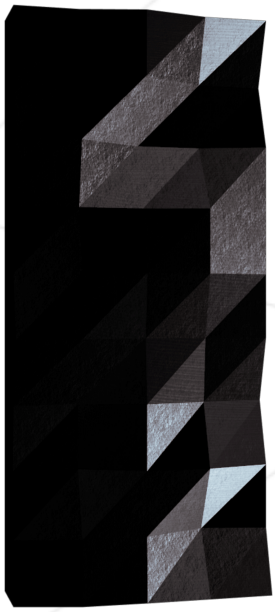

alter_white

A deep dive into Man in the Middle attacks

Summary: This is the second project of a network security branch created by maabou-h.

Contents

Chapter I

Foreword

Here is the recipe for Ham and Banana Hollandaise:

6 medium bananas
1/4 cup lemon juice
6 thin slices boiled ham (about 1/2 lb)
3 tablespoons prepared mustard
2 envelopes (1 1/4-oz size) hollandaise sauce mix
1/4 cup light cream

1. Preheat oven to 400F. Lightly butter 2-quart, shallow baking dish.
2. Peel bananas; sprinkle each with 1/2 tablespoon lemon juice, to prevent darkening.
3. Spread ham slices with mustard. Wrap each banana in slice of ham. Arrange in single layer in casserole. Bake 10 minutes.
4. Meanwhile, make sauce: In small saucepan, combine sauce mix with 1 cup water, 1 tablespoon lemon juice, and cream. Heat, stirring, to boiling; pour over bananas. Bake 5 minutes longer, or until slightly golden. Nice with a green salad for brunch or lunch. Makes 6 servings.

Pineapple-Glazed Baked Ham
1 can (2 lb) boneless ham
1/4 cup dry white wine
Pineapple Glaze
1/2 cup pineapple preserves
1/2 teaspoon dry mustard
Dash ground cloves
Preheat oven to 350F. Place ham in small, shallow baking pan. Bake 10 minutes. Pour wine over ham. Bake 20 minutes longer. Remove from oven. Increase oven temperature to 450F.
Meanwhile, make Pineapple Glaze: In small bowl, combine preserves, mustard, and cloves; mix well. Spread on top and sides of ham. Bake ham 12 to 20 minutes, or until glaze is slightly browned. Remove to serving platter. Garnish with maraschino cherries and parsley, if desired. Makes 6 servings.

If I were you, I would not try this at home.
Seriously.

Chapter II

Introduction

This project will have you dive deeper into network security with packet manipulation on the fly using netfilter, a userspace library that provides packet manipulation capabilities for linux kernel.

`You should take a look at netfilter and iptables.`

Chapter III

Goals

In this offensive project, you will learn how to sniff and queue packets to edit them according to certain criteria before actually sending them, allowing you to understand in more details how packets are treated by your kernel and how much security is present in a basic network environment. You will find that altering packets on the fly on an unprotected network is a piece of cake.

You might find useful stuff in RFC 791, 1071 and 4413 (and possibly others that you will find by yourself)

Chapter IV

General instructions

• This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules

• You are free to use the language of your choice, some might be more adequate than others

• You have to handle errors carefully. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).

• Even if there is no norm required, keep a clean coding style to allow your peers to understand your code. Be reasonable.

• Your program must be able to activate packet forwarding and restore it to 0 once the program quits.

Chapter V

Mandatory part

Since raw sockets require low level privileges, your project must be done inside a VM and must only run as root/sudo user.

For this project, you must turn in a program named alter_white in the language of your choice.
Your program must be able to show a usage menu, as usual.

Example usage:

```
   foo@debian:~/alter_white$ sudo python alter_white.py
usage:
   python alter_white.py [target ip] [filter expression] [options]

   options:
      - neutralize:               neutralize target by flooding icmp packets
      - awesome bonus functionality: do stuff
      - another great bonus option: do some other stuff
   target ip:
      the victim of the attack
   filter expression:
      filter expression (tcpdump style) for monitoring rules

   foo@debian:~/alter_white$
```

The program will take two mandatory arguments, the target ip and the filter expression, using the pcap-filter syntax.

`man tcpdump man pcap-filter`

After being launched, the program will listen to the traffic on the target ip according to the filter expression. Your program will spoof the arp cache of the victim by associating the gateway IP to your own mac address, allowing you to listen to the traffic.

How much information is displayed is up to you, but you must at least print the type of packet and if it is an incoming or an outgoing packet.

Example usage:

```
   foo@debian:~/alter_white$ sudo python alter_white.py 10.0.2.15 "tcp or icmp"
- Launched alter_white on victim 10.0.2.15:
** Spoofing arp cache of victim **
arpspoofing: SUCCESS
** Listening to traffic with rule: tcp or icmp **
Incoming: [ICMP] 10.0.2.255 -> 10.0.2.15
Incoming: [ICMP] 10.0.2.255 -> 10.0.2.15
Outgoing: [ICMP] 10.0.2.15 -> 10.0.2.255
^C
** User requested end of program **
** Restoring iptables and ip forwarding rules **
out: SUCCESS
   foo@debian:~/alter_white$
```

A simple Ctrl+C should end the program, restoring the normal state of the computer (ip forwarding, firewall rules etc...)

Chapter VI

Bonus part

We will look at your bonuses if and only if your mandatory part is EXCELLENT. This means that your must complete the mandatory part, beginning to end, and your error management must be flawless, even in cases of twisted or bad usage. If that's not the case, your bonuses will be totally IGNORED.

Chapter VII

Turn-in and peer-evaluation

• Submit your work on your GiT repository as usual. Only the work on your repository will be graded.

• You have to be in a VM with a Linux kernel > 3.14. Note that grading was designed on a Debian 7.0 stable.