσμα την αποστολή κάποιου μηνύματος και οι συναλλαγές που καταλήγουν στη δημιουργία ενός καινούργιου λογαριασμού και τη συσχέτιση του κώδικα που τον συνοδεύει.

Και οι δύο κατηγορίες έχουν κάποια κοινά πεδία:

- *Nonce*: Ένας αριθμός που ισούται με τον αριθμό των συναλλαγών που έχει πραγματοποιήσει ο αποστολέας.
- *gasPrice*: Ένας αριθμός που ισούται με την ποσότητα του wei που θα καταβληθεί ανά μονάδα gas, για το συνολικό υπολογιστικό κόστος της συναλλαγής.

- *gasLimit*: Ένας αριθμός που ισούται με τη μέγιστη ποσότητα gas που θα χρησιμοποιηθεί κατά την εκτέλεση της συναλλαγής. Πληρώνεται προκαταβολικά, πρωτού να γίνει κάποιος υπολογισμός και δεν μπορεί να αυξηθεί στη συνέχεια.

- *to*: Η διεύθυνση μεγέθους 160-bit του παραλήπτη του μηνύματος, ή της δημιουργίας του contract.

- *value*: Μια τιμή που ισούται με την ποσότητα wei η οποία θα μεταφερθεί από τον αποστολέα προς τον παραλήπτη ή στην περίπτωση που πρόκειται για τη λειτουργία ενός contract, ως επιχορήγηση για τον πρόσφατο δημιουργημένο λογαριασμό.

- *v, r, s*: Τιμές που αντιστοιχούν στην υπογραφή της συναλλαγής και θα χρησιμοποιηθούν για προσδιοριστεί ο αποστολέας της συναλλαγής.

Πέρα από αυτά τα πεδία όμως υπάρχουν και επιπρόσθετα, ανάλογα με τον τύπο της συναλλαγής. Πιο συγκεκριμένα, η συναλλαγή που δημιουργεί ένα contract περιλαμβάνει και το πεδίο init.

- *init*: Μια λίστα τύπου byte αορίστου μεγέθους, που καθορίζει τον EVM κώδικα για την εκκίνηση της δημιουργίας του λογαριασμού.

Init είναι ένα τμήμα του EVM κώδικα το οποίο επιστρέφει το κύριο μέρος κώδικα του contract και ένα δεύτερο τμήμα κώδικα το οποίο εκτελείται κάθε φορά που ο λογαριασμός αυτός λαμβάνει κάποιο μήνυμα. Το τμήμα init, εκτελείται μόνο μια φορά και αυτό κατά τη διάρκεια της δημιουργίας του λογαριασμού και στη συνέχεια διαγράφεται άμεσα.

Αντίθετα, όταν η συναλλαγή περιέχει κάποιο μήνυμα, εμπεριέχεται το πεδίο data.

- *data*: Μια λίστα τύπου byte αορίστου μεγέθους που καθορίζει τα δεδομένα που θα χρησιμοποιηθούν σαν είσοδος.

2.6.1. Κοστολόγηση

Για να μην υπάρξει κάποια κατάχρηση των πόρων του δικτύου όλος ο προγραμματιζόμενος υπολογισμός υπόκειται σε τέλη. Έτσι, κάθε δοσμένη ενέργεια που θα χρησιμοποιήσει την επεξεργαστική ισχύ του δικτύου έχει και ένα καθολικά συμφωνημένο κόστος, το οποίο αναφέρεται ως gas.

Κάθε συναλλαγή έχει μια συγκεκριμένη ποσότητα που πρέπει να πληρωθεί ως τέλος, και ονομάζεται *gasLimit*. Αυτή η ποσότητα καταβάλλεται έμμεσα από τον αποστολέα.

Η κοστολόγηση του gas γίνεται σύμφωνα με το *gasPrice*, το οποίο επίσης είναι καθορισμένο στη συναλλαγή. Αν ο αποστολέας δεν μπορεί να καταβάλει το απαιτούμενο ποσό τότε η συναλλαγή ακυρώνεται. Η διαφορά μεταξύ αυτών των δύο ποσοτήτων βρίσκεται στην περίπτωση που ο χρήστης πληρώσει μια ποσότητα wei ως gasLimit το οποίο υπερβαίνει το gas που χρειάζεται η συγκεκριμένη συναλλαγή, τότε η διαφορά θα του επιστραφεί άμεσα. Η έννοια του gas υφίσταται αποκλειστικά εντός των ορίων της εκτέλεσης συναλλαγών.

2.6.2. Επισκόπηση τελών (Fees Overview)

Τέλη στην πλατφόρμα του Ethereum χρεώνονται σε τρεις διακριτές περιστάσεις, και η πληρωμή τους είναι πάντα προαπαιτούμενη για την εκτέλεση της συναλλαγής.

- Το πρώτο και πιο κοινό τέλος που τίθεται να πληρώσουν οι χρήστες της πλατφόρμας είναι το εγγενές κόστος εκτέλεσης της διεργασίας.
- Δεύτερο τέλος μπορεί να εμφανιστεί κατά τη διάρκεια της δημιουργίας ενός δευτερεύοντος μηνύματος ή δημιουργίας contract.
- Τέλος, τρίτο τέλος μπορεί να τοποθετηθεί στη συναλλαγή όταν πρόκειται να χρησιμοποιήσει επιπλέον χώρο της μνήμης του EVM.

2.6.3. Περιβάλλον εκτέλεσης (Execution environment)

Υπάρχουν αρκετά κομμάτια σημαντικής πληροφορίας που χρησιμοποιούνται στο περιβάλλον εκτέλεσης πέρα από την κατάσταση του συστήματος και το υπόλοιπο gas (g) κάθε υπολογισμού που γίνεται. Οι πληροφορίες αυτές περιέχονται σε μια πλειάδα (tuple) και είναι οι εξής:

- Η διεύθυνση του λογαριασμού που ανήκει ο κώδικας που εκτελείται.
- Η διεύθυνση του αποστολέα της συναλλαγής από όπου προήλθε η εκτέλεση.
- Η τιμή του gas μέσα στη συναλλαγή.
- Η λίστα των bytes που θα χρησιμοποιηθούν ως δεδομένα εισόδου.
- Η διεύθυνση του λογαριασμού που προκάλεσε την εκτέλεση του κώδικα.
- Η τιμή, σε wei, που αποστάλθηκε σε αυτόν το λογαριασμό ως μέρος της ίδιας διαδικασίας
- Η λίστα των byte, που είναι ο κώδικας μηχανής που θα εκτελεστεί.
- Το block header του παρόντος block.
- Το βάθος του παρόντος μηνύματος που αποστάλθηκε ή του contract που δημιουργήθηκε.

2.7. Εξόρυξη (Mining) – Proof of Work (PoW)

Η εξόρυξη ως απόδειξη της εργασίας (PoW) υπάρχει ως ασφαλής κρυπτογραφικός ορισμός που αποδεικνύει πέρα από κάθε λογική αμφιβολία ότι μια σεβαστή ποσότητα υπολογισμών έχουν πραγματοποιηθεί για τον καθορισμό του διακριτικού (token).

Η εξόρυξη αναπτύχθηκε ώστε να αυξήσει την ασφάλεια στην τεχνολογία blockchain και πραγματικά δίνει νόημα στον ορισμό της δυσκολίας. Ωστόσο, επειδή η δημιουργία κάθε

καινούργιου block συνδέεται και με μια ανταμοιβή, όλη η διαδικασία δε λειτουργεί μόνο ως μια μέθοδος εξασφάλισης εμπιστοσύνης αλλά και ως κατανεμητής πλούτου. Για αυτούς τους λόγους υπάρχουν δύο σημαντικοί στόχοι που πρέπει να επιτύχει. Πρώτον, η εξόρυξη θα πρέπει να είναι προσβάσιμη από όσο το δυνατόν περισσότερους χρήστες και δεύτερον δεν πρέπει να είναι εφικτό σε κάποιον χρήστη να έχει υπερβολικά γραμμικά αυξανόμενα κέρδη.

2.7.1. Ethash

Ο αλγόριθμος που χρησιμοποιείται στο Ethereum ονομάζεται Ethash. Είναι η τελευταία έκδοση του αλγορίθμου "Dagger-Hashimoto", ο οποίος εισήχθη από τον Buterin και τον Dryja. Ωστόσο, δε διατηρεί πλέον το αρχικό του όνομα καθώς έχει αλλάξει δραματικά.

Η λογική που ακολουθεί αυτός ο αλγόριθμος έχει ως εξής:

- Υπάρχει ένας "σπόρος" (seed) ο οποίος μπορεί να υπολογιστεί για κάθε block σαρώνοντας τον block header του μέχρι ένα σημείο.
- Από τον "σπόρο", ένας χρήστης μπορεί να υπολογίσει μια ψευδοτυχαία cache μεγέθους 16MB. Η cache αποθηκεύεται στους *ελαφρείς* χρήστες.
- Από την cache, μπορεί να υπολογιστεί ένα dataset το οποίο σήμερα έχει μέγεθος 4 GB. Κάθε αντικείμενο του dataset εξαρτάται από ένα μικρό αριθμό αντικειμένων που βρίσκεται στην cache. Πλήρεις χρήστες και miners αποθηκεύουν αυτό το dataset, το οποίο μεγαλώνει κατά τη διάρκεια του χρόνου.
- Η διαδικασία της εξόρυξης περιλαμβάνει την επιλογή τυχαίων τμημάτων του dataset και τον κατακερματισμό του ομαδοποιημένου συνόλου αυτών. Η επαλήθευση του αποτελέσματος μπορεί να γίνει με περιορισμένη χρήση μνήμης, χρησιμοποιώντας την cache για την ανάπλαση των συγκεκριμένων κομματιών που χρειάζεται από το dataset.

Η διαδικασία επίλυσης του δύσκολου αυτού μαθηματικού προβλήματος διαρκεί συνήθως μεταξύ 12 και 15 δευτερολέπτων. Ο αλγόριθμος μεταβάλει τη δυσκολία του αυτόματα στην

περίπτωση που ο χρόνος μεταξύ της δημιουργίας των δύο τελευταίων block είναι μεγαλύτερος ή μικρότερος από αυτό το χρονικό όριο.

2.8. Smart Contracts

Όπως ειπώθηκε και στην εισαγωγή του κεφαλαίου, τα smart contracts είναι προγράμματα τα οποία δημοσιεύονται στην πλατφόρμα του Ethereum και εκτελούνται με την βοήθεια του Ethereum Virtual Machine, EVM. Τα προγράμματα αυτά γράφονται με την γλώσσα προγραμματισμού Solidity και μπορούν να χρησιμοποιηθούν είτε ως αυτόνομα προγράμματα, είτε ως βοηθητικά για άλλα, π.χ. ως βιβλιοθήκες. Από τη στιγμή που το πρόγραμμα θα δημοσιευθεί στην πλατφόρμα, μετατρέπεται σε κώδικα μηχανής και αποκτά μια δημόσια διεύθυνση που το αντιπροσωπεύει και του επιτρέπει την αλληλεπίδραση με τους υπόλοιπους χρήστες και προγράμματα.

2.9. Ethereum Virtual Machine

Το EVM είναι βασισμένο στην αρχιτεκτονική στοίβας. Το μέγεθος κάθε λέξης στη μηχανή αυτή είναι 256-bit. Αυτή η επιλογή έγινε για διευκόλυνση αλλά και για λόγους συμβατότητας με τον αλγόριθμο κατακερματισμού που χρησιμοποιείται, τον Keccak-256, καθώς και για τον υπολογισμό των ελλειπτικών καμπυλών. Το μοντέλο μνήμης (memory model) είναι μια απλή λίστα byte μεγέθους 256-bit. Η στοίβα έχει μέγιστο όριο 1024. Επίσης, το EVM υποστηρίζει το μοντέλο αποθήκευσης (storage model) με τη διαφορά ότι αντί για μια λίστα byte έχουμε μια λίστα word. Η διαφορά μεταξύ αυτών των δύο μοντέλων, είναι ότι στο πρώτο η λίστα είναι ευμετάβλητη ενώ στο δεύτερο δεν είναι και η λίστα διατηρείται ως μέρος της κατάστασης του συστήματος. Όλες οι θέσεις τόσο στο μοντέλο μνήμης όσο και στο μοντέλο αποθήκευσης είναι ορισμένες αρχικά ως μηδέν. Το πρόγραμμα δεν αποθηκεύεται στη μνήμη ούτε στον αποθηκευτικό χώρο, αλλά σε μια ψηφιακή μνήμη ROM η οποία αλληλεπιδρά αποκλειστικά μέσω μιας εξειδικευμένης εντολής. Το EVM μπορεί να εκτελέσει κάποιον

κώδικα ανάγκης για διαφόρους λόγους που συμπεριλαμβάνουν την υποχείλιση και τις λανθασμένες εντολές.

2.10. Η τριάδα της ασφάλειας

Βασική τριάδα της ασφάλειας είναι η ακεραιότητα, η διαθεσιμότητα και η εμπιστευτικότητα της πληροφορίας. Το blockchain μέσω της κρυπτογραφίας διασφαλίζει ότι η πληροφορία που αποθηκεύεται στην αλυσίδα δεν μπορεί να τροποποιηθεί. Η διαθεσιμότητα της πλατφόρμας εξαρτάται από την σωστή λειτουργία των κόμβων, οι οποίοι απαντάνε στα ερωτήματα του δικτύου και προωθούν μηνύματα προς την αλυσίδα. Τέλος, η εμπιστευτικότητα που παρέχει η κάθε πλατφόρμα εξαρτάται από την ποσότητα της κρίσιμης πληροφορίας που αφήνει προσβάσιμη προς όλους τους χρήστες της.

2.10.1. Ακεραιότητα

Κάθε block της αλυσίδας περνάει από μια συνάρτηση κατακερματισμού και το αποτέλεσμα αυτής τοποθετείται μέσα στο επόμενο block που θα τοποθετηθεί στην αλυσίδα. Όταν το περιεχόμενο κάποιου block αλλάξει τότε θα αλλάξει και το αποτέλεσμα το οποίο θα παράγει η συνάρτηση κατακερματισμού. Αν κάποιος κακόβουλος χρήστης προσπαθήσει να τροποποιήσει κάποιο προηγούμενο block το οποίο υπάρχει ήδη στην αλυσίδα, η συνάρτηση κατακερματισμού θα βγάλει άλλο αποτέλεσμα και τότε το επόμενο block, το οποίο περιέχει το προηγούμενο κατακερματισμένο περιεχόμενο του, θα το απορρίψει. Από αυτό συμπεραίνουμε ότι μέσα στην αλυσίδα υπάρχει συνεχής έλεγχος ακεραιότητας των δεδομένων.

2.10.2. Διαθεσιμότητα

Επειδή το blockchain είναι μια τεχνολογία που βασίζεται πάνω στην αποκεντρωμένη λειτουργία και όχι σε έναν διακομιστή ο οποίος ανήκει σε μια επιχείρηση και σε ένα μηχάνημα, κάθε κόμβος του δικτύου έχει στην κατοχή του όλα τα κατάλληλα εργαλεία τα οποία μπορούν να κάνουν λειτουργική την πλατφόρμα. Αυτό σημαίνει ότι για να πάψει η πλατφόρμα να είναι διαθέσιμη, πρέπει να σταματήσει η λειτουργία όλων των κόμβων του δικτύου και κανείς να μην έχει επαφή με κανέναν.

2.10.3. Εμπιστευτικότητα

Κάθε συναλλαγή που πραγματοποιείται στο blockchain είναι δημόσια προσβάσιμη από όλους τους χρήστες της. Αυτό σημαίνει ότι εαν ο χρήστης Μπομπ πραγματοποιήσει μια συναλλαγή με την χρήστη Άλις, τη στιγμή που η συναλλαγή θα τοποθετηθεί στην δημόσια αλυσίδα, θα μπορεί να εξεταστεί από όλους τους χρήστες της, οπότε όλοι μπορούν να ξέρουν ότι έγινε μια συναλλαγή μεταξύ αυτών των δύο καθώς και την ποσότητα Ether που μεταφέρθηκε από τον έναν λογαριασμό στον άλλον. Επίσης, υπάρχει η εντύπωση στο ευρύ κοινό ότι το blockchain είναι ανώνυμο. Αυτό είναι μερικώς αλήθεια. Για να χρησιμοποιήσει κάποιος την πλατφόρμα πρέπει να έχει μια ποσότητα κρυπτονομίσματος στην κατοχή του. Για να αποκτήσει αυτή την ποσότητα σημαίνει ότι διαθέτει ένα πορτοφόλι συνδεδεμένο με τον λογαριασμό του. Αυτό το πορτοφόλι μπορεί να είναι είτε ψηφιακό είτε εξωτερικός εξοπλισμός συνδεδεμένος στον υπολογιστή του χρήστη. Στην συνέχεια πρέπει να αποκτήσει μια ποσότητα κρυπτονομίσματος η οποία μπορεί να κερδηθεί από το δίκτυο λύνοντας κάποιο δύσκολο μαθηματικό πρόβλημα, ή να την αγοράσει από κάποια άλλη πλατφόρμα συναλλαγής. Αυτές οι πλατφόρμες τις περισσότερες φορές ζητάνε τα πλήρη στοιχεία του χρήστη και κάποιο λογαριασμό τραπέζης ή τον αριθμό κάρτας για να ολοκληρωθεί η αγορά. Σε αυτή την περίπτωση ο χρήστης της πλατφόρμας, ενώ μέσα στην πλατφόρμα φαίνεται ως ανώνυμος αφού αντιπροσωπεύεται από τη μοναδική του διεύθυνση, μπορεί πάντα να εντοπιστεί και αναγνωριστεί. Αν ο χρήστης ανήκει στην πρώτη κατηγορία είναι πρακτικά ανώνυμος αλλά πλέον είναι αρκετά δύσκολο.

3. Ευπάθειες – επιθέσεις

3.1. Γενικές επιθέσεις

Στις παρακάτω παραγράφους θα αναλύσουμε τρεις από τις πιο επικίνδυνες επιθέσεις που μπορούν να πραγματοποιηθούν σε οποιαδήποτε πλατφόρμα χρησιμοποιεί την τεχνολογία blockchain. Θα αναλύσουμε που εμφανίζεται η ευπάθεια κάθε φορά και πως μπορεί να αντιμετωπιστεί.

3.1.1. Διπλή δαπάνη (double spend)

3.1.1.α. Περιγραφή

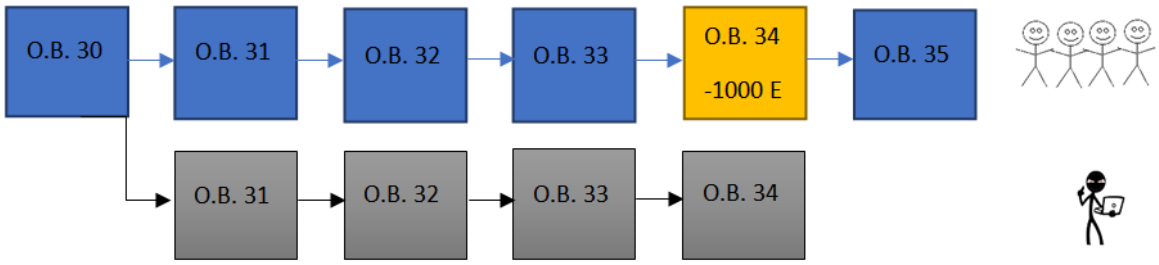
Η επίθεση διπλής δαπάνης είναι μια επίθεση δύο σταδίων που μπορεί να πραγματοποιηθεί στην πλειοψηφία των blockchain πλατφορμών, κατά συνέπεια και στην πλατφόρμα του Ethereum. Ο κακόβουλος χρήστης που θα πραγματοποιήσει αυτή την επίθεση έχει σαν τελικό στόχο την αντιστροφή μίας ή παραπάνω, συναλλαγής που έχει ήδη πραγματοποιήσει.

3.1.1.β. Η ευπάθεια

I. Αόρατη εξόρυξη (stealth mining)

Το πρώτο στάδιο αυτής της επίθεσης ονομάζεται αόρατη εξόρυξη. Όπως αναλύθηκε στην παράγραφο της λειτουργίας του Ethereum, όταν κάποιος χρήστης λύσει το δύσκολο μαθηματικό πρόβλημα το οποίο είναι προαπαιτούμενο για την δημιουργία ενός block, το δημοσιοποιεί και στους άλλους χρήστες ώστε να το επιβεβαιώσουν ομόφωνα. Αυτό όμως δεν συμβαίνει πάντα. Για να πραγματοποιηθεί η επίθεση της αόρατης εξόρυξης ο κακόβουλος χρήστης πρέπει πρώτα να δημιουργήσει έναν ιδιωτικό κλώνο της δημόσιας ενεργής αλυσίδας. Όσο οι χρήστες της πλατφόρμας προσπαθούν να λύσουν και να προσθέσουν καινούργια block

σε αυτήν, ο επιτιθέμενος προσπαθεί μόνος του να λύσει το κάθε πρόβλημα αλλά δεν δημοσιεύει τις λύσεις που βρίσκει στο υπόλοιπο δίκτυο.



Σχ. 1. Το στάδιο stealth mining της ευπάθειας Double Spend.

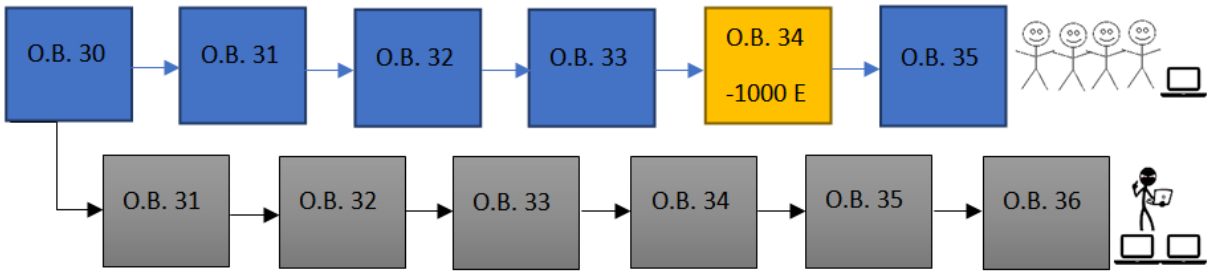
Το block 34, που έχει κίτρινο χρώμα στην δημόσια αλυσίδα, είναι το block το οποίο περιέχει και την συναλλαγή του κακόβουλου χρήστη, σε αντίθεση με το γκρι block 34 το οποίο δεν την περιλαμβάνει.

Ο χρήστης που πραγματοποιεί την επίθεση, συνεχίζει να λαμβάνει blocks από τη δημόσια αλυσίδα και να τα επιβεβαιώνει μόνος του. Αν σε αυτό το σημείο προσπαθήσει να δημοσιεύσει την δική του αλυσίδα, το σύστημα θα την απορρίψει διότι η τεχνολογία blockchain θεωρεί τη μακρύτερη αλυσίδα ως την αυθεντική και όλοι οι λογαριασμοί μαζί με τα ψηφιακά πορτοφόλια που χρησιμοποιούν, βρίσκονται σε συγχρονισμό με αυτή. Ο επιτιθέμενος δεν έχει τη μεγαλύτερη αλυσίδα οπότε η δημοσίευση δεν θα πραγματοποιηθεί.

Για να προσπεραστεί αυτό το πρόβλημα ο επιτιθέμενος πρέπει να περάσει στο δεύτερο στάδιο που είναι η επίθεση 51%.

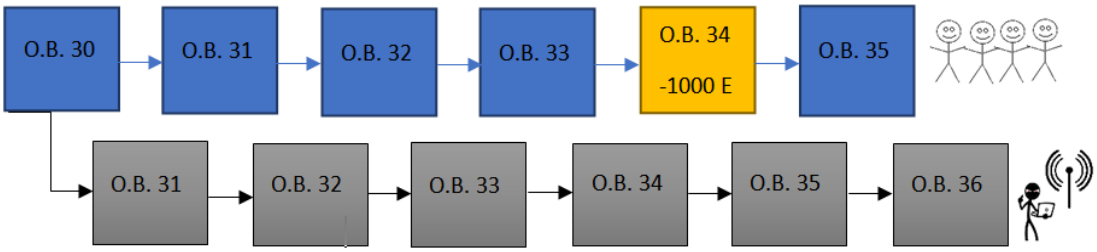
II. 51% Επίθεση

Ένας από τους μεγαλύτερους κινδύνους που αντιμετωπίζουν όλες οι πλατφόρμες αυτής της τεχνολογίας είναι οι επιθέσεις 51%. Επειδή όλη η τεχνολογία έχει την βάση της στην δημοκρατική διακυβέρνηση, δηλαδή στην πλειοψηφία, αν το μεγαλύτερο ποσοστό χρηστών οδηγηθεί από την αυθεντική αλυσίδα σε κάποια άλλη η οποία περιέχει διαφορετικές συναλλαγές, τότε οι συναλλαγές της πρώτης αλυσίδας θα αναιρεθούν και οι χρηματικές ποσότητες που διακινήθηκαν θα επιστραφούν στους αρχικούς κατόχους.



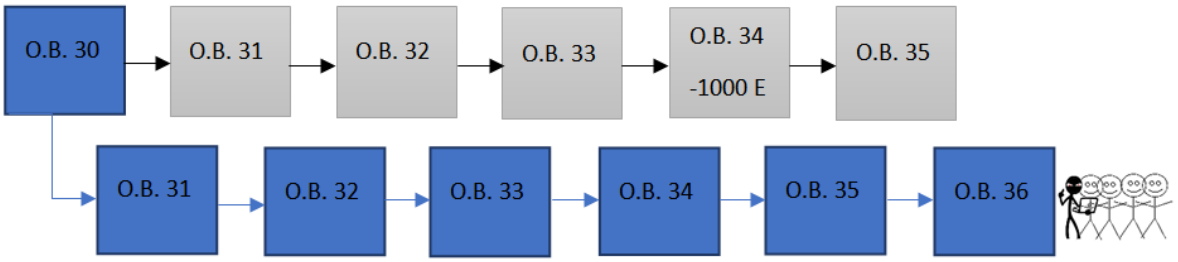
Σχ. 2.α.. Το στάδιο 51% επίθεσης της ευπάθειας Double Spend.

Για να πετύχει τον σκοπό του ο κακόβουλος χρήστης μπορεί είτε να αγοράσει εξοπλισμό ο οποίος θα έχει μεγαλύτερη επεξεργαστική ισχύ από το υπόλοιπο δίκτυο υπολογιστών, είτε να βρει κάποια ευπάθεια στο πρωτόκολλο δημιουργίας των block και να επιταχύνει την προσθήκη στην δική του προσωπική αλυσίδα. Όταν η δική του αλυσίδα θα είναι μεγαλύτερη από την αυθεντική αλυσίδα τότε θα προσπαθήσει να τη δημοσιεύσει.



Σχ. 2.β. Το στάδιο 51% επίθεσης της ευπάθειας Double Spend.

Πλέον η ιδιωτική αλυσίδα του επιτιθέμενου είναι μεγαλύτερη από τη δημόσια και μπορεί να δημοσιευθεί χωρίς να αποτύχει. Αυτό σημαίνει ότι όλοι οι χρήστες της πλατφόρμας θα στραφούν προς την αλυσίδα του κακόβουλου χρήστη και όλα τα πορτοφόλια των χρηστών και οι συναλλαγές θα ανανεωθούν με βάση αυτήν. Η αλυσίδα του κακόβουλου χρήστη είναι πλέον η αλυσίδα εμπιστοσύνης και όλες οι συναλλαγές που δεν υπάρχουν σε αυτή θα αναιρεθούν άμεσα. Οπότε ο κακόβουλος χρήστης πέτυχε τον σκοπό του. Η συναλλαγή που είχε πραγματοποιήσει προηγουμένως δεν υπάρχει πλέον στην αλυσίδα και η ποσότητα η οποία σπατάλησε βρίσκεται στην κατοχή του για να την ξοδέψει ξανά.



Σχ. 2.γ. Το στάδιο 51% επίθεσης της ευπάθειας Double Spend.

3.1.1.γ. Αντίμετρα

Το καλύτερο αντίμετρο για μια τέτοιου είδους επίθεση, είναι το ίδιο το κόστος της επίθεσης και τα περιορισμένα κίνητρα που δίνει η πλατφόρμα στον επιτιθέμενο. Καθοριστικό ρόλο έχει και η αντιστοιχία του κάθε κρυπτονομίσματος στην αγορά η οποία όσο υψηλότερη είναι τόσο πιο ακριβή κάνει την επίθεση. Επίσης, σε μια επίθεση όπου είναι εμφανής και μπορεί να επηρεάσει δραματικά την λειτουργία της πλατφόρμας, υπάρχει η δυνατότητα «επανεκκίνηση της» από μια δοσμένη χρονική στιγμή και μετά, επαναφέροντας την στην κατάσταση που βρισκόταν πριν την επίθεση και αντιστρέφοντας όλες τις συναλλαγές.

3.1.1.δ. Πραγματικό περιστατικό

Το Απρίλιο του 2018 στην πλατφόρμα του Verge (XVG) πραγματοποιήθηκε μια τέτοιου είδους επίθεση με αποτέλεσμα να καταστρέψει την οικονομία της πλατφόρμας και να αναγκάσει τους διαχειριστές της να την επαναφέρουν σε κάποια προηγούμενη κατάσταση. Όλες οι συναλλαγές που είχαν πραγματοποιηθεί από το σημείο μηδέν της επίθεσης και μετά, αντιστράφηκαν και οι ποσότητες γύρισαν στους νόμιμους κάτοχους τους.

Οι επιτιθέμενοι εκμεταλλεύτηκαν μια σειρά από ευπάθειες που υπήρχαν στον κώδικα της πλατφόρμας και κατάφεραν να επαναχρησιμοποιούν τον ίδιον αλγόριθμο για να δημιουργούν και να επαληθεύουν νέα block μέσα σε ένα δευτερόλεπτο. Μέσα σε ένα μικρό χρονικό διάστημα είχαν δημιουργήσει μια αλυσίδα που ήταν μεγαλύτερη από την δημόσια και μπόρεσαν εύκολα να την αντικαταστήσουν.

3.1.2. Επίθεση έκλειψης (Eclipse Attack)

3.1.2.α. Περιγραφή

Ο σκοπός αυτής της επίθεσης είναι να οδηγήσει το θύμα σε μια απομόνωση από το δίκτυο και τους υπόλοιπους χρήστες της πλατφόρμας χωρίς εκείνο να το γνωρίζει. Στη συνέχεια ο επιτιθέμενος μπορεί να χρησιμοποιήσει τη συνολική επεξεργαστική ισχύ του θύματος ώστε να πραγματοποιήσει πιο περίπλοκες επιθέσεις, όπως η επίθεση 51% που εξετάσαμε προηγουμένως.

Η επίθεση αυτή πραγματοποιήθηκε από το πανεπιστήμιο του Pittsburgh στις αρχές του χρόνου στην πλατφόρμα του Ethereum, με μεγάλο ποσοστό επιτυχίας. Ωστόσο, μπορεί να πραγματοποιηθεί και σε παρόμοια δίκτυα χρησιμοποιώντας ίδιες ή παρόμοιες τεχνικές.

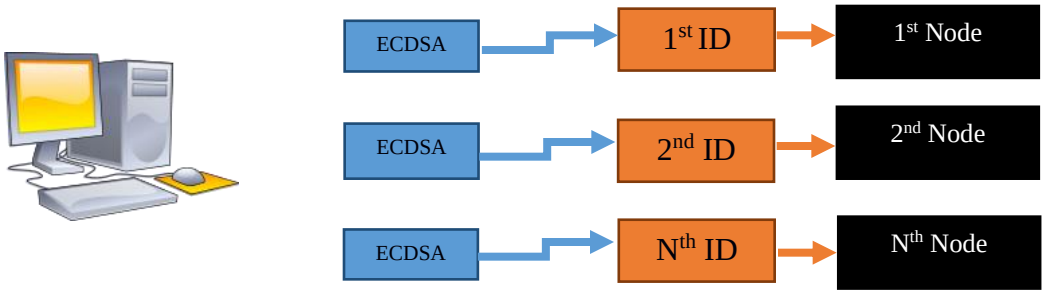
3.1.2.β. Η ευπάθεια

I. ECDSA

Στη θεωρία, μια τέτοια επίθεση στην πλατφόρμα του Ethereum θα έπρεπε να είναι αρκετά δύσκολη και να απαιτεί μεγάλη χρηματική επένδυση του επιτιθέμενου σε τεχνικό εξοπλισμό. Στην παράγραφο της λειτουργίας του κόμβου, αναφέρθηκε ότι κάθε κόμβος κατά την εκκίνηση του πραγματοποιεί μια πληθώρα συνδέσεων, 13 στον αριθμό, προς άλλους υπολογιστές και κατά συνέπεια στην αλυσίδα που θεωρείται έμπιστη. Στην πλατφόρμα του Bitcoin κάθε κόμβος πραγματοποιεί οκτώ τέτοιες συνδέσεις κατά την εκκίνηση του και για να πραγματοποιηθεί μια επίθεση έκλειψης θα έπρεπε να υλοποιηθεί από εκατοντάδες μηχανήματα με διαφορετικές IP. Αναλογικά, για να πραγματοποιηθεί μια τέτοια επίθεση στο Ethereum, ο αριθμός θα έπρεπε να είναι αρκετά μεγαλύτερος, ίσως και να έφτανε κάποιες χιλιάδες κόμβους. Το πανεπιστήμιο του Pittsburgh πραγματοποίησε αυτή την επίθεση με το χαμηλότερο δυνατό κόστος, χρησιμοποιώντας δύο υπολογιστές οι οποίοι είχαν διαφορετική IP.

Από αυτό γίνεται κατανοητό ότι η επίθεση έκλειψης που πραγματοποιήθηκε στις αρχές του 2018 εκμεταλλεύτηκε κάποιο άλλο στοιχείο της πλατφόρμας. Το στοιχείο αυτό είναι η

δημιουργία του δημόσιου κλειδιού χρησιμοποιώντας τον αλγόριθμο Ελλειπτικής Καμπύλης Ψηφιακής Υπογραφής (ECDSA).



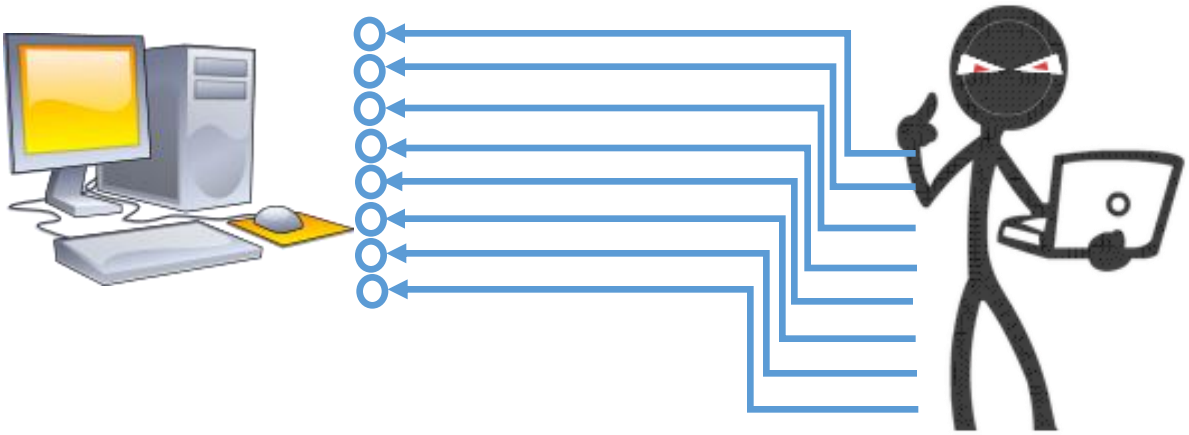
Σχ. 3.α. Ο κακόβουλος χρήστης εκτελεί n φορές τον αλγόριθμο και αποκτάει n αναγνωριστικά

Αυτό είναι το πρώτο στάδιο της επίθεσης. Η ερευνητική ομάδα του πανεπιστημίου πραγματοποίησε την επίθεση έκλειψης με δύο διαφορετικούς τρόπους.

Η πρώτη επίθεση πραγματοποιείται με την μονοπώληση των συνδέσεων του θύματος. Για να επιτευχθεί αυτό ο επιτιθέμενος πρέπει να εκμεταλλευτεί διάφορα στοιχεία της πλατφόρμας. Η πρώτη ευπάθεια έγκειται στο γεγονός ότι όλες οι συνδέσεις που θα πραγματοποιηθούν με την είσοδο του χρήστη στην πλατφόρμα μπορεί να είναι εισερχόμενες, δηλαδή να έχουν ξεκινήσει από άλλους χρήστες. Η δεύτερη ευπάθεια παρουσιάζεται μετά την επανεκκίνηση του κόμβου και με απουσία εισερχόμενων ή εξερχόμενων συνδέσεων. Τρίτη υφιστάμενη ευπάθεια κατά την πραγματοποίηση της επίθεσης είναι το γεγονός ότι μετά την επανεκκίνηση το μηχάνημα του χρήστη ακούει σε εισερχόμενες TCP και UDP συνδέσεις προτού προλάβει να εκκινήσει τις εξερχόμενες. Λαμβάνοντας όλα αυτά υπόψιν, ο επιτιθέμενος κατασκευάζει ένα μεγάλο αριθμό από ID node στον υπολογιστή του, με τον τρόπο που περιεγράφηκε προηγουμένως, και περιμένει να κάνει επανεκκίνηση ο στόχος. Όταν η επανεκκίνηση ολοκληρωθεί ο επιτιθέμενος ξεκινάει εξερχόμενες συνδέσεις από όλα τους κόμβους που έχει στην κατοχή του με σκοπό να καταληφθούν όλες οι συνδέσεις του θύματος πριν να προλάβει να πραγματοποιήσει κάποια.

Η ευπάθεια στη συγκεκριμένη τεχνική επίθεσης παρουσιάζεται κατά την επανεκκίνηση του θύματος, η οποία μπορεί να γίνει για αρκετούς λόγους. Μερικοί από αυτούς είναι πιθανή διακοπή ρεύματος, απώλεια απαιτούμενης ενέργειας λειτουργίας, αναβάθμιση λογισμικού ή ακόμα και επίθεση στο λειτουργικό σύστημα του στόχου. Επίσης, άλλοι πιθανοί λόγοι είναι η

αποστολή «πακέτου θανάτου» από τον επιτιθέμενο το οποίο μπορεί να σταματήσει αιφνίδια τη λειτουργία του λειτουργικού συστήματος.



Σχ. 3.β. Ο κακόβουλος χρήστης έχει κλείσει όλες τις συνδέσεις του θύματος

II. Εκτεταμένη επίθεση

Η δεύτερη επίθεση έκλειψης απαιτεί δύο υπολογιστές και εκμεταλλεύεται τη λίστα που έχει ο κάθε χρήστης στην κατοχή του και αποθηκεύει τους κόμβους οι οποίοι έχουν συνδεθεί μαζί του στο παρελθόν και εμπιστεύεται.

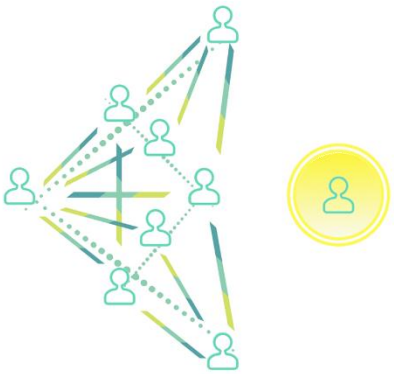
Πρώτο βήμα σε αυτή την επίθεση είναι η δημιουργία μερικών εκατοντάδων node ID σε κάθε υπολογιστή του επιτιθέμενου με τον τρόπο που ήδη περιγράψαμε. Για να γίνει πιο κατανοητό το μέγεθος ευκολίας της δημιουργίας αυτών των ταυτοτήτων δίνεται ότι για 272 IDs ο μέσος χρόνος επεξεργασίας που απαιτείται είναι περίπου 15 λεπτά.

Από την στιγμή που ο επιτιθέμενος έχει κατασκευάσει αυτές τις 272 ταυτότητες διαφορετικών κόμβων, μπορεί να προχωρήσει στο επόμενο βήμα της επίθεσης που είναι να τα εισάγει στη λίστα του θύματος. Για να επιτευχθεί αυτό εκκινείτε μια λειτουργία *ping* από κάθε κόμβο που έχει στην κατοχή του. Στη συνέχεια, περιμένει να λάβει την απάντηση *pong* του θύματος, ώστε να δημιουργηθεί ο δεσμός. Για να διατηρηθεί αυτή η σχέση μεταξύ θύματος και επιτιθέμενου, η διαδικασία αυτή επαναλαμβάνεται κάθε 24 ώρες. Στο ενδιάμεσο χρονικό

διάστημα, το μηχάνημα του επιτιθέμενου πρέπει να απαντάει σε όλα τα *ping* και *findnode* αιτήματα του θύματος.

Το τρίτο βήμα της επίθεσης είναι η επανεκκίνηση του εξοπλισμού του θύματος. Στην αρχή, ακριβώς μετά την επανεκκίνηση του ξεκινάει η διαδικασία *ping* για κάθε ένα από τα IDs που έχει στην κατοχή του. Σε αυτό το σημείο η προσωρινή λίστα των κόμβων του θύματος είναι άδεια, γεγονός που αποτελεί σημαντικό παράγοντα για την υλοποίηση της επίθεσης. Η διαδικασία του *ping* δημιουργεί έναν ταχύτατο δεσμό μεταξύ του θύματος και του επιτιθέμενου, καθώς το θύμα προσθέτει όλους τους κόμβους από τους οποία έλαβε *ping* κατευθείαν στη λίστα του χωρίς εκείνος να τα κάνει *ping*. Δηλαδή ο επιτιθέμενος εκμεταλλεύεται το γεγονός ότι με την εκκίνηση του, ο κόμβος ανταποκρίνεται σε UDP συνδέσεις. Έτσι, η αναζήτηση των κόμβων και οι εξερχόμενες συνδέσεις που θα πραγματοποιήσει το θύμα στη συνέχεια, πιθανότατα θα γίνουν προς ID που βρίσκονται κάτω από την επιρροή του επιτιθέμενου. Τέλος, με τη χρήση ενός δεύτερου υπολογιστή, η ομάδα του πανεπιστημίου του Pittsburgh κατέλαβε όλες τις εισερχόμενες συνδέσεις που θα μπορούσε να λάβει ο υπολογιστής του θύματος, άρα το θύμα σε αυτό το σημείο βρίσκεται σε έκλειψη.

Σχ. 3.γ. Ο κακόβουλος χρήστης συνδέεται στο θύμα, ενώ το θύμα συνδέεται στον κακόβουλο χρήστη.



Σχ. 3.δ. Η τελική εικόνα του δικτύου με το θύμα απομονωμένο.

3.1.2.γ. Αντίμετρα

Η ίδια ομάδα η οποία πραγματοποίησε τις παραπάνω επιθέσεις προτείνει έναν αριθμό αντιμέτρων που θα σταματούσαν την πληθώρα αυτών.

Ένα αντίμετρο στην πρώτη εκδοχή της επίθεσης έκλειψης είναι να τεθεί περιορισμός στις εισερχόμενες ενεργές συνδέσεις κάθε κόμβου. Αυτό θα έχει σαν αποτέλεσμα να υπάρχει μια ανάμιξη εισερχόμενων και εξερχόμενων συνδέσεων και άρα λιγότερες πιθανότητες να συμβεί το περιστατικό που μόλις περιεγράφηκε. Από την έκδοση 1.8 του geth και έπειτα το αντίμετρο αυτό έχει εφαρμοστεί.

Ένα δεύτερο αντίμετρο το οποίο προτείνεται είναι να υπάρχει μια συσχέτιση ένα-προς-ένα μεταξύ κάθε node ID και IP. Με αυτό τον τρόπο δεν μπορεί ο κακόβουλος χρήστης να δημιουργεί πολλαπλά node IDs με την χρήση ενός υπολογιστή και μιας IP. Ένας τρόπος για να υλοποιηθεί αυτό είναι η εισαγωγή μιας ακόμα δομής δεδομένων όπου θα αποθηκεύεται η ένα-προς-ένα σύνδεση των δημοσίων κλειδιών που έχουν δημιουργηθεί με την χρήση του ECDSA και της IP που χρησιμοποιήθηκε. Ο πίνακας αυτός θα πρέπει να ενημερώνεται τακτικά για κάθε αλλαγή της IP ή του κλειδιού που χρησιμοποιήθηκε από αυτήν τελευταία.

Το επόμενο αντίμετρο επιβάλλει την απουσία δημόσιας σύνδεσης μεταξύ των ID στις λίστες των κόμβων. Ένας τρόπος για να επιτευχθεί αυτό είναι η αποθήκευση των κατακερματισμένων ταυτοτήτων στην προσωρινή λίστα, με ένα μυστικό αριθμό μεγέθους 256 bit που αποκτάει κάθε κόμβος κατά την εκκίνησή του. Με παρόμοιο τρόπο λειτουργεί και η πλατφόρμα του Bitcoin η οποία δεν αντιμετωπίζει παρόμοιο πρόβλημα.

Ένα ακόμα αντίμετρο αποτρέπει τη χρήση της συνάρτησης *lookup(self),* όπου *self* είναι το δημόσια γνωστό κατακερματισμένο node ID του χρήστη, και την αντικατάστασή του με έναν μυστικό αριθμό ο οποίος διαλέγεται ομοιόμορφα στην τύχη, κάθε φορά που χρησιμοποιείται η συνάρτηση *lookup(self).*

Επιπροσθέτως, η ομάδα του πανεπιστημίου του Pittsburgh προτείνει μια πιο επιθετική αναζήτηση των κόμβων. Η αναζήτηση πρέπει να γίνεται ακόμα και όταν έχουν τοποθετηθεί στη λίστα κάποια ID χρηστών. Αυτό μειώνει την πιθανότητα του θύματος να κάνει όλες τις εξωτερικές του συνδέσεις αποκλειστικά προς κακόβουλους κόμβους.

Το τελευταίο αντίμετρο προτείνει την εξάλειψη του χρονικού διαστήματος μετά την επανεκκίνηση του κόμβου, κατά το οποίο μπορεί να γίνει λήψη μηνύματος ping, με αποτέλεσμα να τοποθετηθούν άγνωστα ID στη λίστα του θύματος. Αυτό θα εξαλείψει περαιτέρω το ενδεχόμενο πραγματοποίησης εξωτερικών συνδέσεων σε στόχους που δεν έχει εμπιστευτεί στο παρελθόν αλλά έχουν προστεθεί εξαιτίας της αποστολής μηνύματος *ping*.

3.1.2.δ. Πραγματικό περιστατικό

Μέχρι την στιγμή εγγραφής αυτής της διπλωματικής εργασίας, δεν έχει καταγραφεί κάποιο πραγματικό περιστατικό σε κάποια δημόσια αλυσίδα ή πλατφόρμα που μπορεί να γίνει αναφορά. Βέβαια αυτό δεν αποκλείει το γεγονός να έχει πραγματοποιηθεί η επίθεση και ο κόμβος – θύμα να μην μπόρεσε να το αντιληφθεί. Η φύση της επίθεσης είναι τέτοια, ώστε η απομόνωση από το δίκτυο επιτυγχάνεται χωρίς να είναι εύκολα αντιληπτή.

3.2. Ειδικές επιθέσεις

Στις παραγράφους που ακολουθούν εξετάζουμε κάποια ευπαθές τμήματα κώδικα από smart contracts τα οποία στο παρελθόν έχουν δημοσιευτεί στην πλατφόρμα του Ethereum και έχουν δεχτεί επιθέσεις. Θα περιγράψουμε τη λειτουργία του contract ενώ στη συνέχεια θα αναλύσουμε την ευπάθεια που παρουσιάζει και θα προτείνουμε κάποια αντίμετρα για αυτή. Τέλος, θα κάνουμε αναφορά σε ένα πραγματικό περιστατικό το οποίο εκμεταλλεύτηκε την αντίστοιχη ευπάθεια και τις συνέπειες που είχε στους χρήστες της πλατφόρμας.

3.2.1. Επανείσοδος (Re-Entrancy)

3.2.1.α. Περιγραφή

Τα smart contract έχουν σχεδιαστεί με τέτοιο τρόπο ώστε να μπορούν να πραγματοποιούν αιτήματα προς άλλα contract και να χρησιμοποιούν τον κώδικά τους ως επεκτάσεις. Επίσης, όπως έχει προαναφερθεί, η βασικότερη και συχνότερα χρησιμοποιούμενη λειτουργία τους είναι η μεταφορά ether από κάποια διεύθυνση της πλατφόρμας προς το smart contract ή από το ίδιο το smart contract προς την διεύθυνση που καλείται.

Σε κάθε περίπτωση το smart contract πρέπει να εκτελέσει κάποιο εξωτερικό αίτημα, είτε με σκοπό να μεταφέρει κάποια ποσότητα ether, είτε με σκοπό να χρησιμοποιήσει κάποιο τμήμα κώδικα το οποίο βρίσκεται αποθηκευμένο σε κάποιο άλλο contract της πλατφόρμας.

Αυτά τα εξωτερικά αιτήματα (external calls) είναι δυνατό να οδηγήσουν σε καταστάσεις όπου η εκτέλεση του κώδικα δεν σταματάει ποτέ με την προϋπόθεση ότι υπάρχουν οι πόροι και οι απαιτήσεις για τα αιτήματα είναι εκπληρωμένες. Έχοντας υπόψιν τα παραπάνω, ο επιτιθέμενος μπορεί να κατασκευάσει ένα smart contract το οποίο μπορεί να κρατήσει σε «ομηρία» το αρχικό smart contract, αναγκάζοντας το να εκτελέσει περαιτέρω κώδικα μετά από κάθε αίτημα, περιλαμβάνοντας ακόμα και αιτήματα προς το ίδιο. Κατά συνέπεια η εκτέλεση κώδικα συνεχίζεται εφόσον «ξαναμπαίνει» στο contract.

3.2.1.β. Η ευπάθεια

Η επίθεση αυτή μπορεί να πραγματοποιηθεί όταν το ευπαθές contract στέλνει κάποια ποσότητα ether προς μια άγνωστη διεύθυνση, η οποία ανήκει δυνητικά σε κάποιο χρήστη ή σε κάποιο άλλο contract. Ο επιτιθέμενος μπορεί να σχεδιάσει ένα τέτοιο contract το οποίο θα δημοσιεύσει στην διεύθυνση της πλατφόρμας στην οποία θα πραγματοποιηθεί το εξωτερικό κάλεσμα. Το κάλεσμα αυτό θα περιέχει τον κακόβουλο κώδικα στη συνάρτηση επιστροφής (fallback function). Έτσι, όταν το contract θα στείλει ether σε αυτή την διεύθυνση θα επικαλεστεί τον κακόβουλο κώδικα ο οποίος θα εκτελεστεί άμεσα. Συνήθως τέτοια τμήματα κακόβουλου κώδικα εκτελούν μια συνάρτηση στο ευπαθές contract, η οποία πραγματοποιεί κάποια ενέργεια ή περαιτέρω λειτουργίες που δεν ήταν αναμενόμενες από τον δημιουργό του ευπαθούς contract. Το όνομα «επανείσοδος» της επίθεσης αυτής προέρχεται από το γεγονός ότι το κακόβουλο contract καλεί πίσω (calls back) μια συνάρτηση στο contract-στόχο του αναγκάζοντάς το να κάνει «επανείσοδο» στην εκτέλεση κώδικα σε μια αυθαίρετη τοποθεσία.

Στο παράδειγμα που ακολουθεί παρουσιάζεται μια επίθεση επανεισόδου.

```
contract EtherStore {

    uint256 public withdrawalLimit = 1 ether;
    mapping(address => uint256) public lastWithdrawTime;
    mapping(address => uint256) public balances;

    function depositFunds() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdrawFunds (uint256 _weiToWithdraw) public {
        require(balances[msg.sender] >= _weiToWithdraw);
        require(_weiToWithdraw <= withdrawalLimit);
        require(now >= lastWithdrawTime[msg.sender] + 1 weeks);
        require(msg.sender.call.value(_weiToWithdraw)());
        balances[msg.sender] -= _weiToWithdraw;
        lastWithdrawTime[msg.sender] = now;
    }
}
```

Σε αυτό το contract έχουμε δύο δημόσιες συναρτήσεις. Η πρώτη είναι η *depositFunds()*, η οποία επιτρέπει σε οποιοδήποτε χρήστη καλέσει το contract να αποθηκεύσει κάποια ποσότητα ether, και δεύτερη συνάρτηση είναι η *withdrawFunds()*, η οποία επιτρέπει στον χρήστη να αφαιρέσει κάποια ποσότητα από την στιγμή που την έχει προηγουμένως αποθηκεύσει σε αυτό. Ο χρήστης δεν μπορεί να ζητήσει μεγαλύτερη ποσότητα ether από εκείνη που βρίσκεται ήδη αποθηκευμένη στη διεύθυνσή του, η οποία περιλαμβάνεται στο contract, και επίσης δεν μπορεί να χρησιμοποιήσει την συνάρτηση withdrawFunds() περισσότερες από μία φορά ανά εβδομάδα. Επιπλέον, το όριο που έχουμε θέσει σε αυτό το contract είναι ότι ο χρήστης δεν μπορεί να κάνει ανάληψη περισσότερα από 10^{18} wei που αντιστοιχούν σε 1 ether.

Από την στιγμή που θα πραγματοποιηθεί η μεταφορά της ποσότητας που έχει ζητηθεί ο λογαριασμός του χρήστη δέχεται και την ανάλογη μείωση στο διαθέσιμο υπόλοιπο. Αν εξετάσουμε προσεκτικά τον κώδικα, θα δούμε ότι αυτό ίσως και να μην συμβεί.

Υποθέτουμε ότι ο επιτιθέμενος δημιουργεί το contract που ακολουθεί και το ανεβάζει στην πλατφόρμα του Ethereum στην υποθετική διεύθυνση *0x1231231231231231231231231231*.

```
import "EtherStore.sol";

contract Attack {
  EtherStore public etherStore;

  constructor(address _etherStoreAddress) {
      etherStore = EtherStore(_etherStoreAddress);
  }

  function pwnEtherStore() public payable {
      require(msg.value >= 1 ether);
      etherStore.depositFunds.value(1 ether)();
      etherStore.withdrawFunds(1 ether);
  }

  function collectEther() public {
      msg.sender.transfer(this.balance);
```

```
  }

  function () payable {
    if (etherStore.balance > 1 ether) {
      etherStore.withdrawFunds(1 ether);
    }
  }
}
```

Στην αρχή δημιουργούμε μια μεταβλητή η οποία περιέχει την δημόσια διεύθυνση του contract το οποίο θέλουμε να εκμεταλλευτούμε και στη συνέχεια κατασκευάζουμε το νόμιμο φορτίο (payload) το οποίο θα χρησιμοποιήσουμε για να κάνουμε αποθήκευση ένα ether στο απομακρυσμένο contract, πληρώνοντας και τα επιπλέον έξοδα (gas) που θα χρειαστούν για αυτή τη συναλλαγή και στη συνέχεια να κάνουμε ανάληψη του αρχικού ποσού (ένα ether). Η κλήση των συναρτήσεων με αυτόν τον τρόπο είναι εφικτή διότι, όπως επισημάνθηκε και σε προηγούμενη παράγραφο, οι δύο συναρτήσεις είναι δημόσιες.

Στη συνέχεια έχουμε κατασκευάσει την κακόβουλη συνάρτηση, η οποία ελέγχει εάν το υπόλοιπο στο contract είναι μεγαλύτερο του ενός ether. Στην περίπτωση που αυτό ισχύει τότε καλεί την *withdrawFunds()* ξανά. Για χάρη του παραδείγματος προϋποτίθεται ότι στο contract έχουν αποθηκευτεί ήδη πέντε ether από άλλους χρήστες πριν ξεκινήσει η πραγματοποίηση της επίθεσης.

Όταν το contract ανέβει στο δίκτυο και εκτελεστεί ο κακόβουλος κώδικας θα συμβούν τα εξής:

1. Από την στιγμή που όλες οι απαιτήσεις πληρούνται, τότε το πρώτο πράγμα που θα κάνει το κακόβουλο contract είναι να στείλει ένα ether προς το ευπαθές contract, EtherStore που είναι ο στόχος.

2. Σε αυτό το σημείο το ευπαθές contract έχει έξι ether διαθέσιμο υπόλοιπο, εκ των οποίων το ένα ανήκει στον επιτιθέμενο.

3. Στην συνέχεια κάνει ανάκτηση του ενός ether που μόλις αποθήκευσε.

4. Το υπόλοιπο του EtherStore είναι τώρα πέντε ether.

5. Επόμενο βήμα είναι η εκτέλεση της συνάρτησης επιστροφής που ελέγχει το υπόλοιπο του ευπαθούς contract και εάν είναι μεγαλύτερο από ένα, τότε ξανακαλεί την συνάρτηση *withdrawFunds()* η οποία επανείσερχεται στο contract.

6. Σε αυτό το δεύτερο κάλεσμα, το υπόλοιπο του επιτιθέμενου φαίνεται να είναι ακόμα ένα ether, γιατί η επόμενη γραμμή μετά την μεταφορά του ether δεν έχει ακόμα εκτελεστεί. Επιπλέον, δεν έχει ανανεωθεί η ημερομηνία της ανάληψης, οπότε πληρούνται όλες οι προδιαγραφές για να συνεχιστεί η εκτέλεση του κώδικα.

7. Γίνεται ανάκτηση ενός ακόμα ether.

Η διαδικασία αυτή θα επαναληφθεί μέχρι να μην πληρείται ο έλεγχος που βρίσκεται μέσα στη συνάρτηση επιστροφής, δηλαδή το υπόλοιπο του EtherStore να είναι μικρότερο του ενός.

3.2.1.γ. Αντίμετρα

Ένα μέτρο αντιμετώπισης των επιθέσεων re-entrance είναι να χρησιμοποιείται η συνάρτηση *transfer()*, η οποία υπάρχει στον πηγαίο κώδικα της Solidity, κάθε φορά που αποστέλλεται μια ποσότητα ether σε κάποιο εξωτερικό contract. Αυτή η συνάρτηση χρησιμοποιεί μόνο 2300 gas για κάθε κάλεσμα, που δεν επαρκούν ώστε να καλέσει η διεύθυνση προορισμού κάποιο άλλο contract ή ακόμα και το ίδιο.

Άλλο αντίμετρο είναι να επιβεβαιώσουμε ότι όλες οι λογικές αλλαγές που πρέπει να γίνουν στις μεταβλητές έχουν πραγματοποιηθεί πριν την τελική αποστολή της ποσότητας ether. Είναι καλή πρακτική για όλα τα contract να τοποθετείται οποιοδήποτε τμήμα κώδικα που πραγματοποιεί εξωτερικές κλήσεις σε άγνωστες διευθύνσεις σαν η τελευταία διαδικασία μέσα στην συνάρτηση. Αυτή η πρακτική είναι γνωστή ως και το μοτίβο (pattern) Έλεγχοι – Επιδράσεις – Αλληλεπιδράσεις (Checks – Effects – Interactions).

Ένα επιπλέον αντίμετρο το οποίο μπορεί να σταματήσει τέτοιου είδους επιθέσεις είναι να χρησιμοποιηθεί κάποιο mutex. Το mutex είναι μια μεταβλητή η οποία αποθηκεύει την κατάσταση του contract και κλειδώνει το contract κατά την διάρκεια της εκτέλεσης κώδικα.

3.2.1.δ. Πραγματικό περιστατικό

Στα πρώιμα στάδια ανάπτυξης της πλατφόρμας Ethereum μια από τις μεγαλύτερες επιθέσεις που πραγματοποιήθηκαν είναι η επίθεση στον DAO, Decentralized Autonomous Organization, η οποία είχε σαν αποτέλεσμα την αφαίρεση περίπου 150 εκατομμύριων δολαρίων σε Ether από το contract.

3.2.2. Ροές υπερχείλισης και υποχείλισης (Arithmetic Over/Under flows)

3.2.2.α. Περιγραφή

Η Ethereum Virtual Machine καθορίζει πάντα σταθερό μέγεθος για όλους τους τύπους δεδομένων, άρα και για τους ακέραιους αριθμούς. Αυτό σημαίνει ότι όταν χρησιμοποιείται μια μεταβλητή για την αποθήκευση κάποιου ακέραιου αριθμού, υπάρχει ένα συγκεκριμένο εύρος τιμών που μπορεί να λάβει. Για παράδειγμα μια μεταβλητή που έχει οριστεί ως *uint8* (unsigned integer 8), μπορεί να αποθηκεύσει μόνο αριθμούς που βρίσκονται μεταξύ του μηδέν και του 255. Αν δοθεί εντολή να αποθηκευτεί ο αριθμός 256 σε αυτή την μεταβλητή, θα έχει σαν αποτέλεσμα την αποθήκευση του αριθμού μηδέν.

3.2.2.β. Η ευπάθεια

Η ευπάθεια αυτή μπορεί να εμφανιστεί όταν μια μαθηματική πράξη εκτελείται και χρησιμοποιείται μια μεταβλητή σταθερού μεγέθους για να αποθηκευτεί το αποτέλεσμα, και αυτό είναι εκτός του εύρους τιμών που μπορεί να δεχτεί ο συγκεκριμένος τύπος.

Για παράδειγμα, έστω ότι έχουμε μια μεταβλητή uint8, η οποία μπορεί να αποθηκεύσει, όπως προαναφέρθηκε, ακέραιους αριθμούς από το μηδέν έως και το 255. Αν αυτή η μεταβλητή περιέχει τον αριθμό μηδέν και από αυτή αφαιρεθεί ο αριθμός ένα, τότε το αποτέλεσμα το οποίο θα αποθηκευτεί μετά την εκτέλεση της μαθηματικής πράξης, είναι ο αριθμός 255. Αυτή η ευπάθεια ονομάζεται υποχείλιση (underflow). Αυτό που συνέβη είναι ότι προσπαθήσαμε να αποθηκεύσουμε έναν αριθμό ο οποίος είναι μικρότερος κατά ένα από τον μικρότερο δυνατό αριθμό που θα μπορούσε να αποθηκευτεί σε αυτή τη μεταβλητή. Αυτό που συνέβη ήταν να

αποθηκευτεί ο επόμενος αριθμός σε κυκλική θέση, όπου στην περίπτωση αυτή είναι ο μεγαλύτερος αριθμός όπου μπορεί να αποθηκευτεί σε αυτόν τον τύπο μεταβλητής.

Αντίστοιχα, αν προσθέσουμε τον αριθμό 256 σε μια μεταβλητή όπου έχουμε ορίσει και εδώ ως uint8 και περιέχει ήδη κάποιον αριθμό, τότε ο αριθμός αυτός θα παραμείνει αμετάβλητος, διότι έχουμε προσπελάσει όλους τους αριθμούς και έχουμε ξαναρχίσει από την αρχή του αποδεκτού μήκους. Για να γίνει κατανοητό, προσθέτοντας τον αριθμό 256 σε μια μεταβλητή, έχει το ίδιο αποτέλεσμα σαν να προσθέτουμε στην ίδια μεταβλητή τον αριθμό 0. Αυτή η ευπάθεια ονομάζεται υπερχείλιση (overflow).

Αυτές οι δύο μαθηματικές ευπάθειες μπορεί να οδηγήσουν σε κατάχρηση του κώδικα και να δημιουργήσουν απροσδόκητες λογικές ροές.

Μπορούμε να δούμε τον παρακάτω ευπαθή κώδικα και ποιές μπορεί να είναι οι πιθανές επιπτώσεις του.

```
contract TimeLock {

    mapping(address => uint) public balances;
    mapping(address => uint) public lockTime;
    function deposit() public payable {
        balances[msg.sender] += msg.value;
        lockTime[msg.sender] = now + 1 weeks;
    }
    function increaseLockTime(uint _secondsToIncrease) public {
        lockTime[msg.sender] += _secondsToIncrease;
    }
    function withdraw() public {
        require(balances[msg.sender] > 0);
        require(now > lockTime[msg.sender]);
        uint transferValue = balances[msg.sender];
        balances[msg.sender] = 0;
        msg.sender.transfer(transferValue);
    }
}
```

Το παραπάνω contract δίνει τη δυνατότητα σε κάποιον χρήστη να αποθηκεύσει μια ποσότητα από ether και να το κλειδώσει για τουλάχιστον μια εβδομάδα. Ο χρήστης έχει τη δυνατότητα να επεκτείνει αυτό τον χρόνο σε παραπάνω από τον προεπιλεγμένο, αλλά από την στιγμή που η ποσότητα αποθηκευτεί στο contract είναι ασφαλής για αυτό το χρονικό διάστημα.

Για να γίνει κατανοητό πως μπορεί κάποιος να εκμεταλλευτεί αυτό το contract ας υποθέσουμε ότι ο επιτιθέμενος αποθηκεύει 100 ether, χρησιμοποιώντας την συνάρτηση deposit(). Δεν θέλει να προσθέσει παραπάνω χρόνο, οπότε αφήνει τον προεπιλεγμένο ελάχιστο χρόνο της μιας εβδομάδας. Όπως βλέπουμε στην αρχή του contract η μεταβλητή lockTime είναι ορισμένη ως δημόσια, πράγμα που σημαίνει ότι όλοι οι χρήστες έχουν πρόσβαση σε αυτή. Επίσης παρατηρούμε ότι η μεταβλητή αυτή είναι του τύπου uint, που είναι ο σύντομος τρόπος δήλωσης του τύπου uint256.

Ο επιτιθέμενος σε αυτό το σημείο μπορεί να δημιουργήσει μια μεταβλητή στην οποία θα αποθηκεύσει τον δικό του χρόνο αναμονής, ας την ονομάσουμε myLockTime, την οποία θα περάσει σαν παράμετρο καλώντας την increaseLockTime() συνάρτηση του contract ως εξής:

2 ^ 256 – myLockTime

Αυτό θα έχει σαν συνέπεια η εντολή που βρίσκεται μέσα στην συνάρτηση increaseLockTime να πάρει την εξής μορφή:

lockTime[msg.sender] = lockTime[msg.sender] + (2 ^ 256) – lockTime[msg.sender]

Το οποίο, απλοποιημένο έχει ως εξής:

lockTime[msg.sender] = 2 ^ 256 = 0

Με αυτόν τον τρόπο ο επιτιθέμενος έχει τροποποιήσει τον χρόνο αναμονής από μια εβδομάδα σε μηδέν. Μπορεί αμέσως να καλέσει την συνάρτηση *withdraw()* και να λάβει το περιεχόμενο που αντιστοιχεί σε αυτόν το λογαριασμό μέσα στο contract.

Αντίστοιχο παράδειγμα μπορούμε να βρούμε και στις διαδικτυακές προκλήσεις του Ethernaut, όπου σε μια άσκηση μας ζητείται να εντοπίσουμε την ευπάθεια στον παρακάτω κώδικα:

```
pragma solidity ^0.4.18;
```

```
contract Token {
  mapping(address => uint) balances;
  uint public totalSupply;

  function Token(uint _initialSupply) {
    balances[msg.sender] = totalSupply = _initialSupply;
  }
  function transfer(address _to, uint _value) public returns (bool) {
    require(balances[msg.sender] - _value >= 0);
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    return true;
  }
  function balanceOf(address _owner) public constant returns (uint balance) {
    return balances[_owner];
  }
}
```

Με μια γρήγορη ματιά, βάσει όσων περιγράψαμε προηγουμένως, μπορούμε να δούμε ότι παρόμοια ευπάθεια υπάρχει και σε αυτό το contract. Πιο συγκεκριμένα, στη συνάρτηση *transfer()* όπου γίνεται ο έλεγχος για το εάν ο χρήστης διαθέτει το υπόλοιπο το οποίο θέλει να μεταφέρει στην διεύθυνση της επιθυμίας του, αν το υπόλοιπο του είναι μηδέν (0) και το ποσό που θέλει να μεταφέρει είναι ένας μη μηδενικός αριθμός, ο έλεγχος:

```
balances[msg.sender] - _value >= 0
```

θα επιστρέφει πάντα Αληθές (True), γιατί η μεταβλητή balance είναι του τύπου uint256, οπότε το αποτέλεσμα αυτής της αφαίρεσης θα επιστρέφει πάντα έναν αριθμό μεγαλύτερο του μηδενός, λόγω του φαινομένου της υποχείλισης όπως εξηγήσαμε παραπάνω.

3.2.2.γ. Αντίμετρα

Ένα από τα αντίμετρα τα οποία μπορούμε να πάρουμε για να αποφύγουμε ευπάθειες τέτοιου τύπου είναι να χρησιμοποιήσουμε μαθηματικές βιβλιοθήκες τις οποίες είτε βρίσκουμε έτοιμες, είτε τις κατασκευάζουμε εμείς. Οι βιβλιοθήκες αυτές αντικαθιστούν τους τελεστές των μαθηματικών πράξεων αφαίρεσης, πρόσθεσης, πολλαπλασιασμού και διαίρεσης.

Μια από τις πιο πολυχρησιμοποιημένες βιβλιοθήκες στην κοινότητα του Ethereum, είναι η Safe Math Library (://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol) της ομάδας OpenZeppelin. Η ίδια ομάδα παρέχει πολλές ασφαλείς βιβλιοθήκες, τις οποίες προγραμματιστές από όλο τον κόσμο μπορούν να βρουν στο GitHub (://github.com/OpenZeppelin/openzeppelin-solidity).

Αν θέλαμε να χρησιμοποιήσουμε την παραπάνω βιβλιοθήκη για να διορθώσουμε τον κώδικα του αρχικού contract που παρουσιάστηκε σε αυτή την ενότητα, το αποτέλεσμα θα ήταν το εξής:

```
library SafeMath {

  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a / b;
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}
```

```
contract TimeLock {
    using SafeMath for uint;
    mapping(address => uint256) public balances;
    mapping(address => uint256) public lockTime;

    function deposit() public payable {
        balances[msg.sender] = balances[msg.sender].add(msg.value);
        lockTime[msg.sender] = now.add(1 weeks);
    }

    function increaseLockTime(uint256 _secondsToIncrease) public {
        lockTime[msg.sender] = lockTime[msg.sender].add(_secondsToIncrease);
    }

    function withdraw() public {
        require(balances[msg.sender] > 0);
        require(now > lockTime[msg.sender]);
        uint transferValue = balances[msg.sender];
        balances[msg.sender] = 0;
        msg.sender.transfer(transferValue);
    }
}
```

3.2.2.δ. Πραγματικό περιστατικό

Στις αρχές του έτους 2018, μια διαδικτυακή ομάδα από προγραμματιστές αποφάσισε να κτίσει μια εταιρία πυραμίδα στο Ethereum, με το όνομα Proof of Weak Hands Coin. Η υλοποίηση αυτού του contract ήταν βασισμένη στην ιδέα του Jocken Hoenicke, που είχε παρουσιάσει λίγο καιρό πριν τον τρόπο με τον οποίο θα μπορούσε να λειτουργήσει ένα τέτοιο Token στο Ethereum. Το τελικό δημοσιευμένο contract, στο οποίο δεν είχε ενσωματωθεί κάποια βιβλιοθήκη για να πραγματοποιεί βασικές μαθηματικές πράξεις, χρησιμοποιήθηκε από μια ομάδα επιτιθέμενων που εκμεταλλεύτηκαν την εμφανή ευπάθεια υποχείλισης και απέσπασαν 866 ether από αυτό.

Δεύτερο μεγάλο συμβάν με βάση την ίδια ευπάθεια συνέβη σε διάφορες υλοποιήσεις του ERC20 Token το οποίο χρησιμοποιείται κατά κύριο λόγο στην πλατφόρμα του Ethereum, και συγκεκριμένα στην συνάρτηση *batchTransfer()*. Για το αντίστοιχο θέμα ο οργανισμός NIST έχει δημοσιεύσει και το CVE-2018-10299.

3.2.3. Καθοριστικά ορατότητας (Default Visibilities)

3.2.3.α. Περιγραφή

Οι συναρτήσεις στην Solidity έχουν προσδιοριστικά ορατότητας τα οποία καθορίζουν πως οι συναρτήσεις αυτές επιτρέπουν να καλεστούν. Κάθε μια από αυτές τις συναρτήσεις καθορίζει πότε μια συνάρτηση μπορεί να καλεστεί από κάποιον εξωτερικό χρήστη ή από άλλα contracts, μόνο εσωτερικά μέσα στο ίδιο contract ή μόνο εξωτερικά. Υπάρχουν τέσσερα διαφορετικά προσδιοριστικά ορατότητας που μπορούν να χρησιμοποιηθούν είτε μόνα τους είτε συνδυαστικά και αυτά είναι: external, public, internal, private.

- External (εξωτερική): Εξωτερική ορατότητα σημαίνει ότι η συγκεκριμένη συνάρτηση μπορεί να καλεστεί από άλλα contracts και μέσω συναλλαγών. Μια συνάρτηση αυτού του τύπου δεν μπορεί να καλεστεί εσωτερικά στο ίδιο contract. Επιπλέον, οι εξωτερικές συναρτήσεις είναι περισσότερο αποτελεσματικές όταν λαμβάνουν μεγάλες ποσότητες δεδομένων.

- Public (δημόσια): Δημόσια ορατότητα σημαίνει ότι η συνάρτηση αυτή μπορεί να καλεστεί είτε μέσα στο ίδιο το contract ή με την αποστολή μηνυμάτων.

- Internal (εσωτερική): Εσωτερική ορατότητα σημαίνει ότι η συνάρτηση μπορεί να χρησιμοποιηθεί και να καλεστεί από το ίδιο contract ή άλλα contracts τα οποία είναι παράγοντα αυτού.

- Private (ιδιωτική): Ιδιωτική ορατότητα σημαίνει ότι η συνάρτηση αυτή μπορεί να καλεστεί μόνο στο ίδιο contract το οποίο έχει αρχικοποιηθεί.

Οι συναρτήσεις έχουν σαν προεπιλεγμένο τύπο τον δημόσιο, που επιτρέπει σε άλλους χρήστες να τις καλέσουν ξεχωριστά. Λανθασμένη χρήση αυτών των τύπων ορατότητας οδηγεί σε καταστροφικές ευπάθειες στα smart contracts.

3.2.3.β. Η ευπάθεια

Αυτή η ευπάθεια προέρχεται καθαρά από την μεριά των προγραμματιστών και την λανθασμένη αντίληψη τους για τον τύπο των συναρτήσεων που χρησιμοποιούν. Ο προεπιλεγμένος τύπος για κάθε συνάρτηση στη Solidity, είναι δημόσιος. Πολλές φορές οι προγραμματιστές δεν αλλάζουν τον τύπο σε ιδιωτικό ή δεν ορίζουν κανέναν τύπο, είτε επειδή βιάζονται είτε επειδή θεωρούν ότι αυτή η συνάρτηση δεν θα επηρεάσει το τελικό contract, με αποτέλεσμα άλλοι χρήστες να έχουν πρόσβαση σε τμήματα κώδικα που δεν θα έπρεπε.

Ένα σύντομο παράδειγμα είναι το παρακάτω:

```
contract HashForEther {

   function withdrawWinnings() {
      require(uint32(msg.sender) == 0);
      _sendWinnings();
   }

   function _sendWinnings() {
      msg.sender.transfer(this.balance);
   }
}
```

Αυτό το contract είναι ένα παιχνίδι τύχης που ελέγχει αν οι οκτώ τελευταίοι χαρακτήρες της διεύθυνσης του χρήστη που παίρνει μέρος είναι μηδενικά. Αν είναι, τότε το contract καλεί την συνάρτηση *_sendWinnings()* και στέλνει την ποσότητα Ether που είναι αποθηκευμένη στο contract, στη διεύθυνση του νικητή. Το πρόβλημα είναι ότι επειδή ο προγραμματιστής του συγκεκριμένου contract, δεν έχει λάβει υπόψη την ορατότητα της συγκεκριμένης συνάρτησης και την έχει αφήσει στην προεπιλεγμένη μορφή της, δηλαδή δημόσια, οποιοσδήποτε χρήστης μπορεί να την καλέσει υπεξαιρώντας την υπάρχουσα ποσότητα ether, χωρίς να πάρει μέρος στο παιχνίδι και χωρίς να είναι νικητής.

3.2.3.γ. Αντίμετρα

Είναι καλή πρακτική να αναγράφεται πάντα η ορατότητα της κάθε συνάρτησης όταν δημιουργείται το contract, ακόμα και αν αυτή η συνάρτηση πρέπει να είναι δημόσια. Στις τελευταίες εκδόσεις της Solidity, κατά τη διάρκεια της μεταγλώττισης εμφανίζονται ειδοποιήσεις για τις συναρτήσεις που η ορατότητα δεν έχει οριστεί.

3.2.3.δ. Πραγματικό περιστατικό

Τα multisignature πορτοφόλια

Σε όλες τις πλατφόρμες του blockchain, οι χρήστες χρησιμοποιούν πορτοφόλια για να αποθηκεύουν τα ether που έχουν στην κατοχή τους. Στο Ethereum το multisignature ψηφιακό αυτό πορτοφόλι, είναι ένα ακόμα contract το οποίο τρέχει στο δίκτυο του Ethereum και έχει τη δυνατότητα να χρησιμοποιείται από πολλούς συμμετέχοντες καθημερινά, επιτρέποντας τους να κάνουν μικρές αναλήψεις με όριο, αλλά και την δυνατότητα να κάνουν συναλλαγές με μεγάλες ποσότητες όταν αυτές επαληθευτούν από πολλαπλές υπογραφές.

Κάθε multisignature πορτοφόλι θα πρέπει να έχει τουλάχιστον τα εξής χαρακτηριστικά στη βάση του:

- Μια λίστα από επιτρεπόμενους χρήστες
- Κανόνες για το πλήθος των χρηστών που πρέπει να συμφωνήσουν προτού γίνει κάποια συναλλαγή
- Έναν τρόπο ώστε να λαμβάνει Ether
- Έναν τρόπο ώστε να κάνει αιτήματα προς το δίκτυο και άλλες διευθύνσεις
- Έναν τρόπο ώστε να μπορούν οι χρήστες του να συμφωνήσουν για την πραγματοποίηση του αιτήματος
- Έναν τρόπο ώστε να μπορεί να επανεκτελεστεί το αίτημα σε περίπτωση που απέτυχε την πρώτη φορά.

Η πρώτη μεγάλη επίθεση η οποία εκμεταλλεύτηκε την παρούσα ευπάθεια έγινε στο Parity MultiSig Wallet. Το αποτέλεσμα αυτής της επίθεσης ήταν να κλαπούν 31 εκατομμύρια

δολάρια σε Ether, κυρίως από τρία διαφορετικά πορτοφόλια. Εξετάζοντας τον κώδικα του contract έχουμε το παρακάτω ευπαθές τμήμα:

```
contract WalletLibrary is WalletEvents {

  function initMultiowned(address[] _owners, uint _required) {
    m_numOwners = _owners.length + 1;
    m_owners[1] = uint(msg.sender);
    m_ownerIndex[uint(msg.sender)] = 1;
    for (uint i = 0; i < _owners.length; ++i)
    {
      m_owners[2 + i] = uint(_owners[i]);
      m_ownerIndex[uint(_owners[i])] = 2 + i;
    }
    m_required = _required;
  }

  ...

  function initWallet(address[] _owners, uint _required, uint _daylimit) {
    initDaylimit(_daylimit);
    initMultiowned(_owners, _required);
  }
}
```

Παρατηρούμε ότι στις δύο αυτές συναρτήσεις δεν έχει οριστεί κανένα καθοριστικό ορατότητας, οπότε και οι δύο έχουν οριστεί από προεπιλογή σαν δημόσιες. Η συνάρτηση *initWallet()* καλείται κατά την κατασκευή του ψηφιακού πορτοφολιού, κατά την οποία ορίζεται και οι νόμιμοι κάτοχοι του, *_owners*, όπως φαίνεται και στον κώδικα της συνάρτησης *initMultiowned()*.

Για τον λόγο που προαναφέρθηκε, επειδή και οι δύο αυτές συναρτήσεις έχουν οριστεί δημόσιες από παράβλεψη του προγραμματιστή, ο επιτιθέμενος μπορεί να τις καλέσει ανά πάσα στιγμή και να αλλάξει την ιδιοκτησία του πορτοφολιού. Από την στιγμή που είναι ο ιδιοκτήτης μπορεί να κάνει ανάληψη όλου του ποσού που βρίσκεται μέσα σε αυτό.

3.2.4. Κώδικας λειτουργίας Delegatecall

3.2.4.α. Περιγραφή

Στην πλατφόρμα του Ethereum οι προγραμματιστές των contract χρησιμοποιούν τους κωδικούς λειτουργίας (opcodes) CALL και DELEGATECALL ώστε να πετύχουν την ευελιξία που θέλουν στα προγράμματα τους. Οι τυπικές κλήσεις εξωτερικού μηνύματος χειρίζονται από τον κώδικα λειτουργίας CALL και στη συνέχεια ο κώδικας μπορεί να εκτελεστεί στο πλαίσιο του απομακρυσμένου contract ή της εξωτερικής συνάρτησης. Στην περίπτωση που χρησιμοποιηθεί ο κώδικας λειτουργίας DELEGATECALL, τότε ο κώδικας εκτελείται στην στοχευμένη διεύθυνση αλλά στο πλαίσιο του καλούντος contract, χωρίς να αλλάζουν οι τιμές των μεταβλητών *msg.sender* και *msg.value*. Αυτή η λειτουργία επιτρέπει τη χρήση βιβλιοθηκών και την επαναχρησιμοποίηση κώδικα για μελλοντικά contract.

Αν και η διαφορά αυτών των δύο κωδικών λειτουργίας είναι απλή και εμφανής, η χρήση της DELEGATECALL μπορεί να οδηγήσει σε απροσδόκητη εκτέλεση κώδικα.

3.2.4.β. Η ευπάθεια

Έχει αποδειχθεί ότι δεν είναι εύκολο να δημιουργηθούν βιβλιοθήκες στο χώρο του Ethereum που δεν περιλαμβάνουν ευπάθειες. Αυτό ισχύει και στην περίπτωση του κώδικα λειτουργίας DELEGATECALL. Ο ίδιος ο κώδικας μέσα στην βιβλιοθήκη μπορεί να είναι αρκετά ασφαλής και να μην περιέχει κάποια ευπάθεια, ωστόσο όταν εκτελείται ως πρόσθετος κώδικας μιας άλλης εφαρμογής νέες ευπάθειες μπορεί να δημιουργηθούν. Για να γίνει κατανοητό θα χρησιμοποιήσουμε τον παρακάτω κώδικα της βιβλιοθήκης Fibonacci, που μπορεί να δημιουργήσει ακολουθίες αριθμών Fibonacci και παρόμοιων τύπων.

```
contract FibonacciLib {
  uint public start;
  uint public calculatedFibNumber;

  function setStart(uint _start) public {
    start = _start;
  }
```

```
  function setFibonacci(uint n) public {
    calculatedFibNumber = fibonacci(n);
  }

  function fibonacci(uint n) internal returns (uint) {
    if (n == 0) return start;
    else if (n == 1) return start + 1;
    else return fibonacci(n - 1) + fibonacci(n - 2);
  }
}
```

Η βιβλιοθήκη παρέχει μια συνάρτηση που μπορεί να δημιουργήσει τον νιοστό αριθμό της ακολουθίας, επιτρέποντας στον χρήστη να θέσει εκείνος τον αριθμό της επιθυμίας του από όπου θα γίνει η εκκίνηση του υπολογισμού.

Υποθέτουμε ότι θέλουμε να χρησιμοποιήσουμε αυτή τη βιβλιοθήκη στο παρακάτω contract:

```
contract FibonacciBalance {

  address public fibonacciLibrary;
  uint public calculatedFibNumber;
  uint public start = 3;
  uint public withdrawalCounter;
  bytes4 constant fibSig = bytes4(sha3("setFibonacci(uint256)"));

  constructor(address _fibonacciLibrary) public payable {
    fibonacciLibrary = _fibonacciLibrary;
  }
  function withdraw() {
    withdrawalCounter += 1;
    require(fibonacciLibrary.delegatecall(fibSig, withdrawalCounter));
    msg.sender.transfer(calculatedFibNumber * 1 ether);
  }
  function() public {
    require(fibonacciLibrary.delegatecall(msg.data));
  }
```

```
}
```

Αυτό το contract επιτρέπει σε ένα συμμετέχοντα να κάνει ανάληψη μιας ποσότητας ether, με την ποσότητα αυτή να είναι ίση με τον αριθμό της ακολουθίας Fibonacci που αντιστοιχεί στην θέση του συμμετέχοντα που πραγματοποιεί την ανάληψη.

Έστω ότι υπάρχουν πέντε συμμετέχοντες οι οποίοι θα χρησιμοποιήσουν το παραπάνω contract για να κάνουν ανάληψη κάποιας ποσότητας ether. Κανείς από αυτούς τους χρήστες δεν μπορεί να δηλώσει την ποσότητα που θέλει να λάβει. Το ποσό της ανάληψης θα οριστεί βάσει της ακολουθίας. Ο πρώτος χρήστης θα παραλάβει ένα ether, ο δεύτερος χρήστης θα παραλάβει επίσης ένα ether, ο τρίτος δύο, ο τέταρτος θα λάβει τρία ether ενώ ο πέμπτος θα δεχθεί πέντε.

Παρατηρώντας καλύτερα το παραπάνω τμήμα κώδικα βλέπουμε ότι χρησιμοποιείται η μεταβλητή *fibSig*, στην οποία αποθηκεύονται τα τέσσερα πρώτα bytes της κατακερματισμένης ακολουθίας χαρακτήρων, που αναγράφει *setFibonacci(uint256)*. Η χρήση αυτής της μεταβλητής γίνεται στη συνέχεια, όταν χρησιμοποιείται η συνάρτηση delegatecall και η *fibSig* δίδεται ως πρώτο όρισμα, για να καθοριστεί ότι θέλουμε να χρησιμοποιήσουμε τη συνάρτηση *fibonnaci(uint256)*. Το δεύτερο όρισμα της συνάρτησης delegatecall είναι η παράμετρος που περνάμε στη συνάρτηση που θα χρησιμοποιηθεί.

Το πρώτο πράγμα που μπορεί να παρατηρήσει κάποιος στη βιβλιοθήκη και στο contract, είναι ότι και οι δύο χρησιμοποιούν την μεταβλητή *start*. Στη βιβλιοθήκη η μεταβλητή *start* έχει οριστεί ίση με μηδέν, ενώ στο contract της έχει αποδοθεί η τιμή τρία. Επίσης, μπορεί να παρατηρηθεί ότι η συνάρτηση επιστροφής του contract FibonacciBalance, επιτρέπει όλα τα αιτήματα να πραγματοποιηθούν προς την βιβλιοθήκη, άρα και το αίτημα προς την συνάρτηση της *setStart()*. Υπενθυμίζοντας ότι με τη χρήση της DELEGATECALL η κατάσταση του contract παραμένει σταθερή, μπορεί να θεωρηθεί λανθασμένα ότι αυτή η συνάρτηση επιτρέπει την αλλαγή της κατάστασης της μεταβλητής *start* στο contract FibonacciBalance. Σε αυτή την περίπτωση, θα επέτρεπε στον χρήστη να αποκτήσει μεγαλύτερη ποσότητα ether, ως αποτέλεσμα της εξάρτησης της *calculatedFibNumber* από την μεταβλητή *start*. Στην

πραγματικότητα όμως, η συνάρτηση *setStart()* δεν μπορεί να αλλάξει την ποσότητα της μεταβλητής *start* που περιλαμβάνεται στο ίδιο το contract.

Γνωρίζοντας πως δουλεύουν οι μεταβλητές κατάστασης που εξετάσαμε στην παράγραφο των μεταβλητών, μπορούμε να αντιληφθούμε που ξεκινάει το πρόβλημα.

Για παράδειγμα ας εξετάσουμε τον κώδικα της βιβλιοθήκης. Η βιβλιοθήκη περιλαμβάνει δύο μεταβλητές κατάστασης, την μεταβλητή *start* και την *calculatedFibNumber*. Η πρώτη μεταβλητή *start*, αποθηκεύεται στο slot[0]. Η δεύτερη μεταβλητή, *calculatedFibNumber*, αποθηκεύεται στην επόμενη ελεύθερη θέση, που είναι το slot[1]. Αν εξετάσουμε ξανά την συνάρτηση *setStart()*, θα δούμε ότι ορίζει την τιμή της μεταβλητής *start* στην τιμή της παραμέτρου που χρησιμοποιήθηκε στο κάλεσμα της. Παρόμοια, η συνάρτηση *setFibonacci()* ορίζει την μεταβλητή *calculatedFibNumber* στο αποτέλεσμα της συνάρτησης *fibonacci(n)*. Τα slot[0] και slot[1] έχουν πλέον ανανεωθεί στις καινούργιες τιμές τους.

Στη συνέχεια εξετάζουμε τον κώδικα του contract, FibonacciBalance. Η πρώτη μεταβλητή που χρησιμοποιείται είναι η διεύθυνση της *fibonnaciLibrary* και τοποθετείται στο slot[0] και η δεύτερη μεταβλητή είναι η *calculcatedFibNumber* η οποία τοποθετείται στο slot[1]. Αυτή η λανθασμένη αντιστοίχιση είναι η αιτία δημιουργίας της ευπάθειας. Η DELEGETACALL διατηρεί το περιεχόμενο του contract. Αυτό σημαίνει ότι ο κώδικας που εκτελείται μέσω της συνάρτησης delegetacall, θα επιδράσει στην κατάσταση του καλεσμένου contract. Λίγες γραμμές παρακάτω στο contract που εξετάζουμε εκτελείται η εντολή *fibonacciLibrary.delegatecall(fibSig, withdrawalCounter)*. Αυτή η εντολή καλεί τη συνάρτηση *setFibonnaci()* και τροποποιεί το περιεχόμενο του slot[1], που στο δικό μας περιεχόμενο είναι η *calculatedFibNumber*, όπως ήταν αναμενόμενο.

Ωστόσο, στο slot[0] της library υπάρχει η αποθηκευμένη μεταβλητή *start* και η τιμή που αυτή περιέχει. Στο contract, στο slot[0] περιέχεται η δημόσια διεύθυνση της βιβλιοθήκης fibonacciLibrary που θέλουμε να χρησιμοποιήσουμε. Αυτό σημαίνει ότι καλώντας την συνάρτηση *fibonacci()* θα επιστραφεί ένα μη αναμενόμενο αποτέλεσμα. Αυτό θα συμβεί διότι το slot[0] αναφέρεται στην μεταβλητή *start*, η οποία στην προκειμένη περίπτωση είναι η διεύθυνση. Κατά συνέπεια είναι πιθανό σε αυτό το σενάριο η συνάρτηση *withdraw()* να

αναιρεθεί γιατί δεν περιέχει μια ποσότητα ether τόσο μεγάλη όσο είναι η δημόσια διεύθυνση της βιβλιοθήκης (uint256).

Το πρόβλημα στο συγκεκριμένο contract όμως δεν σταματάει εκεί. Όπως αναφέρθηκε προηγουμένως, μέσω της συνάρτησης επιστροφής του contract FibonacciBalance, μπορεί να καλεστεί οποιαδήποτε συνάρτηση περιλαμβάνεται στη βιβλιοθήκη, συμπεριλαμβανομένης της συνάρτησης *setStart()*. Αυτή η συνάρτηση επιτρέπει την τροποποίηση της μεταβλητής στο slot[0]. Επομένως, ένας επιτιθέμενος θα μπορούσε να κατασκευάσει το παρακάτω κακόβουλο contract, να μετατρέψει την διεύθυνση της μεταβλητής σε έναν uint τύπο και μετά να καλέσει την *setStart(<attack_contract_address_as_uint>)*. Αυτό θα αλλάξει τη διεύθυνση του fibonacciLibrary στη διεύθυνση του επιτιθέμενου contract. Στη συνέχεια, οποιαδήποτε στιγμή ένας χρήστης χρησιμοποιούσε τη συνάρτηση *withdraw()* ή τη συνάρτηση επιστροφής, το κακόβουλο πρόγραμμα θα μπορούσε να εκτελεστεί επειδή έχει τροποποιηθεί η πραγματική διεύθυνση της βιβλιοθήκης.

Ένα παράδειγμα μιας τέτοιας επίθεσης θα μπορούσε να είναι το παρακάτω:

```
contract Attack {
   uint storageSlot0;
   uint storageSlot1;
   function() public {
      storageSlot1 = 0;
      <attacker_address>.transfer(this.balance);
   }
}
```

Ο επιτιθέμενος τροποποιεί την μεταβλητή calculatedFibNumber αλλάζοντας το slot[1]. Σε αυτό το contract, ένας επιτιθέμενος μπορεί να τροποποιήσει οποιοδήποτε άλλο slot θέλει για να πραγματοποιήσει διαφορετικών τύπων επιθέσεις. Είναι σημαντικό να γίνει κατανοητό ότι όταν λέμε ότι η κατάσταση παραμένει με την χρήση της delegatecall, δεν αναφερόμαστε τόσο στο όνομα των μεταβλητών, όσο στη θέση που καταλαμβάνουν στη λίστα των slots. Όπως γίνεται κατανοητό, από αυτό το απλό λάθος ένα ολόκληρο contract μπορεί να πέσει στα χέρια ενός χρήστη με κακόβουλες προθέσεις.

3.2.4.γ. Αντίμετρα

Η Solidity, παρέχει την λέξη-κλειδί library για την υλοποίηση των βιβλιοθηκών. Έτσι εξασφαλίζεται ότι η βιβλιοθήκη που θα καλεστεί δεν θα επηρεάσει την κατάσταση του contract και είναι μη-αυτοκαταστροφική. Οι βιβλιοθήκες που δε διατηρούν καμία κατάσταση (stateless), εμποδίζουν τους επιτιθέμενους που προσπαθούν να επηρεάσουν τα contracts τα οποία τις χρησιμοποιούν. Ως γενικό κανόνα μπορούμε να κρατήσουμε ότι η χρήση της DELEGATECALL πρέπει να γίνεται πολύ προσεκτικά, και όποτε είναι δυνατόν να γίνεται η χρήση βιβλιοθηκών χωρίς κατάσταση.

3.2.4.δ. Πραγματικό περιστατικό

Ένα αξιοσημείωτο περιστατικό που συνέβη στον κόσμο του Ethereum είναι η δεύτερη εκμετάλλευση του Parity MultiSig Wallet, το οποίο είναι ένα πολύ καλό παράδειγμα μιας πολύ προσεγμένης και ασφαλούς βιβλιοθήκης και της δυνατότητας να μετατραπεί σε αντικείμενο εκμετάλλευσης όταν εκτελείται σε ένα πλαίσιο που δεν έχει δημιουργηθεί για αυτή.

Εξετάζοντας την ιστοσελίδα της εφαρμογής στο GitHub (://github.com/paritytech/parity-ethereum/blob/b640df8fbb964da7538eef268dffc125b081a82f/js/src/contracts/snippets/enhanced-wallet.sol) μπορούμε να δούμε τα ευάλωτα τμήματα κώδικα τα οποία οδήγησαν στο παραπάνω περιστατικό.

Συγκεκριμένα το ενδιαφέρον τμήμα του κώδικα της βιβλιοθήκης, αφαιρώντας τα τμήματα κώδικα που δεν μας ενδιαφέρουν, είναι το εξής:

```
contract WalletLibrary is WalletEvents {

 modifier only_uninitialized { if (m_numOwners > 0) throw; _; }

  function initWallet(address[] _owners, uint _required, uint _daylimit) only_uninitialized {
    initDaylimit(_daylimit);
    initMultiowned(_owners, _required);
  }
```

```
  function kill(address _to) onlymanyowners(sha3(msg.data)) external {
    suicide(_to);
  }
}
```

Και ο κώδικας στο contract του wallet που μας ενδιαφέρει είναι ο εξής:

```
contract Wallet is WalletEvents {
  ...
  function() payable {
    if (msg.value > 0)
      Deposit(msg.sender, msg.value);
    else if (msg.data.length > 0)
      _walletLibrary.delegatecall(msg.data);
  }
  ...
  address constant _walletLibrary = 0xcafecafecafecafecafecafecafecafecafecafe;
}
```

Όπως και στο τμήμα κώδικα το οποίο χρησιμοποιήθηκε ως παράδειγμα για την κατανόηση της ευπάθειας, έτσι και στο τμήμα κώδικα του contract, όλες οι κλήσεις στη βιβλιοθήκη γίνονται μέσα από την συνάρτηση της delegatecall. Η μεταβλητή *_walletLibrary* χρησιμοποιείται για την αποθήκευση της διεύθυνσης της βιβλιοθήκης που χρησιμοποιείται. Η προβλεπόμενη λειτουργία αυτών των contract ήταν να επιτευχθεί μια χαμηλού κόστους διαδικασία ενώ η κύρια λειτουργικότητα θα περιλαμβάνεται στην βιβλιοθήκη.

Η ευπάθεια εμφανίζεται στο γεγονός ότι όλες οι κλήσεις μπορούν να γίνουν στο ίδιο το contract. Συγκεκριμένα το WalletLibrary contract μπορεί να αρχικοποιηθεί ξανά και να αλλάξει ο κάτοχος του. Και αυτό ακριβώς συνέβη. Ο επιτιθέμενος, καλώντας την συνάρτηση *initWallet()* που βρίσκεται στην βιβλιοθήκη, έγινε ο κάτοχος της και στην συνέχεια κάλεσε την συνάρτηση *kill()*. Για το λόγο που προαναφέρθηκε, ο έλεγχος για την κατοχή του contract ήταν επιτυχής και είχε σαν αποτέλεσμα την αυτοκαταστροφή της βιβλιοθήκης. Όλα τα contracts που βασίζονταν στη λειτουργικότητα της βιβλιοθήκης σταμάτησαν να λειτουργούν

και η συνολική ποσότητα ether που ήταν αποθηκευμένη σε αυτά εξαφανίστηκε ή έγινε μη προσβάσιμη στους νόμιμους κατόχους της.

3.2.5. Ψευδαίσθηση Εντροπίας (Entropy Illusion)

3.2.5.α. Περιγραφή

Όλες οι συναλλαγές που πραγματοποιούνται στην πλατφόρμα του Ethereum είναι προσδιοριστικές λειτουργίες μετάβασης μιας κατάστασης. Αυτό σημαίνει ότι κάθε συναλλαγή που πραγματοποιείται μπορεί να μεταβάλει την παγκόσμια κατάσταση του οικοσυστήματος, και το πετυχαίνει αυτό με υπολογιστική ακρίβεια και χωρίς καθόλου αβεβαιότητα. Έτσι, μπορούμε να καταλήξουμε στο συμπέρασμα ότι μέσα στο οικοσύστημα αυτό δεν υπάρχει καθόλου εντροπία ή κάποια άλλη μορφή τυχαιότητας. Οπότε, για να επιτευχθεί αποκεντρωμένη εντροπία (δηλαδή τυχαιότητα) σε κάποιο contract, οι προγραμματιστές του blockchain πρέπει να βρουν έναν διαφορετικό τρόπο υλοποίησης της.

3.2.5.β. Η ευπάθεια

Όσο το Ethereum βρισκόταν, και βρίσκεται, σε αρχικό στάδιο χωρίς να έχει κάποια συγκεκριμένη χρήση στο ευρύ κοινό, οι προγραμματιστές φτιάχνουν contracts τα οποία έχουν κάποια συσχέτιση με τον τζόγο. Για να υπάρξει όμως τζόγος θα πρέπει να υπάρξει τυχαιότητα, και αυτό από μόνο του κάνει τον προγραμματισμό μιας τέτοιας εφαρμογής αρκετά δύσκολο. Ενώ η λειτουργία στοιχημάτων μπορεί να επιτευχθεί μεταξύ δύο χρηστών της πλατφόρμας, όταν ο προγραμματιστής θέλει να προσθέσει κάποιο μηχανισμό στο contract που θα λειτουργεί ως μεσάζοντας και κριτής κάθε παιχνιδιού, όπως είναι ο κρουπιέρης σε ένα καζίνο, τότε ο προγραμματισμός δυσκολεύει πολύ. Η παγίδα στην οποία πέφτουν συχνά αρκετοί προγραμματιστές είναι η χρήση μεταβλητών από μελλοντικά blocks, όπως είναι η τιμή του κατακερματισμένου block, χρονικές σημάνσεις (timestamps), ο αριθμός του block, το βάθος του block, το όριο του gas που χρησιμοποιήθηκε και άλλες παρόμοιες μεταβλητές. Το πρόβλημα με τις παραπάνω παραμέτρους είναι ότι δεν είναι τυχαίες αλλά εξαρτώνται από τον δημιουργό και νικητή του block.

Για παράδειγμα, υποθέτουμε την ύπαρξη μιας ρουλέτας σε μορφή contract, το οποίο έχει υλοποιηθεί με τέτοιο τρόπο ώστε να επιστρέφει έναν κόκκινο αριθμό όταν το επόμενο block που θα τοποθετηθεί στην αλυσίδα, έχει σαν τελευταίο ψηφίο του hash του έναν ζυγό αριθμό. Ο δημιουργός αυτού του block, ή αλλιώς ο επιτιθέμενος, μπορεί να ποντάρει μια πολύ μεγάλη ποσότητα από ether στο γεγονός ότι ο επόμενος αριθμός που θα επιλεγεί στη ρουλέτα θα είναι κόκκινος. Αν κερδίσει το επόμενο block, και ο τελευταίος αριθμός δεν καλύπτει το κριτήριο για να κερδίσει τον γύρο, μπορεί να μη δημοσιεύσει το block και να ξεκινήσει την διαδικασία από την αρχή με την ελπίδα ότι το επόμενο block, θα πληροί την προϋπόθεση που θα τον βγάλει νικητή.

Ένα τέτοιο παράδειγμα ευπαθούς κώδικα θα μπορούσε να είναι το παρακάτω:

```
function generateRand() private returns (uint) {
    privSeed = (privSeed*3 + 1) / 2;
    privSeed = privSeed % 10**9;
    uint number = block.number;
    uint diff = block.difficulty;
    uint time = block.timestamp;
    uint gas = block.gaslimit;
    uint total = privSeed + number + diff + time + gas;
    uint rand = total % 37;
    return rand;
}
```

Όπως φαίνεται στο τμήμα αυτού του κώδικα, η ψευδοτυχαία γεννήτρια αριθμών χρησιμοποιεί συγκυριακές παραμέτρους όπως είναι η *block.number*, *block.difficulty*, *block.timestamp*, *block.gaslimit* και μια μεταβλητή από τον αποθηκευτικό χώρο του contract, *privSeed*.

Επιπλέον, η χρήση των παραμέτρων αυτών σημαίνει ότι ο ψευδοτυχαίος αριθμός που θα προκύψει από αυτή την γεννήτρια θα είναι ο ίδιος αριθμός για όλες τις συναλλαγές που υπάρχουν μέσα στο block, οπότε ο επιτιθέμενος μπορεί να πολλαπλασιάσει τα κέρδη του δημιουργώντας και τοποθετώντας πολλαπλές συναλλαγές μέσα στο ίδιο block.

3.2.5.γ. Αντίμετρα

Η πηγή της εντροπίας στα contracts πρέπει να προέρχεται από κάποιον εξωτερικό παράγοντα και να μη βασίζεται στο ίδιο οικοσύστημα του blockchain. Για να επιτευχθεί αυτό μεταξύ των χρηστών μπορεί να χρησιμοποιηθεί κάποιο σύστημα όπως είναι το commit-reveal, ή αλλάζοντας το μοντέλο εμπιστοσύνης σε μια ομάδα συμμετεχόντων.

3.2.5.δ. Πραγματικό περιστατικό

Δεν έχει υπάρξει κάποιο μεγάλο περιστατικό, μέχρι στιγμής, στο οποίο θα μπορούσε να γίνει αναφορά ή να έχει χρησιμοποιηθεί η συγκεκριμένη ευπάθεια για την ανάληψη μιας μεγάλης ποσότητας από ether σε κάποιας μορφής επίθεση. Ωστόσο, μετά από ανάλυση 3640 ενεργών contract στο blockchain του Ethereum αποκαλύφθηκε ότι τουλάχιστον 43 από αυτά χρησιμοποιούν κάποια γεννήτρια παραγωγής ψευδοτυχαίων αριθμών που μπορεί δυνητικά να αποτελέσει αντικείμενο εκμετάλλευσης με κάποιον τρόπο.

3.2.6. Μη ελεγμένες τιμές επιστροφής της συνάρτησης CALL (Unchecked CALL return values)

3.2.6.α. Περιγραφή

Υπάρχουν αρκετοί τρόποι για να πραγματοποιηθούν εξωτερικά αιτήματα στην Solidity. Για την αποστολή ποσότητας ether σε λογαριασμούς της πλατφόρμας ο πιο κοινός και ασφαλής τρόπος είναι η χρήση της συνάρτησης *transfer()*. Ωστόσο, η συνάρτηση *call()* μπορεί επίσης να χρησιμοποιηθεί σε παραπλήσια αιτήματα και στην περίπτωση που πρέπει να πραγματοποιηθούν πιο ευέλικτα εξωτερικά αιτήματα μπορεί να χρησιμοποιηθεί ο κώδικας λειτουργίας της CALL.

Οι συναρτήσεις *call()* και *send()* επιστρέφουν μια τιμή τύπου Boolean, ως αναφορά για την επιτυχία ή την αποτυχία της κλήσης. Έτσι, η χρήση τους πρέπει να γίνεται λαμβάνοντας υπόψιν το ενδεχόμενο αποτυχίας της εξωτερικής κλήσης και την επιπλέον εκτέλεση κώδικα. Σε αντίθεση με τις άλλες συναρτήσεις που πραγματοποιούν αιτήματα προς άλλα contracts ή διευθύνσεις, και έχουν την δυνατότητα να αναιρέσουν την κλήση όταν αυτή αποτύχει, οι δύο

αυτές συναρτήσεις θα επιστρέψουν false (λανθασμένη/ψευδής). Ένα κοινό πρόβλημα εμφανίζεται όταν η τιμή επιστροφής δεν ελέγχεται ενώ ο προγραμματιστής περιμένει αναίρεση της κλήσης.

3.2.6.β. Η ευπάθεια

Θεωρούμε το εξής contract:

```
contract Lotto {

  bool public payedOut = false;
  address public winner;
  uint public winAmount;

  ...

  function sendToWinner() public {
      require(!payedOut);
      winner.send(winAmount);
      payedOut = true;
  }

  function withdrawLeftOver() public {
      require(payedOut);
      msg.sender.send(this.balance);
  }
}
```

Το παραπάνω contract είναι μια παραλλαγή του Λόττο στο Ethereum, όπου ο νικητής (μεταβλητή *winner*) λαμβάνει μια ποσότητα ether (μεταβλητή *winAmount*) η οποία συνήθως αφήνει κάποια ποσότητα στο contract για τους υπόλοιπους χρήστες και μπορεί να αναληφθεί μέσω της συνάρτησης *withdrawLeftOver()*.

Όπως παρατηρείται στο παραπάνω τμήμα κώδικα, μετά την εντολή αποστολής της ποσότητας ether στο νικητή, *winner.send(winAmount)*, δεν υπάρχει κάποιος έλεγχος για την τιμή

επιστροφής της συνάρτησης *send()*. Αφού πραγματοποιηθεί η συναλλαγή και ο νικητής λάβει το ποσό που του αντιστοιχεί, εκτελείται η εντολή *payedOut = true* και δίνει τη δυνατότητα σε οποιονδήποτε άλλο χρήστη να πραγματοποιήσει ανάληψη του υπόλοιπου ποσού. Στην περίπτωση όμως που η συναλλαγή απέτυχε, είτε επειδή τελείωσε το gas προτού ολοκληρωθεί η συναλλαγή είτε επειδή έγινε προς κάποιο άλλο contact που εκ προθέσεως οδηγήθηκε στη συνάρτηση επιστροφής, η νικητήρια ποσότητα ether μπορεί να αναληφθεί από οποιονδήποτε χρήστη της πλατφόρμας μέσω της συνάρτησης *withdrawLeftOver()*, διότι ο έλεγχος θα επαληθευτεί.

3.2.6.γ. Αντίμετρα

Είναι καλή πρακτική να χρησιμοποιείται η συνάρτηση *transfer()* αντί της συνάρτησης *send()* όποτε αυτό είναι δυνατόν. Αν η συνάρτηση *send()* είναι αναγκαστικό να χρησιμοποιηθεί, πρέπει να γίνεται πάντα έλεγχος της τιμής επιστροφής της μετά από κάθε χρήση.

Ενώ η συνάρτηση transfer() θεωρείται αρκετά ασφαλής, ο επίσημος οδηγός της Solidity για τη σωστή αποστολή ether προτείνει τη χρήση ενός προτύπου ανάληψης. Με αυτό τον τρόπο, κάθε χρήστης επιβαρύνεται με το κάλεσμα μιας απομονωμένης συνάρτησης που αναλαμβάνει να στείλει μια ποσότητα ether από το contract προς αυτόν. Επομένως επιτυγχάνεται ένας ανεξάρτητος έλεγχος για κάθε αποτυχημένη συναλλαγή με κάθε χρήστη και οι συνέπειες χειρίζονται ξεχωριστά. Η ιδέα πίσω από αυτό το προτεινόμενο πρότυπο είναι η λογική απομόνωση της συναλλαγής από τον υπόλοιπο κώδικα και η τοποθέτηση του φόρτου της αποτυχημένης συναλλαγής στον χρήστη που καλεί την συνάρτηση ανάληψης.

3.2.6.δ. Πραγματικό περιστατικό

Σε αυτή την ευπάθεια αντιστοιχούν δύο μεγάλες επιθέσεις που έχουν πραγματοποιηθεί μέχρι στιγμής στην πλατφόρμα του Ethereum. Το πρώτο περιστατικό συνέβη στο contract με την ονομασία Etherpot, το οποίο είναι μια παραλλαγή της κλασικής λοταρίας και ο κώδικάς του δε διαφέρει πολύ από τον κώδικα του παραπάνω παραδείγματος. Ο βασικός λόγος που αυτό το contract δέχτηκε επίθεση ήταν ο λάθος τρόπος χρήσης των κατακερματισμένων block.

Επιπλέον όμως, το ίδιο contract εμφάνιζε και την ευπάθεια του μη ελέγχου της τιμής επιστροφής.

Εξετάζοντας ένα κομμάτι του συνολικού contract, που μπορεί να βρεθεί στο GitHub (://github.com/etherpot/contract/blob/master/app/contracts/lotto.sol), και συγκεκριμένα τη συνάρτηση *cash()*, παρατηρούμε ότι ενώ χρησιμοποιείται η συνάρτηση *send()* για την αποστολή του κερδισμένου ποσού δεν γίνεται κάποιος έλεγχος στη συνέχεια, με αποτέλεσμα ακόμα και να μην γίνει η συναλλαγή, ενώ η κατάσταση του contract έχει πλέον αλλάξει.

```
function cash(uint roundIndex, uint subpotIndex){

    var subpotsCount = getSubpotsCount(roundIndex);

    if(subpotIndex>=subpotsCount)
        return;

    var decisionBlockNumber = getDecisionBlockNumber(roundIndex,subpotIndex);

    if(decisionBlockNumber>block.number)
        return;

    if(rounds[roundIndex].isCashed[subpotIndex])
        return;

    var winner = calculateWinner(roundIndex,subpotIndex);
    var subpot = getSubpot(roundIndex);

    winner.send(subpot);

    rounds[roundIndex].isCashed[subpotIndex] = true;
}
...
```

Ένα δεύτερο και αρκετά σοβαρότερο περιστατικό, σημειώθηκε λίγο αργότερα σε ένα άλλο contract με όνομα King of the Ether.

3.2.7. Συνθήκες αγώνα (Race Conditions)

3.2.7.α. Περιγραφή

Ο συνδυασμός των εξωτερικών κλήσεων σε άλλα contracts και η πολλαπλή φύση των χρηστών της τεχνολογίας του blockchain, προκαλεί μια ποικιλομορφία από πιθανές παγίδες και ευπάθειες της Solidity, οι χρήστες της οποίας βρίσκονται σε έναν συνεχή αγώνα ενάντια στον κώδικα ώστε να δημιουργήσουν απροσδόκητες καταστάσεις. Ένα τέτοιο παράδειγμα είναι η επίθεση της επανεισόδου που εξετάσαμε σε προηγούμενο κεφάλαιο.

3.2.7.β. Η ευπάθεια

Όπως σε όλες τις πλατφόρμες που χρησιμοποιούν την τεχνολογία του blockchain, έτσι και στην πλατφόρμα του Ethereum οι κόμβοι που συμμετέχουν στη λειτουργία της ανασύρουν μη επικυρωμένες συναλλαγές από μια λίστα και δημιουργούν με αυτές τα block. Η συναλλαγή θεωρείται έγκυρη από τη στιγμή που έχει τοποθετηθεί σε κάποιο block της αλυσίδας και μετά. Η λίστα είναι κυρίως οργανωμένη με τέτοιον τρόπο ώστε οι συναλλαγές που έχουν το μεγαλύτερο *gasPrice* να βρίσκονται στην κορυφή. Και αυτό από μόνο του μπορεί να είναι αρκετό για να πραγματοποιηθεί μια επίθεση, ή να αποτελέσει το αρχικό στάδιό της. Ο επιτιθέμενος μπορεί να παρακολουθεί τη λίστα με τις συναλλαγές που περιμένουν επικύρωση μέχρι να βρει την κατάλληλη, η οποία μπορεί να περιέχει κάποια λύση σε κάποιο πρόβλημα, να τροποποιήσει ή να ανακαλέσει τα δικαιώματά του ή να αλλάξει την κατάσταση του contract που είναι ανεπιθύμητο σε αυτόν. Στη συνέχεια, μπορεί να πάρει τα δεδομένα από αυτή τη συναλλαγή και να δημιουργήσει μια δική του με μεγαλύτερο κόστος *gasPrice* με αποτέλεσμα να καταφέρει να ταξινομηθεί αυτή η συναλλαγή πριν από την πρωτότυπη.

Για να γίνει κατανοητό αυτό μπορούμε να εξετάσουμε το παρακάτω contract:

```
contract FindThisHash {
   bytes32 constant public hash =
0xb5b5b97fafd9855eec9b41f74dfb6c38f5951141f9a3ecd7f44d5479b630ee0a;

   constructor() public payable {}

   function solve(string solution) public {
      require(hash == sha3(solution));
      msg.sender.transfer(1000 ether);
   }
}
```

Σε αυτό το contract οι συμμετέχοντες ανταγωνίζονται ο ένας τον άλλον για να βρουν ποια είναι η αρχική μορφή της κατακερματισμένης έκφρασης *`0xb5b5b97fafd9855eec9b41f74dfb6c38f5951141f9a3ecd7f44d5479b630ee0a`* χρησιμοποιώντας τον αλγόριθμο SHA3(). Το contract αυτό περιέχει 500 Ether τα οποία θα λάβει ο νικητής. Ας υποθέσουμε ότι η σωστή φράση που πρέπει να δώσει ο χρήστης είναι `Ethereum!`. Καλώντας την συνάρτηση *solve()* και χρησιμοποιώντας οποιαδήποτε φράση σαν παράμετρο, ο χρήστης μπορεί να πάρει μέρος σε αυτό το παιχνίδι.

Ο επιτιθέμενος παρατηρεί τη λίστα με τις συναλλαγές και περιμένει έως ότου βρει κάποια που να περιέχει την απάντηση. Όταν εντοπίσει τη συναλλαγή, ελέγχει την εγκυρότητα της και στη συνέχεια κατασκευάζει μια δική του, με το ίδιο περιεχόμενο αλλά πληρώνοντας μια αρκετά μεγαλύτερη ποσότητα Ether ως *gasPrice*. Σε αυτό το σημείο, είναι πιθανό ο χρήστης που θα κατασκευάσει το επόμενο block να εισάγει στις συναλλαγές τη συναλλαγή του επιτιθέμενου από τη στιγμή που θα βρίσκεται πιο ψηλά στη λίστα. Ο επιτιθέμενος θα λάβει τα 500 Ether που ορίζει το contract και ο χρήστης που είχε την αρχική επίλυση του προβλήματος δε θα λάβει καμία ποσότητα.

Ένα πιο ρεαλιστικό πρόβλημα αυτού του τύπου επιθέσεων προβλέπεται να αντιμετωπιστεί με την αναδιαμόρφωση του Ethereum που έχει ήδη ξεκινήσει και προβλέπεται να έχει ολοκληρωθεί μέχρι το 2020.

3.2.7.γ. Αντίμετρα

Υπάρχουν δύο διαφορετικοί τύποι χρηστών που μπορεί να πραγματοποιήσουν μια τέτοια επίθεση. Στην πρώτη κατηγορία έχουμε τους χρήστες οι οποίοι μπορούν να τροποποιήσουν την ποσότητα του ether που χρησιμοποιείται ως *gasPrice* στις συναλλαγές που πραγματοποιούν και στη δεύτερη κατηγορία έχουμε τους miners οι οποίοι μπορούν να αλλάξουν τη σειρά των συναλλαγών που τοποθετούνται στο block όπως αυτοί θέλουν.

Ένα contract που είναι ευπαθές στην πρώτη κατηγορία χρηστών είναι σε σημαντικά χειρότερη θέση από αυτό που βρίσκεται στο στόχο της δεύτερης κατηγορίας χρηστών. Αυτό συμβαίνει γιατί ο miner μπορεί να υλοποιήσει μια τέτοια επίθεση μόνο όταν κατασκευάσει ένα block, το οποίο είναι πολύ απίθανο να συμβεί για κάθε μεμονωμένο miner που στοχεύει ένα συγκεκριμένο block.

Ένα αντίμετρο το οποίο μπορεί να χρησιμοποιηθεί είναι η τοποθέτηση μιας λογικής τιμής μέσα στο contract που ορίζει ποια είναι η μέγιστη ποσότητα ether ως *gasPrice* που μπορεί να χρησιμοποιηθεί σε κάθε συναλλαγή. Με αυτόν τον τρόπο ο επιτιθέμενος δεν μπορεί να αυξήσει απεριόριστα το gasPrice και να μπει πρώτος στην λίστα με τις υπόλοιπες συναλλαγές. Αυτό το αντίμετρο λειτουργεί στην πρώτη κατηγορία χρηστών, δηλαδή τους αυθαίρετους χρήστες του συστήματος. Οι χρήστες της δεύτερης κατηγορίας μπορούν να συνεχίσουν να πραγματοποιούν τέτοιου είδους επιθέσεις, διότι μπορούν να ελέγχουν την σειρά των συναλλαγών που εγκρίνονται όπως εκείνοι θέλουν.

Ενναλακτικά, μια πιο αξιόπιστη μέθοδος αντιμετώπισης τέτοιων επιθέσεων είναι η χρήση του συστήματος commit-reveal που έχει αναφερθεί σε προηγούμενη παράγραφο. Ένα τέτοιο σύστημα, όταν υλοποιηθεί σωστά, επιτρέπει στους χρήστες να κάνουν συναλλαγές οι οποίες περιλαμβάνουν κρυμμένες πληροφορίες. Αφού η συναλλαγή τοποθετηθεί σε ένα block, τότε ο χρήστης στέλνει μια επιπλέον συναλλαγή που συμπεριλαμβάνει, φανερά αυτή την φορά, την πληροφορία που είχε κρύψει στην αρχική. Αυτή η μέθοδος αποτρέπει και τους δύο τύπους χρηστών από το να προτρέχουν σε συναλλαγές από την στιγμή που δεν ξέρουν το περιεχόμενο τους. Αυτή η μέθοδος βέβαια, δεν μπορεί να αποκρύψει την τιμή της συναλλαγής που τις περισσότερες φορές είναι μια πολύ σημαντική πληροφορία η οποία θα έπρεπε να παραμένει κρυφή.

Βασισμένο σε αυτή την λογική είναι το ENS smart contract. Στην πρώτη φάση ο χρήστης είναι ελεύθερος να στείλει μια αυθαίρετη ποσότητα από ether, την οποία πρέπει να έχει στην κατοχή του, στην συναλλαγή που πραγματοποιεί, και στη δεύτερη φάση αποκαλύπτει την ποσότητα που ήταν πρόθυμος να χρησιμοποιήσει. Η διαφορά μεταξύ αυτών των δύο ποσών επιστρέφεται στον χρήστη.

Μια ακόμη πρόταση είναι να χρησιμοποιείται μια υποβρύχια αποστολή (Submarine Sends). Η αποτελεσματική υλοποίηση αυτής της ιδέας απαιτεί τη χρήση κωδικού λειτουργίας της CREATE2, που μέχρι στιγμής δεν έχει υιοθετηθεί αλλά είναι πιθανό αυτό να συμβεί σε κάποια μελλοντική έκδοση της πλατφόρμας.

3.2.7.δ. Πραγματικό περιστατικό

Ένα περιστατικό τέτοιας επίθεσης σημειώθηκε στο πρότυπο ERC20 το οποίο είναι αρκετά γνωστό για τη δημιουργία διακριτικών στο Ethereum. Αυτό το πρότυπο έχει μια πιθανή ευπάθεια τέτοιου τύπου, η οποία προέρχεται από τη συνάρτηση *approve()*.

Η συνάρτηση *approve()* συντάσσεται ως εξής:

```
function approve(address _spender, uint256 _value) returns (bool success)
```

Αυτή η συνάρτηση δίνει το δικαίωμα σε κάποιον χρήστη της πλατφόρμας να μεταφέρει token για λογαριασμό κάποιου άλλου χρήστη. Το πιθανό σενάριο εκμετάλλευσης αυτής της ευπάθειας εμφανίζεται όταν ένας χρήστης, ας πούμε η Έλλη, επιτρέψει σε κάποιον άλλον χρήστη, ας πούμε στον Κώστα, να χρησιμοποιήσει 100 token. Η προαναφερόμενη, στη συνέχεια αποφασίζει ότι θέλει να αναιρέσει την άδεια που έδωσε προηγουμένως στον Κώστα, οπότε δημιουργεί ακόμη μια συναλλαγή για να αλλάξει την επιτρεπόμενη ποσότητα από 100 σε 50. Ο Κώστας, ως επιτιθέμενος πλέον, παρατηρεί τις συναλλαγές που τοποθετούνται στη λίστα και βλέπει την αλλαγή που ζήτησε να κάνει η Έλλη. Έτσι, δημιουργεί μια καινούργια συναλλαγή όπου ξοδεύει τα 100 token, και βάζει μια πιο υψηλή τιμή στο gasPrice από ότι η Έλλη στη δική της. Κάποιες υλοποιήσεις της παραπάνω συνάρτησης επιτρέπουν στον Κώστα να στείλει αυτά τα 100 token και όταν η συναλλαγή της Έλλης γίνει αποδεκτή, επαναφέρει το

αποδεκτό ποσό του Κώστα στα 50 token, ενώ έχει ως αποτέλεσμα να του δώσει πρόσβαση σε συνολικά 150 token.

Ένα δεύτερο περιστατικό που εκμεταλλεύτηκε την ευπάθεια race condition σημειώθηκε στο δίκτυο Bancor.

3.2.8. Άρνηση υπηρεσίας (Denial Of Service)

3.2.8.α. Περιγραφή

Η κατηγορία αυτή είναι πολύ γενική, αλλά περιλαμβάνει διάφορους τρόπους και επιθέσεις, που αφήνουν το contract σε τέτοια κατάσταση που δεν μπορεί να θεωρηθεί πλέον λειτουργικό, είτε για ένα μικρό χρονικό διάστημα είτε για πάντα. Τέτοιου είδους επιθέσεις, έχουν σαν αποτέλεσμα τον εγκλωβισμό μιας ποσότητας ether μέσα στο contract η οποία σταματάει να είναι προσβάσιμη.

3.2.8.β. Η ευπάθεια

Υπάρχουν αρκετοί τρόποι που μπορεί να οδηγήσουν ένα contract στο να καταλήξει μη λειτουργικό. Μερικά λάθη τα οποία μπορεί να οδηγήσουν σε μια τέτοια κατάσταση και δεν είναι εμφανή με την πρώτη ματιά είναι τα εξής:

Επανάληψη μέσω εξωτερικά χειραγωγούμενων αντιστοιχιών ή πινάκων

Αυτή η μορφή ευπάθειας παρουσιάζεται κυρίως στα σενάρια όπου ο κάτοχος του contract θέλει να μοιράσει μια ποσότητα από token σε επενδυτές, και για να το πραγματοποιήσει αυτό κατασκευάζει μια συνάρτηση όπως είναι η distribute() στο παράδειγμα που ακολουθεί.

```
contract DistributeTokens {
    address public owner;
    address[] investors;
    uint[] investorTokens;

    ...

    function invest() public payable {
        investors.push(msg.sender);
        investorTokens.push(msg.value * 5);
        }

    function distribute() public {
        require(msg.sender == owner);
        for(uint i = 0; i < investors.length; i++) {
            transferToken(investors[i],investorTokens[i]);
        }
    }
}
```

Το πρόβλημα στο παραπάνω smart contract υπάρχει στην επανάληψη η οποία γίνεται για κάθε μέλος της λίστας *investors*. Η λίστα αυτή μπορεί να διογκωθεί τεχνητά, αν ο επιτιθέμενος φτιάξει αρκετούς λογαριασμούς και τους προσθέσει σε αυτή. Αυτό θα έχει σαν συνέπεια το gas που απαιτείται για να εκτελεστεί η επανάληψη να είναι περισσότερο από το gas limit της συναλλαγής με αποτέλεσμα η συνάρτηση distribute() να μην μπορεί να εκτελεστεί ποτέ.

Λειτουργίες ιδιοκτητών

Ένα ακόμα κοινό μοτίβο που παρατηρείται σε διάφορα contract είναι οι ιδιοκτήτες να έχουν συγκεκριμένα προνόμια και να πρέπει να πραγματοποιήσουν συγκεκριμένες ενέργειες ώστε να προχωρήσει το contract στο επόμενο στάδιο εκτέλεσης. Ένα τέτοιο παράδειγμα είναι το παρακάτω contract ICO το οποίο απαιτεί από τον δημιουργό του να εκτελέσει τη συνάρτηση *finalize()* ώστε να επιτραπεί η μεταφορά των token.

```
bool public isFinalized = false;
address public owner;

function finalize() public {
   require(msg.sender == owner);
   isFinalized == true;
}

...

function transfer(address _to, uint _value) returns (bool) {
   require(isFinalized);
   super.transfer(_to,_value)
}

...
```

Σε αυτή την περίπτωση, αν οι προνομιακοί χρήστες χάσουν τα ιδιωτικά τους κλειδιά ή σταματήσουν να είναι ενεργοί, σταματάει και η λειτουργία του contract. Ο ιδιοκτήτης δεν μπορεί πλέον να καλέσει τη συνάρτηση *finalize()* οπότε και τα token δεν μπορούν να μεταφερθούν.

Κατάσταση προόδου βασισμένη σε εξωτερικές κλήσεις

Άλλο ένα κοινό μοτίβο στην συγγραφή των smart contract είναι η αποστολή ether σε κάποια εξωτερική διεύθυνση ή η παύση της λειτουργίας του contract μέχρι να λάβει κάποια είσοδο από κάποια εξωτερική πηγή, ώστε να συνεχιστεί η λειτουργία του. Αυτές οι δύο τακτικές μπορούν να οδηγήσουν σε άρνηση υπηρεσίας, όταν το εξωτερικό κάλεσμα αποτύχει ή εμποδιστεί από κάποιον εξωτερικό παράγοντα. Για παράδειγμα, στην περίπτωση αποστολής ether, ο επιτιθέμενος μπορεί να κατασκευάσει ένα contract το οποίο δε δέχεται μεταφορές ποσότητας ether. Αν το αρχικό contract απαιτεί την ανάληψη ether για να προχωρήσει την εκτέλεση σε μια καινούργια κατάσταση, το contract δε θα φτάσει ποτέ σε αυτή διότι δεν μπορεί να γίνει αποστολή ether σε μια διεύθυνση που δεν έχει γίνει αποδεκτή.

3.2.8.γ. Αντίμετρα

Αντίμετρο για το παράδειγμα της πρώτης κατηγορίας είναι η αποφυγή επανάληψης σε δομές δεδομένων που μπορούν να ελεγχθούν από εξωτερικούς χρήστες. Το πρότυπο ανάληψης που έχει αναλυθεί σε προηγούμενη παράγραφο συνιστάται για τέτοιες λειτουργίες, όπου ο κάθε χρήστης διεκδικεί τα token του ανεξάρτητα.

Στο δεύτερο παράδειγμα, όπου κάποιος χρήστης με συγκεκριμένα προνόμια πρέπει να χρησιμοποιήσει μέρος του contract για να γίνει η αλλαγή της κατάστασής του, πρέπει να χρησιμοποιηθεί μια δικλίδα ασφαλείας για την περίπτωση όπου ο ιδιοκτήτης καταστεί ανίκανος να εκτελέσει τα καθήκοντά του. Μια προτεινόμενη λύση είναι να ρυθμιστεί ως ιδιοκτήτης ένα multisig contract, ώστε διαφορετικά ιδιωτικά κλειδιά να οδηγούν στη σωστή λειτουργία του. Άλλη λύση είναι η χρήση ενός μηχανισμού χρονοδιακόπτη, όπου εάν ο προνομιακός χρήστης δεν πραγματοποιήσει τις απαιτούμενες ενέργειες μέσα σε ένα ορισμένο χρονικό περιθώριο, οποιοσδήποτε άλλος χρήστης θα μπορέσει να τις πραγματοποιήσει στην συνέχεια. Ο ίδιος μηχανισμός θα μπορούσε να εφαρμοστεί και στο τρίτο παράδειγμα, τοποθετώντας ένα επιτρεπτό χρονικό διάστημα πριν αλλάξει η κατάσταση του contract.

3.2.8.δ. Πραγματικό περιστατικό

Πριν από δύο χρόνια εμφανίστηκε στο Ethereum μια εταιρεία πυραμίδα που κατάφερε να συγκεντρώσει μια αξιοπρεπή ποσότητα Ether σε σύντομο χρονικό διάστημα. Δυστυχώς, το contract το οποίο χρησιμοποιήθηκε ήταν ευπαθές στην πρώτη μορφή επίθεσης άρνησης λειτουργίας που αναλύθηκε σε αυτό το κεφάλαιο. Για να γίνει η ανάληψη του ποσού που αντιστοιχούσε σε 1100 ether, απαιτούσε τη διαγραφή μιας μεγάλης αντιστοίχισης. Για να πραγματοποιηθεί όμως αυτή η διαγραφή, έπρεπε να πληρωθεί ένα κόστος gas το οποίο ξεπερνούσε το όριο που μπορούσε να χρησιμοποιήσει κάποιος τότε, και έτσι δεν ήταν δυνατόν να πραγματοποιηθεί η ανάληψη αυτής της ποσότητας.

Fun fact
Όταν το όριο του gas άλλαξε, ένας χρήστης πλήρωσε 2.5 εκατομμύρια σε gas ώστε να λάβει 1100 ether.
Η συναλλαγή μπορεί να βρεθεί στον παρακάτω σύνδεσμο, με κωδικό συναλλαγής:
0x0d80d67202bd9cb6773df8dd2020e7190a1b0793e8ec4fc105257e8128f0506b.

4. Εργαλεία και συμβουλές

Επιπρόσθετα στα αντίμετρα τα οποία αναφέραμε στις προηγούμενες παραγράφους, έχουν δημιουργηθεί κάποια εργαλεία τα οποία μπορούν να εντοπίσουν πιθανές ευπάθειες αναλύοντας τον κώδικα του smart contract. Αυτά τα προγράμματα έχουν κατασκευαστεί είτε από ομάδες ανεξάρτητων προγραμματιστών οι οποίες υποστηρίζουν τον ανοιχτό κώδικα και παρέχουν τα εργαλεία αυτά δίχως κόστος, είτε από μεγάλες εταιρίες με το αντίστοιχο αντάλλαγμα, τα οποία μπορούν να φανούν πολύ χρήσιμα σε προγραμματιστές με μεγάλη εμπειρία αλλά και σε προγραμματιστές που κάνουν τα πρώτα τους βήματα σε αυτόν τον χώρο.

Μερικά από τα προγράμματα που αξίζει να αναφερθούν σε αυτή την ενότητα είναι:

- *Slither* (https://github.com/trailofbits/slither): Πλαίσιο (framework) το οποίο πραγματοποιεί στατική ανάλυση στον κώδικα του smart contract και κατασκευάζει αναφορές.
- *Echidna* (https://github.com/trailofbits/echidna): Βιβλιοθήκη η οποία κάνει ανάλυση και δοκιμές στον κώδικα που τρέχει στο EVM.
- *Manticore* (https://github.com/trailofbits/manticore): Εργαλείο το οποίο προσομοιώνει επιθέσεις σε smart contracts και δημιουργεί αναφορές με τις ευπάθειες που εντόπισε.
- *Ethersplay* (https://github.com/trailofbits/ethersplay): Αποκωδικοποιητής του δυαδικού κώδικα που εκτελείται στο EVM και ανάλυση του.
- *IDA – EVM* (https://github.com/trailofbits/ida-evm): Επιπρόσθετο εργαλείο ανάλυσης και αποκωδικοποίησης smart contract χωρίς να γνωρίζουμε τον πηγαίο κώδικα του, το οποίο τρέχει σαν επέκταση στο πολύ γνωστό εργαλείο αποκωδικοποίησης IDA Pro.
- *Rattle* (https://securityboulevard.com/2018/09/rattle-an-ethereum-evm-binary-analysis-framework/): Πλαίσιο το οποίο πραγματοποιεί στατική ανάλυση στον δυαδικό κώδικα των smart contracts που εκτελούνται στο Virtual Machine του Ethereum.

Επιπλέον, αξίζει να αναφερθούν άλλες δύο πολύ χρήσιμες συλλογές δεδομένων:

- *(Not So) Smart Contracts* (https://github.com/trailofbits/not-so-smart-contracts): Περιέχει μια συλλογή από τις πιο κοινές ευπάθειες και επιθέσεις που μπορεί να έρθει αντιμέτωπο ένα smart contract

- *EVM Opcode Database* (https://github.com/trailofbits/evm-opcodes): Περιέχει μια συλλογή από τα opcodes που μπορεί να βρει κάποιος αναζητώντας την δυαδική μορφή ενός contract.

Με την χρήση αυτών των εργαλείων οι προγραμματιστές μπορούν να κρίνουν με μεγάλο ποσοστό επιτυχίας αν το πρόγραμμα που έχουν φτιάξει περιέχει κάποιου είδους ευπάθεια και να προσπαθήσουν να τροποποιήσουν τον κώδικα τους, ή στην περίπτωση που αυτό δεν γίνεται, να εφαρμόσουν τα αντίστοιχα αντίμετρα τα οποία προτείναμε.

5. Τα επόμενα βήματα

Το Ethereum βρίσκεται σε συνεχή εξέλιξη και μέχρι το 2020 θα έχουν πραγματοποιηθεί δραστικές αλλαγές στις βασικές λειτουργίες του. Μία από αυτές τις βασικές αλλαγές είναι η μεταφορά του συστήματος από Proof of Work σε ένα υβριδικό σύστημα Proof of Work και Proof of Stake (PoS), το οποίο ονομάζεται Casper. Αυτό περιλαμβάνει την διάσπαση της κεντρικής αλυσίδας σε μικρότερες αλυσίδες οι οποίες θα δουλεύουν ανεξάρτητα η μία από την άλλη, αλλά στο τέλος θα τροφοδοτούν όλες μαζί τα block της κεντρικής αλυσίδας.

Η αρχική πρόταση, η οποία είχε γίνει το 2015, ήταν γύρω από ένα σύστημα το οποίο θα χρησιμοποιούσε μόνο το PoS σύστημα και θα είχε τον εξής τρόπο λειτουργίας:

- Δημιουργείται μια λίστα από χρήστες που έχουν στην κατοχή τους κάποια ποσότητα από κρυπτονομίσματα. Στην συνέχεια μια ψευδοτυχαία γεννήτρια αριθμών διαλέγει έναν χρήστη από αυτούς και του επιτρέπει να φτιάξει ένα έγκυρο block μέσα σε ένα μικρό χρονικό διάστημα. Η διαδικασία επαναλαμβάνεται μέχρις ότου όλοι οι χρήστες να έχουν προτείνει ένα block.

- Επίσης, υπάρχει ένα σύνολο από χρήστες οι οποίοι θα μπορούν να επαληθεύσουν την εγκυρότητα αυτού του block, στοιχηματίζοντας ποιο block θα υπερτερήσει.

- Οι χρήστες που πραγματοποιούν την επικύρωση μπορούν να τοποθετήσουν ή να αποσύρουν μια ποσότητα από τα κρυπτονομίσματα που έχουν στην κατοχή τους σε όποιο block επιθυμούν ώστε να επιτύχουν την καλύτερη δυνατή επιστροφή κερδών.

- Μετά από αρκετούς στοιχηματικούς γύρους, και καθώς η πιθανότητα του block να δημιουργηθεί πλησιάζει το 1, ή διαφορετικά το 99%, το block θεωρείται τελικό και θα τοποθετηθεί στην αλυσίδα.

Στο παραπάνω σύστημα, για να πραγματοποιηθεί ένα στοίχημα θα πρέπει να ακολουθεί τους παρακάτω κανόνες:

- Αν το block δεν έχει παρουσιαστεί, και το χρονικό περιθώριο δημιουργίας φτάνει στο τέλος του, το κέρδος είναι ίσο με 0.5.

- Αν το block δεν έχει παρουσιαστεί, και το χρονικό περιθώριο έχει ξεπεραστεί κατά πολύ, το κέρδος θα είναι ίσο με 0.3.

- Αν το block έχει δημιουργηθεί πολύ κοντά στην λήξη του χρονικό περιθωρίου του, τότε το κέρδος ισοδυναμεί με 0.7.

- Αν το block είναι παρόν αλλά δημιουργήθηκε πάρα πολύ νωρίς ή πάρα πολύ αργά το κέρδος του είναι ίσο με 0.3.

Αυτή η πρόταση δεν έγινε τελικά αποδεκτή επειδή πολλοί παράμετροι και πτυχές αυτής της υλοποίησης ήταν ελλιπείς. Έτσι συνέχισε η μελέτη για το επόμενο σύστημα το οποίο θα έλυνε κάποια προβλήματα που παρουσιάζονται στην τρέχουσα πλατφόρμα, ενώ παράλληλα θα την έκανε πιο ασφαλή σε επιθέσεις.

Αυτή η λύση ήρθε με την πιο πρόσφατη πρόταση του Casper, η οποία βασίζεται σε μια διαφορετική φιλοσοφία σε σχέση με τις προηγούμενες PoS προτάσεις και υλοποιήσεις που έχουν γίνει σε διάφορες πλατφόρμες. Είναι βασισμένη στο θεώρημα του Lampert (Lampert's timestamps), το οποίο δημοσιεύθηκε στη δεκαετία του 1980, και αναλύει το πως ένα τέτοιο κατανεμημένο σύστημα μπορεί να δουλέψει σωστά, αν και μόνο αν, τα δύο τρίτα των συνολικών χρηστών του είναι έμπιστα. Έτσι η έκδοση αυτή καταλήγει σε ένα σύστημα το οποίο δεν χρησιμοποίει το PoS για τη δημιουργία των block ή για το χρονοδιάγραμμα τους, αλλά το PoW σύστημα όπως συμβαίνει μέχρι σήμερα. Το PoS σύστημα θα χρησιμοποιείται ως μια διαδικασία ελέγχου.

Το σύστημα αυτό δουλεύει ως εξής:

Το PoS σύστημα θα χρησιμοποιείται μόνο ανά 100 block, ώστε να παρέχει ένα παραπάνω επίπεδο ασφάλειας από το PoW, λειτουργώντας ως σύστημα ελέγχου.

- Οι συμμετέχοντες στη διαδικασία που ορίζεται στο PoS στέλνουν ποσότητες Ether στο χώρο συγκέντρωσης και επικύρωσης.

- Κάθε 100 block οι χρήστες που συμμετέχουν στη διαδικασία επικύρωσης τοποθετούν το μερίδιο τους σε ένα block ελέγχου, το οποίο περιέχει κάποια αναφορά στο

προηγούμενο block ελέγχου. Αν τα δύο τρίτα των χρηστών συμφωνήσουν σε μια πρόταση, τότε η ύπαρξη του block θεωρείται “δικαιολογημένη”.

- Όταν το block αξιολογηθεί ως δικαιολογημένο, μπορεί να χρησιμοποιηθεί ως παραπομπή σε μελλοντικούς ψήφους. Όταν τα δύο τρίτα ψηφίσουν γι αυτό το block, τότε το block θεωρείται οριστικοποιημένο και αυτό λαμβάνει προτεραιότητα στο PoW.

- Μετά την τοποθέτηση του τελευταίου block ελέγχου οι ψήφοι των χρηστών-ελεγκτών επιδέχονται μόνο 12 επιβεβαιώσεις.

- Αν τα δύο τρίτα των χρηστών που συμμετέχουν δε λάβουν μέρος στη διαδικασία ψηφοφορίας, η αλυσίδα συνεχίζει να εξελίσσεται εξολοκλήρου βάσει του PoW σύστημα.

- Αν οι κάτοχοι Ether πραγματοποιήσουν κάποια από τις παρακάτω απαγορευμένες ενέργειες, με αντάλλαγμα μια μικρή ποσότητα της τάξης του 4%, ένας τρίτος παράγοντας μπορεί να υποβάλει μια απόδειξη της παρατυπίας με αποτέλεσμα ο κακόβουλος χρήστης να χάσει ολόκληρη την ποσότητα κρυπτονομίσματος που έχει στην κατοχή του:

    1. Αν ο χρήστης ψηφίσει πολλαπλά αντικρουόμενα block τα οποία προκειται να τοποθετηθούν στο ίδιο ύψος

    2. Αν ο χρήστης ψηφίσει πολλαπλά αντικρουόμενα block τα οποία πρόκειται να τοποθετηθούν σε διαφορετικό ύψος αλλά χρησιμοποιούν αντικρουόμενα block αναφοράς, με εξαίρεση την περίπτωση που το νέο block αναφοράς έχει τοποθετηθεί σε μια από τις αλυσίδες μεταγενέστερα της ψηφοφορίας.

Οι ανταμοιβές της δομής θα αξιολογηθούν και θα τροποποιηθούν κατάλληλα, ώστε οι χρήστες-ελεγκτές του PoS να λαμβάνουν επίσης ένα μερίδιο από αυτές, επιπρόσθετα από τους εξορύκτες PoW.

Το σύστημα αυτό καταλήγει να χρησιμοποιεί και τα δύο συστήματα PoS και PoW. Ωστόσο, το PoW σύστημα χρησιμοποιείται για την παραγωγή των block και για την κρίσιμη ιδιότητα

της διασφάλισης της σωστής σύγκλισης σε μια αλυσίδα. Παρόλο που το σύστημα PoS μπορεί να περιορίσει κάποιους από τους κινδύνους, είναι αβέβαιο κατά πόσο θα επηρεάσει τη λειτουργία σύγκλισης και τη συνολική ασφάλεια του συστήματος.

6. Συμπεράσματα

Κατά τη διάρκεια της έρευνας και συγγραφής της παρούσης διπλωματικής εργασίας συμπεράναμε ότι το blockchain αποτελεί μια νέα τεχνολογία η οποία μπορεί να χρησιμοποιηθεί με διαφορετικούς τρόπους και να εξυπηρετήσει διαφορετικούς σκοπούς. Ωστόσο, μέχρι αυτή τη στιγμή δεν έχει αξιοποιηθεί σε τέτοιο βαθμό ώστε να θεωρηθεί ηγέτης στον τομέα του. Στην αρχή της εργασίας αυτής εντοπίσαμε και αναλύσαμε επιθέσεις που μπορούν να πραγματοποιηθούν στην πλατφόρμα ώστε να οδηγήσουν σε μερική ή ολική χειραγώγησή της. Η πραγματοποίηση αυτών των επιθέσεων στις διάφορες πλατφόρμες οδηγεί στο συμπέρασμα για ακόμη μια φορά ότι η θεωρητική ανάλυση διαφέρει από την τελική υλοποίηση της κάθε τεχνολογίας. Επιθέσεις όπως η 51% εκμεταλλεύονται ανθρώπινα προγραμματιστικά λάθη και την λανθασμένη υλοποίηση πρωτοκόλλων.

Στη συνέχεια στρέψαμε την προσοχή μας περισσότερο στην πλατφόρμα του Ethereum που είναι η πρώτη ανοιχτή πλατφόρμα της τεχνολογίας blockchain, η οποία επιτρέπει σε προγραμματιστές από όλο τον κόσμο να κατασκευάζουν τα δικά τους προγράμματα και να τα δημοσιεύουν σε αυτήν, χωρίς να χρειάζεται να καταβάλουν κάποιο ιδιαίτερο κόστος. Αυτές οι εφαρμογές ονομάζονται smart contracts και ορίζονται ως εφαρμογές που "*λειτουργούν πάντα όπως έχουν φτιαχτεί να λειτουργούν"*. Αυτός ο ορισμός μαζί με το όνομα έξυπνα συμβόλαια θα μπορούσε να οδηγήσει κάποιον στο συμπέρασμα ότι αυτές οι εφαρμογές είναι αρκετά ασφαλείς και δύσκολα μπορούν να αποτελέσουν αντικείμενο εκμετάλλευσης από κακόβουλους χρήστες. Δυστυχώς, η έρευνά μας κατέληξε στο αντίθετο συμπέρασμα. Η αλήθεια είναι ότι τα έξυπνα συμβόλαια είναι έξυπνα και ασφαλή μόνο στο βαθμό ο δημιουργός τους κατάφερε να τα συντάξει.

Επίσης, έγινε φανερό ότι μεγάλο ποσοστό των ενεργών και δημοσιευμένων προγραμμάτων έχουν εγγενή συσχέτιση με τον τζόγο και άλλα τυχερά παιχνίδια, όπως καζίνο, ρουλέτες, ιππόδρομο, αλλά και εφαρμογές που λειτουργούν ως τραπεζικές θυρίδες.

Το blockchain είναι μια ενδιαφέρουσα τεχνολογία, αλλά η χρήση της μέχρι αυτή την στιγμή περιορίζεται σε συγκεκριμένους τομείς χωρίς ξεκάθαρη στοχοθεσία. Τα επόμενα χρόνια θα

αποδείξουν αν αυτή η τεχνολογία και οι πλατφόρμες που βασίζονται σε αυτή θα μπορέσουν να προσφέρουν κάτι πραγματικά χρήσιμο στην ανθρωπότητα.

Βιβλιογραφικές αναφορές

Atzei N., Bartoletti M., και Cimoli T. (2017) ***'A Survey of Attacks on Ethereum Smart Contracts'*** (SoK). In: Maffei M., Ryan M. (eds) Principles of Security and Trust. POST 2017. Lecture Notes in Computer Science, vol 10204. Springer, Berlin, Heidelberg

Dr. Gavin Wood, (2018) ***'Ethereum: A Secure Decentralised Generalised Transaction Ledger - Byzantium Version'***, https://ethereum.github.io/yellowpaper/paper.pdf, τελευταία επίσκεψη: 08/08/2018

Yuval Marcus, Ethan Heilman και Sharon Goldberg, (2018) ***'Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network'***, http://www.cs.bu.edu/~goldbe/projects/eclipseEth.pdf, τελευταία επίσκεψη: 20/09/2018

Anthony Akentiev, (2017) 'Parity Multisig Hacked. Again', https://medium.com/chain-cloud-company-blog/parity-multisig-hack-again-b46771eaa838, τελευταία επίσκεψη: 10/08/2018

Arseny Reutov, (2018) 'Predicting Random Numbers in Ethereum Smart Contracts', https://blog.positive.com/predicting-random-numbers-in-ethereum-smart-contracts-e5358c6b8620, τελευταία επίσκεψη: 10/10/2018

Dr Adrian Manning, (2018) 'Solidity Security: Comprehensive list of known attack vectors and common anti-patterns', https://blog.sigmaprime.io/solidity-security.html, τελευταία επίσκεψη: 22/10/2018

BITMEX Research, (2018) 'Complete guide to Proof of Stake – Ethereum's latest proposal', https://blog.bitmex.com/complete-guide-to-proof-of-stake-ethereums-latest-proposal-vitalik-buterin-interview/?vtk, τελευταία επίσκεψη: 24/09/2018

Cem Paya, (2018) 'Ethereum, Solidity and integer overflows: programming blockchains like 1970', https://randomoracle.wordpress.com/2018/04/27/ethereum-solidity-and-integer-overflows-programming-blockchains-like-1970/, τελευταία επίσκεψη: 04/09/2018

Dave Appleton, (2017) 'Ethereum Multi-Signature Wallets.', https://medium.com/hellogold/ethereum-multi-signature-wallets-77ab926ab63b, τελευταία επίσκεψη: 02/10/2018

Eric Banisadr, (2018) 'How $800K evaporated from the PoWH coin Ponzi Scheme overnight', https://blog.goodaudience.com/how-800k-evaporated-from-the-powh-coin-ponzi-scheme-overnight-1b025c33b530, τελευταία επίσκεψη: 14/08/2018

Hassed Quresh, (2017) 'A hacker stole $31M of Ether', https://medium.freecodecamp.org/a-hacker-stole-31m-of-ether-how-it-happened-and-what-it-means-for-ethereum-9e5dc29e33ce, τελεταία επίσκεψη: 12/08/2018

Ivan Bogatyy, (2018) 'Implementing Ethereum trading front-runs on the Bancor exchange in Python', https://hackernoon.com/front-running-bancor-in-150-lines-of-python-with-ethereum-api-d5e2bfd0d798, τελευταία επίσκεψη: 19/08/2018

Jimi S., (2018) 'Blockchain: how a 51% attack works (double spend attack)', https://medium.com/@JimiS/what-is-a-51-attack-or-double-spend-attack-aa108db63474, τελευταία επίσκεψη: 29/08/2018

Lorenz Breidenbach, Phil Daian, Ari Juels, και Emin Gün Sirer, (2017) 'An In-Depth Look at the Parity Multisig Bug', http://hackingdistributed.com/2017/07/22/deep-dive-parity-bug/, τελευταία επίσκεψη: 20/07/2018

Phil Daian, (2016) 'Analysis of the DAO exploit', http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/, τελεταία επίσκεψη: 12/08/2018

Preston Byrne, (2018) 'Whether Ether is a security', https://prestonbyrne.com/2018/04/23/on-ethereum-security/, τελευταία επίσκεψη: 22/07/2018

Παράρτημα Α - Ethereum API caller

Το πρωτότυπο πρόγραμμα που ακολουθεί αποτελεί μέρος της παρούσης διπλωματικής εργασίας. Στο πρόγραμμα καλείται API της δημόσιας αλυσίδας και αντλούνται πληροφορίες για τα block, τις συναλλαγές και τους χρήστες της. Το πρόγραμμα είναι γραμμένο με την γλώσσα προγραμματισμού Python3.

```
import requests, re, json, optparse

def blockAnalysis(xBlock):
    blockEtherUrl = "https://api.blockcypher.com/v1/eth/main/blocks/" + str(xBlock)
    blockReq = requests.get(blockEtherUrl)
    blockReq.raise_for_status()
    blockData = json.loads(blockReq.text)
    try:
        hashBlock = blockData['hash']
        heightBlock, depthBlock = blockData['height'], blockData['depth']
        prevHashBlock, prevUrlBlock = blockData['prev_block'], blockData['prev_block_url']
        sizeBlock, timeBlock = blockData['size'], blockData['time']
        rootBlock = blockData['mrkl_root']
        coinbaseAddress = blockData['coinbase_addr']
        nonceNumber = blockData['nonce']
        transactionDataHash = blockData['txids'] # type: list
        internalDataHash = blockData['internal_txids'] # type: list
    except KeyError:
        pass
    print("The hash (Keccak/SHA3) of the block is: " + hashBlock + ".")
    print("The hash (Keccak/SHA3) of the previous block is: " + prevHashBlock + ".")
    print("The height of the block is " + str(heightBlock) + ", and the depth is " + str(depthBlock) + ".")
    print("The size of the block is " + str(sizeBlock) + ".")
    print("The Merkle root of this block is " + rootBlock + ".")
    print("The time of the block was created: " + str(timeBlock).split("T")[0] + " and " +
str(timeBlock).split("T")[1][:-1] + ".")
    print("The address that received the coinbase and the fee reward for this block: " +
coinbaseAddress + ".")
    print("The number that this user used is " + str(nonceNumber) + ".")
```

```
    if (len(transactionDataHash) > 0):
        print(str(len(transactionDataHash)) + " transaction hashes have been found in this block." + "\n")
    else:
        print("No transaction data in this block!")
    return transactionDataHash

def accountAnalysis(xAddress):
    # (1 Ether = 10^18 wei)
    print("Information for the address " + xAddress + ":")
    addressEtherUrl = "https://api.blockcypher.com/v1/eth/main/addrs/" + xAddress
    addressReq = requests.get(addressEtherUrl)
    if addressReq.raise_for_status() == 404:
        print("Address not found or unsupported.")
    addressData = json.loads(addressReq.text)
    try:
        addressBalance = addressData['balance']
        addressTotalRec = addressData['total_received']
        addressTotalSen = addressData['total_sent']
        addressTotalBalance = addressData['final_balance']
        numberOfTransactions = addressData['final_n_tx']
    except KeyError:
        pass
    print("\tThe confirmed balance of this address is " + str(addressBalance/(10**18)) + " Ether.")
    print("\tThe total amount of confirmed Ether received is " + str(addressTotalRec/(10**18)) + ".")
    print("\tThe total amount of confirmed Ether sent is " + str(addressTotalSen/(10**18)) + ".")
    print("\tThe total balance of this address is " + str(addressTotalBalance/(10**18)) + " Ether.")
    print("\tThis address has done " + str(numberOfTransactions) + " transactions in total.\n")

def transactionAnalysis(xTra):
    transactionEtherUrl = "https://api.blockcypher.com/v1/eth/main/txs/" + str(xTra)
    transactionReq = requests.get(transactionEtherUrl)
    transactionReq.raise_for_status()
    transactionData = json.loads(transactionReq.text)
    try:
        transactionGasUsed, transactionGasPrice = transactionData['gas_used'],
transactionData['gas_price']
        transactionBlockHeight = transactionData['block_height']
```

```
        transactionHash = transactionData['hash']
        transactionAddresses = transactionData['addresses'] # type : list
        transactionTotal = transactionData['total']
        transactionFees = transactionData['fees'] # gas_price * gas_used
        transactionSize = transactionData['size']
        transactionTime = transactionData['received']
        transactionDouble = transactionData['double_spend']
        if (transactionDouble == True):
            doubleSpend = "This was an attempted double spend transaction."
        else:
            doubleSpend = "This was an single spend transaction."
        transactionNumberIn = transactionData['vin_sz']
        transactionNumberOut = transactionData['vout_sz']
        transactionConfirm = transactionData['confirmations']
    except KeyError:
        pass
    print("The height of the block that contains this transaction is " + str(transactionBlockHeight) + ".")
    print("The hash of this transcation is " + transactionHash + ".")
    print("The total number of wei exchanged is: " + str(transactionTotal) + ".")
    print("The transaction was between the addresses " + transactionAddresses[0] + " and " +
transactionAddresses[1])
    print("The amount of gas used by this transaction is " + str(transactionGasUsed) + " and the price of
gas (in wei) was : " + str(transactionGasPrice))
    print("The total number of fees collected by miners in this transaction: " +
str(transactionFees/(10**18)) + ".")
    print("The size of the transaction in bytes is " + str(transactionSize) + ".")
    print("The total number of inputs in the transaction are " + str(transactionNumberIn) + " and the
outputs are " + str(transactionNumberOut))
    print("The transaction was received by the server at " + str(transactionTime).split("T")[0] + " and by
the time of " + str(transactionTime).split("T")[1][:-1] +".")
    print(doubleSpend)
    print("This transaction has been confirmed " + str(transactionConfirm) + " times.")
    return transactionAddresses


def main():
    # Run the parser for the user
```

```
    parser = optparse.OptionParser("\nChoose one: -B <target block> or -A <target address> or -T
<target transaction> or -L(ast block)")
    parser.add_option("-B", dest="tgtBlock", type="string", help="specify target block")
    parser.add_option("-A", dest="tgtAddress", type="string", help="specify target address. Length = 40
characters")
    parser.add_option("-T", dest="tgtTrans", type="string", help="specify target transaction. Length = 64
characters")
    parser.add_option("-L", action='store_true', dest="lastBlock", help="examines the last block")
    (options, args) = parser.parse_args()
    tgtBlock = options.tgtBlock
    tgtAddress = options.tgtAddress
    tgtTransaction = options.tgtTrans
    tgtLastBlock = options.lastBlock
    # API url
    mainEtherUrl = "https://api.blockcypher.com/v1/eth/main"
    # A request to the blockchain - The response is in JSON format
    print("Request to the Ethereum blockchain in progress...")
    mainR = requests.get(mainEtherUrl)
    mainR.raise_for_status()
    mainData = json.loads(mainR.text)
    # Find the height of the blockchain
    height = mainData['height']
    lastUpdate = mainData['time']
    print("\nThe current height of the Ethereum blockchain is: " + str(height) + " (last block)")
    print("The last update was on " + str(lastUpdate).split("T")[1][:-11] + " (Greenwich Mean Time) of " +
str(lastUpdate).split("T")[0] +".")
    if tgtBlock != None and tgtAddress == None and tgtTransaction == None and tgtLastBlock ==
None:
        if (blockReg.search(tgtBlock) != None and int(tgtBlock) <= height ):
            blockAnalysis(tgtBlock)
        else:
            print(parser.usage)
            exit(0)
    elif tgtAddress != None and tgtBlock == None and tgtTransaction == None and tgtLastBlock ==
None:
        if (len(tgtAddress) == 40 and addressReg.search(tgtAddress) != None):
            accountAnalysis(tgtAddress)
```

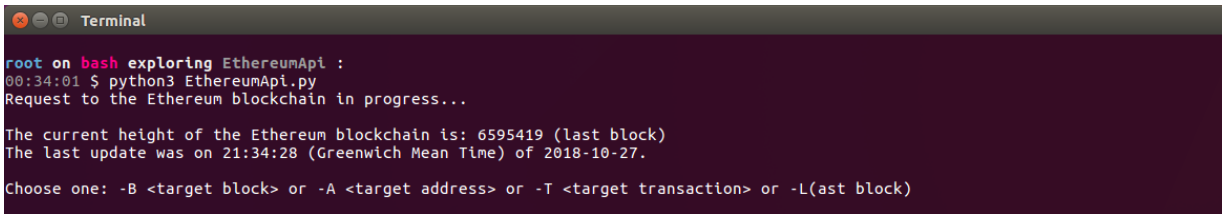
```
        else:
            print(parser.usage)
            exit(0)
    elif tgtTransaction != None and tgtBlock == None and tgtAddress == None and tgtLastBlock ==
None:
        if (len(tgtTransaction) == 64 and transReg.search(tgtTransaction) != None):
            transactionAnalysis(tgtTransaction)
        else:
            print(parser.usage)
            exit(0)
    elif tgtLastBlock == True and tgtBlock == None and tgtAddress == None and tgtTransaction ==
None:
        print("\nFirst the block...")
        t = blockAnalysis(height)
        print("\nAfter the transaction...")
        a = transactionAnalysis(t[0])
        print("\nAnd finally the addresses from the transaction...")
        accountAnalysis(a[0])
        accountAnalysis(a[1])
    else:
        print(parser.usage)
        exit(0)

# Regexes:
addressReg = re.compile(r'^[a-z0-9]*$')
blockReg = re.compile(r'^[0-9]{,10}$')
transReg = re.compile(r'^[a-z0-9]{64}$')
# main method
main()
```

Παράρτημα Β - Οδηγίες εκτέλεσης προγράμματος

Όταν ο χρήστης εκτελέσει το πρόγραμμα του Ethereum API caller θα εμφανιστούν στην οθόνη του οδηγίες για τη σωστή λειτουργία του προγράμματος. Το πρόγραμμα αυτό μπορεί να αναζητήσει και να εμφανίσει τις εξής πληροφορίες:

- Πληροφορίες για μια συγκεκριμένη συναλλαγή που πραγματοποιήθηκε στην πλατφόρμα Ethereum.
- Πληροφορίες για μια συγκεκριμένη διεύθυνση ενός χρήστη και την ποσότητα από κρυπτονομίσματα που έχουν περάσει από την κατοχή του.
- Πληροφορίες για ένα block της δημόσιας αλυσίδας και τις συναλλαγές που αυτό περιέχει.
- Συνολικές πληροφορίες για το τελευταίο block που προστέθηκε στην αλυσίδα.

Για να γίνει αυτό περισσότερο κατανοητό, στις παρακάτω εικόνες φαίνεται ποια είναι η χρήση του και τις πληροφορίες που μπορεί να αντλήσει.



Εικόνα 1: Οδηγίες χρήσης του προγράμματος

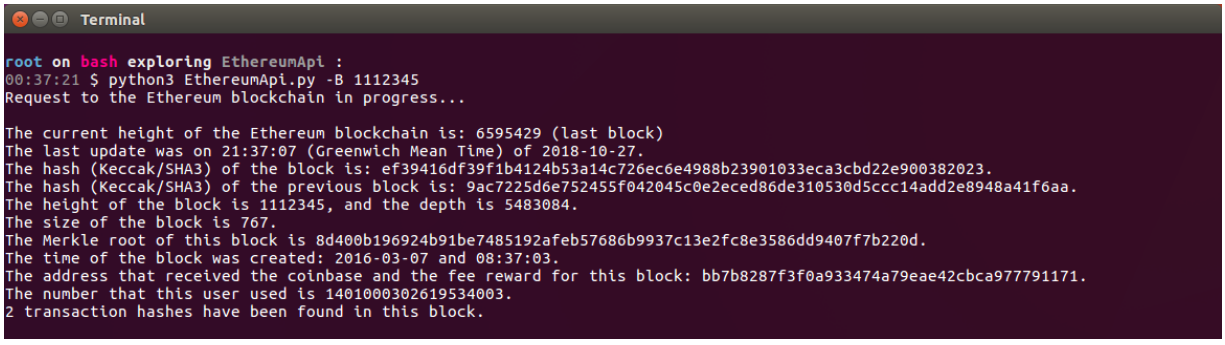
Το πρόγραμμα λειτουργεί με Python 3.2+ και όταν εκτελεστεί για πρώτη φορά εμφανίζει ποιες παραμέτρους μπορεί να χρησιμοποιήσει ο χρήστης και πως αυτές συντάσσονται. Αν ο χρήστης επιθυμεί να αναλύσει ένα block τότε πρέπει να χρησιμοποιήσει την παράμετρο *-B* μαζί με τον αριθμό του block, αν θέλει να αντλήσει πληροφορίες για μια συγκεκριμένη διεύθυνση τότε πρέπει να χρησιμοποιήσει την παράμετρο *-A* μαζί με την διεύθυνση που ενδιαφέρει τον χρήστη και τέλος αν έχει σκοπό να αποκτήσει πληροφορίες για μια συναλλαγή που έχει επικυρωθεί και τοποθετηθεί στη δημόσια αλυσίδα θα πρέπει να χρησιμοποιήσει την παράμετρο *-T* μαζί με τον κατακερματισμένο κωδικό της συναλλαγής. Επιπλέον, αν δεν έχει

στην κατοχή του καμία από τις παραπάνω πληροφορίες που χρειάζονται για την εκτέλεση του προγράμματος, μπορεί να χρησιμοποιήσει την παράμετρο *-L*, χωρίς να προσθέσει τίποτα άλλο, ώστε να γίνει η ανάλυση του πιο πρόσφατου block που προστέθηκε στην αλυσίδα.

B1. Εμφάνιση πληροφορίας ενός block

Για την εμφάνιση ενός block ο χρήστης μπορεί να χρησιμοποιήσει την παράμετρο *-B «αριθμός του block»*. Το πρόγραμμα θα πραγματοποιήσει ένα ερώτημα προς τη δημόσια αλυσίδα και θα αντλήσει τις εξής πληροφορίες:

- Την κατακερματισμένη τιμή του block.
- Την κατακερματισμένη τιμή του προηγούμενου block.
- Το ύψος και το βάθος του block, δηλαδή την ακριβή θέση του.
- Το μέγεθός του.
- Την κατακερματισμένη τιμή του Merkle root.
- Την ημερομηνία κατά την οποία αυτό το block δημιουργήθηκε.
- Την διεύθυνση όπου έλαβε την ανταμοιβή δημιουργίας του.
- Τον αριθμό της λύσης του block.
- Τον αριθμό των συναλλαγών που περιέχει.



Εικόνα 2: Πληροφορίες ενός τυχαίου block.

B2. Εμφάνιση πληροφορίας διεύθυνσης

Για να αντλήσει κάποιος χρήστης του προγράμματος πληροφορίες για μια δοσμένη διεύθυνση πρέπει να χρησιμοποιήσει την παράμετρο *-A «την δημόσια διεύθυνση»* του χρήστη που τον ενδιαφέρει. Αυτό θα εμφανίσει τις εξής πληροφορίες στην οθόνη του προγράμματος:

- Την επικυρωμένη ποσότητα Ether που έχει στην κατοχή του ο χρήστης της διεύθυνσης.
- Την επικυρωμένη ποσότητα Ether που έχει λάβει στο παρελθόν.
- Την επικυρωμένη ποσότητα Ether που έχει χρησιμοποιήσει στο παρελθόν.
- Την συνολική ποσότητα Ether που έχει ο χρήστης στην κατοχή του.
- Τον αριθμό των συναλλαγών που έχει πραγματοποιήσει μέχρι αυτή την στιγμή.

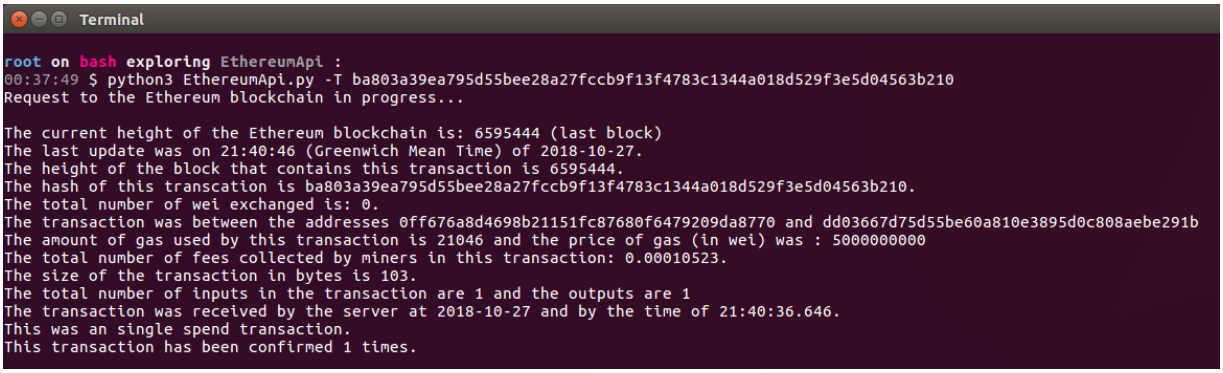
Εικόνα 3: Πληροφορίες μιας τυχαίας διεύθυνσης.

B3. Εμφάνιση πληροφορίας συναλλαγής

Ο χρήστης για να αποκτήσει πρόσβαση στις πληροφορίες που είναι συνδεδεμένες σε μια συναλλαγή που έχει επικυρωθεί και τοποθετηθεί σε ένα block της δημόσιας αλυσίδας, πρέπει να εκτελέσει το πρόγραμμα χρησιμοποιώντας την παράμετρο -T «κατακερματισμένος αριθμός συναλλαγής» και το πρόγραμμα θα επιστρέψει στην οθόνη του τις εξής πληροφορίες:

- Τον κετακερματισμένο αριθμό συναλλαγής.
- Την ποσότητα wei που ανταλλάχτηκε κατά τη διάρκεια αυτής της συναλλαγής.
- Τις διευθύνσεις που χρησιμοποιήθηκαν κατά τη συναλλαγή.

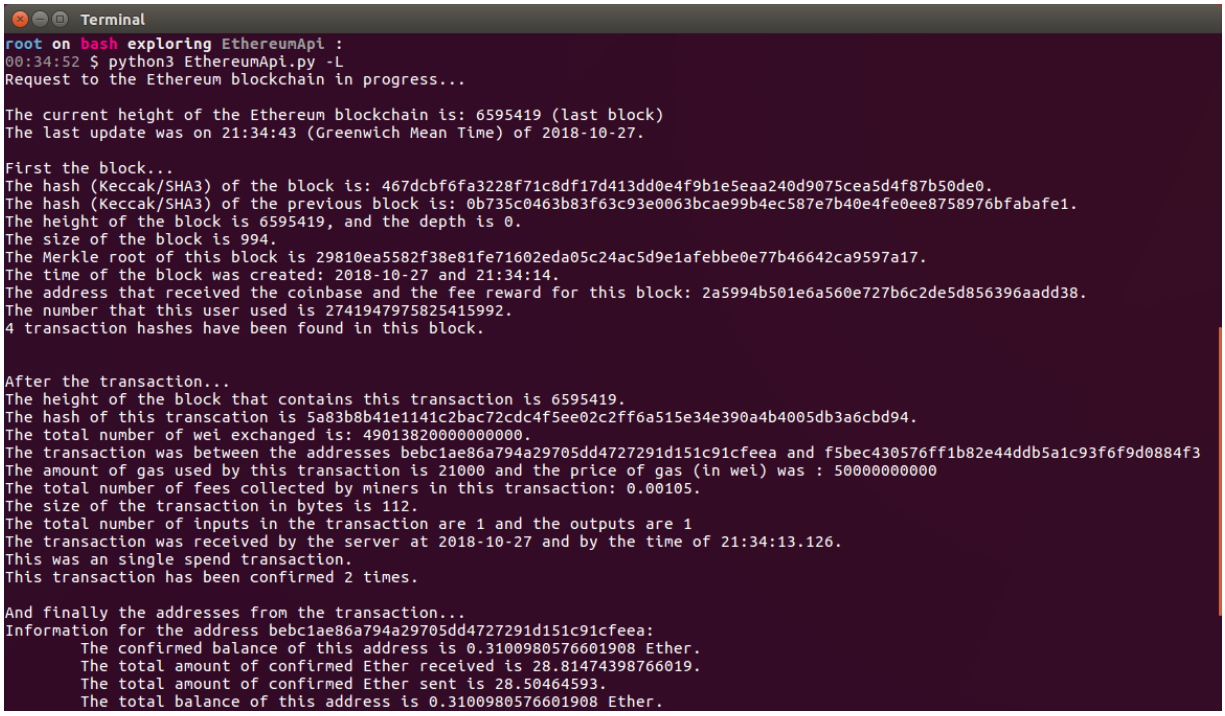
- Την ποσότητα του gas που χρησιμοποιήθηκε σε αυτή τη συναλλαγή όπως και την τιμή του.
- Τα τέλη που συνέλεξαν οι εξορύκτες από αυτή τη συναλλαγή.
- Το μέγεθος της συναλλαγής σε bytes.
- Οι είσοδοι και οι έξοδοι αυτής της συναλλαγής.
- Πότε πραγματοποιήθηκε αυτή η συναλλαγή.
- Την πληροφορία σχετικά με το εάν η συναλλαγή πραγματοποιήθηκε περισσότερες από μια φορές.
- Τον αριθμό των φορών που έχει επαληθευτεί (συνήθως είναι μία).



Εικόνα 4: Πληροφορίες μιας τυχαίας συναλλαγής.

B4. Αναζήτηση του τελευταίου block

Στην περίπτωση που ο χρήστης δεν έχει στην κατοχή του καμία από τις παραπάνω πληροφορίες μπορεί να εκτελέσει το πρόγραμμα ζητώντας όλες τις διαθέσιμες πληροφορίες για το τελευταίο block που τοποθετήθηκε στην αλυσίδα. Για να το πετύχει αυτό απλά χρησιμοποιεί την παράμετρο *-L* χωρίς καμία επιπλέον προσθήκη.



Εικόνα 5: Το πιο πρόσφατο block.