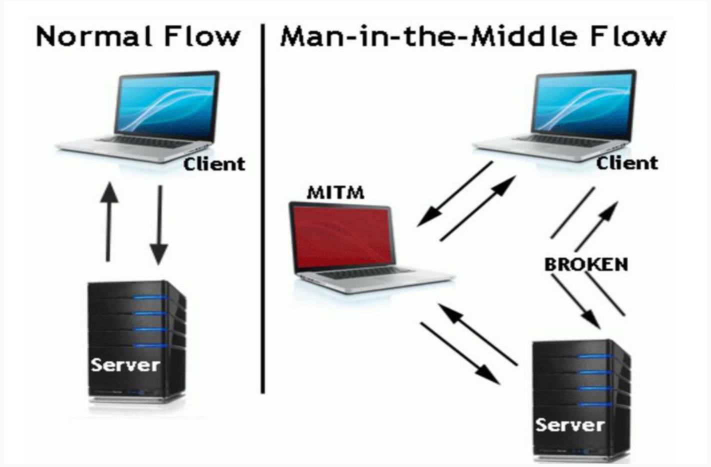
Objective

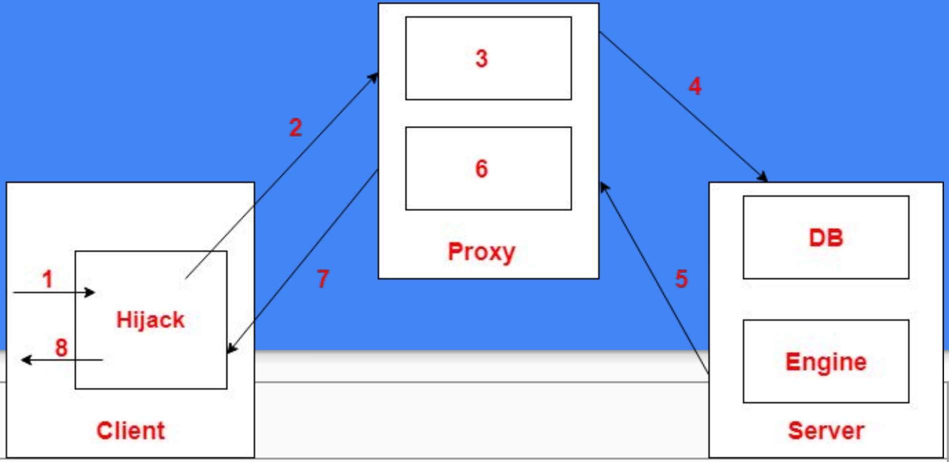
1) Hands-on implementing a man-in-the-middle (MITM) attack
2) Understand how **MITM** works
3) Understand the **socket** communication
4) Understand the importance of **authentication** in web security
5) ...

Function Overview

- **C-S**: Normal communication between client and server - **NOT** on the **target** port
- **C-M-S**: Compromised communication between client and server - on the **target** port: 8885
  - MITM - damage on client request
    1. Forward as is
    2. Drop request
    3. Modify and forward
  - MITM - damage on server reply
    1. Forward as is
    2. Drop request
    3. Modify and forward



MITM Procedure



Step	SRC	DEST	Notes
1	client_port	server_port	Client believes it is connecting to server.
2	client_port	proxy_dw_port	Hijacking redirect it to proxy
3	-	-	Proxy receives client request: 1) authentication 2) damage(action 0/1/2)
4	proxy_up_port	server_port	Proxy send the modified request to target server
5	server_port	proxy_up_port	Server treats proxy as normal client: 1) authentication 2) echo
6	-	-	Proxy receives server reply: 1) authentication 2) damage(action 0/1/2)
7	proxy_dw_port	client_port	Proxy send the modified reply to Hijacking
8	server_port	client_port	Hijacking change source addr to target, client believes it is from server.

Implementation - overview

1) Language: C
2) Programs: client, server, proxy
3) Protocol:
   - Self-defined header on top of socket
     - SRC = struct sockaddr_in src;
     - DST = struct sockaddr_in dst;
     - SEP = “-”
   - build package: *int pack_build(char* frame, struct MyHeader* head, char* msg, uint32_t   msg_len)*
   - parse package: *int pack_parse(char* frame, uint32_t frm_len, struct MyHeader* head, char* buf)*
   - Authentication: *bool isEqual_sockaddr_in(struct sockaddr_in src, struct sockaddr_in target)*

SRC	DST	-	MSG
16B	16B	1B	MAX 1024B

Implementation - client

1) Initiate connection to server - *connect(...)*, hostname+port
2) Ask user to enter content for sending
3) Build the package: *pack_build(...)*
4) Send request to server: *send(...)*
5) Wait for server reply: *recv(...)*, block operation
6) Receive reply, go back and restart from step 2

Implementation - server

1) *listen(...)* on specified port
2) *accept(...)* client connection
3) Process in child process - *fork()*,
4) parent process continues listening - could deal with multiple connections
5) In child process:
    a) *recv()* client content
    b) Parse the package - *pack_parse(...)*
    c) Authentication client - *isEqual_sockaddr_in(...)*
    d) Authentication pass: Echo client by prepending “$$$” - *pack_build(...)*
    e) Continue from step a), if blank content, close child process

Implementation - proxy/Hijacking

1) Compare client connection destination: *isEqual_sockaddr_in(...)*
    a) If match target address (8885): redirect connection to proxy
    b) Otherwise, keep silent.

2) After receive proxy reply, modify source address to target server

Note: hard code in client, injection procedure not implemented

Implementation - proxy

1) Connect to target server 8885 - *connect(...)*
2) Listening on port 9999
3) Accept Hijacking redirected connection - *accept(...)*
4) Process in child process - *fork()*, parent process continues listening - could deal with multiple connections
5) In child process:

Continue on next page...

Implementation - proxy (cont)

5) In child process:
    a) *recv()* client content, parse the package - *pack_parse(...)*
    b) Authentication client - *isEqual_sockaddr_in(...)*
    c) Authentication pass: ask hacker to enter damage operation(0,1,2) on upstream
    d) Execute the damage and build the package: *pack_build(...)*
    e) *send()* package to server
    f) *recv()* reply from server, parse the package - *pack_parse(...)*
    g) Authentication server - *isEqual_sockaddr_in(...)*
    h) Authentication pass: ask hacker to enter damage operation(0,1,2) on downstream
    i) Execute the damage and build the package: *pack_build(...)*
    j) *send()* package to hijacking/client, back to step a)

Demo (localhost)

1) Normal C-S.
    a) 1st terminal: ./server 8884 (*not listening on the target port*);
    b) 2nd terminal: ./client localhost 8884;
    c) Server echoes whatever client send by prepending $$$;
2) C-M-S: MITM.
    a) 1st terminal: ./server 8885 -- *listening on the target port*;
    b) 2nd terminal: ./proxy --*listening on 9999*;
    c) 3rd terminal: ./client localhost 8885;
    d) Enter string from client terminal and observe information on server end and proxy end;
    e) The proxy modifies the client package by repeating all its content;
    f) The proxy modifies the target server's package by replacing $$$ pilot to @@@;

Demo (IP, could test on three machine in the same LAN)

1) Normal C-S.
    a) 1st terminal: ./server 8884 (*not listening on the target port*);
    b) 2nd terminal: ./client <server_ip> 8884;
    c) Server echoes whatever client send by prepending $$$;
2) C-M-S: MITM.
    a) 1st terminal: ./server 8885 -- *listening on the target port*;
    b) 2nd terminal: ./proxy <server_ip> 8885--*listening on 9999*;
    c) 3rd terminal: ./client <server_ip> 8885;
    d) Enter string from client terminal and observe information on server end and proxy end;
    e) The proxy modifies the client package by repeating all its content;
    f) The proxy modifies the target server's package by replacing $$$ pilot to @@@;