**CIS-449 – Intro to Software Security**

**Personal Notes – Demetrius Johnson**

**1-11-2023**

**With Professor Dr. Anys Bacha**

# Lecture 1 – Introduction (intro slides)

* What is security?
  + Attackers want to get programs to be stuck in kernel mode.
  + That way, they can perform an attack; so they desire to find and exploit bugs.
* The Security Problem
  + Exists largely because of buggy software – that is spurred by companies rushing development due to financial and project deadlines.
  + Software engineers are under a lot of pressure around the clock.
    - They try to meet deadlines.
  + Bugs often come up early in a software project and go undetected; and later on it can be too expensive (time and thus financial) to fix a bug as a result of pure mistake or bad design; so software work arounds by developers are employed even in the next generations/versions of a software.
    - General goal of work arounds: goal is users will see/find the bug with a low probability.
    - Think about Software Engineering 1 principles.
  + So, time to market (through agile development) is very important, but the compromise is security/bugs; time to market is necessary to keep a software competitive.
    - Rely on future releases after core is developed to fix bugs and add features.
    - Agile development is inherently a buggy process.
* Social Engineering
  + Bait, Phish, and Tailgating.
  + It is easier to conduct attacks on the LAN of a company/organization because often you cannot have too much security on the inside of the local network because users often need more unrestricted access and faster speeds (such as a research facility).
  + Usually, organizations have guards/firewalls up around the organization, but internally it is a lot less secure because too much security would slow operations and make working inside of the organization not practical/unusable.
* How companies lose Data
  + Inside error, lost/stolen laptops, insider attack, malware.
* Goal of malware: outpace human response.
  + Do as much damage as possible in shortest amount of time.
* From the professor: “there is no such thing as a secure system; rather, it is a matter of what level of security a system has”…
* “Prevention is better than curing”.
* How to Think about Security (security model)
  + Components of Security:
    - Policy
      * Confidentiality
      * Integrity (i.e.: CRC = cyclic redundancy check to make sure all bits transmitted were received)
      * Availability
      * (CIA) acronym
    - Threat model(what threats are we considering?)
    - Mechanism (method to enforce threat model)
    - End goal (make sure mechanism upholds your threat model)
* Issues with policies
  + Account recovery policy is a security compromise.
* PUFs: Physically Unclonable Functions
  + An alternative method for login credentials, which use atomic-level fingerprints on a chip to check authentication using the chip of a device.
* Issues with Threat Model (Secure against WHAT?)
  + DES = Data Encryption Security; now inferior (use AES).
  + We may incorrectly assume certain things (such as DES is secure, but then 20 years later, it may become easily overturned/broken by new technology or some unconsidered situation.
* Issues with Mechanism
  + Mechanism = method of enforcement.
  + One example of an issue: limit number of attempts before you can try to attempt to authenticate again (i.e. exponential backoff = time grows exponentially between allowed attempts).
    - So for example: if you do not put in place a limitation of attempts, then some brute force attack may be successful (and more faster at success) (such as iCloud when it first released).
* Other case studies…
  + Mat Honan case
    - Amazon allowed (in the past) anyone to add a valid credit card to any account
  + Stuxnet malware
    - First worm known to attack SCADA systems (supervisory and control data acquisition system) that controls things like DAMS, bridges, nuclear plants…etc…its an entirely different architecture than x86…etc.
    - In order to get official Windows logo to load on your machine, there are all of these Microsoft logos testing 🡪 which can then give out official windows logo certificates (digitally signed software/firmware that can be trusted to be incorporated into the official Windows OS).
    - Caused Centrifuges connected to PLCs via Siemens software to destroy U-235 Uranium enrichment centrifuges.
  + Heartbleed
    - SSL/TLS provides https (the ‘s’ = secure aspect of http).
    - SSL had many bugs and instead of changing version in update, they just renamed the software to TTL.
    - You usually do not want to tear down and rebuild connection between requests on a website 🡪 it would make browsing a webpage very slow.
    - Sent 64KB per transaction (malicious request).
      * Scraping nearby memory that belongs to other sessions.
  + WannaCry (ransomware)
    - Public key can be passed around.
      * Only associated private key can decrypt the messages encrypted by the associated public key.
    - Used the AES (CBC = Cypher Block Chain version) encryption key algorithm (only 1 private and 1 public key) to encrypt files on a machine 🡪 they would also leave the private key on the server on which they sent the encrypted files.
    - If person paid the ransom, they would be sent the private key so they can decrypt their files.

# Lectures (3 of them) – Unix/Linux

* Intel x86 = CISC-based (complex instruction set) architecture processor
* Linux kernel = developed for RISC based architectures (Reduced instruction set computer)
  + Android is Linux-based kernel.
* GNU = GNU NOT UNIX (software suite/tools that Unix supported – versions for Linux were written and named GNU)
* Shell
  + Interact with the system using a Script-based command program to manage other programs.
  + Less overhead than GUI
* Shell commands
  + You can do cd .. to go to parent directory
    - You can do cd ..\.. to go up to grandparent directory
    - Etc…
* Stdin = by default expects input from keyboard
* Stdout and stderr the default to is output to current shell window.
* Parameter v input
  + Parameter is part of original command (function/program)
  + Input = when a given command (function/program) is running and you interact with it by giving it additional input besides the initial input from parameters.
* Note: not all commands are a part of the shell program itself; many of them are programs that you can call from the shell and the output of the program is sent to the terminal/shell window.
  + i.e. exit = a shell function
  + ls = a separate program where output is sent to terminal window.
* To remove a directory
  + Rmdir
    - If not empty, it will not allow you; so you need to recursive (using -r option/version of the function) deletion of all files inside the directory before you can delete the directory 🡪 use rm -r
    - Not -r is not he same as -R 🡪 capitalization matters for options!
* Note when you have a bridge (like USB hub) connected to a machine
  + Each device is a function under that one device
    - The device is the actual module (I.E. usb 3.0 module) that is attached via busses to the motherboard.
* “--" versus single “-“
  + Most long or fully written names of options uses two dashes
  + Single letter options or shortened named versions of the option usually uses a single dash
* Files that begin with . are hidden files 🡪 use ls -a (all) option to view them
* Create a file using “touch” command
  + You can change time stamp of files using -t option; as far as file system is concerned, the time stamp is whatever you change it to including in the meta data
    - However there are ways to look at inodes and figure out true date of a file creation (digital forensics)
* In Linux (Unix file system), all devices are treated as files (including disks, usb devices, etc.)
  + Makes writing device drivers very easy to do since it is simply writing to a file to change a devices registers.
  + Inside of the /bin directory (/ = root directory)
* Cat command = concatenate information of a file to the terminal 🡪 thus it simply outputs text that is in a file.
* /proc directory is a virtual file system built when machine boots (when power lost, virtual file system is lost) – it is not stored on the disk; it is created and stored in working memory only during boot.
  + Allows you to change the kernel on the fly without having to recompile the kernel.
  + Basically allows you to test changes to the kernel and be efficient and save time before truly committing the change and having to recompile kernel every time during your testing.
  + All of the numbers are directories = programs running in the system (PIDs = process IDs)
    - Any time a program launches, it is going to create a directory inside of /proc with a number that represents its process ID.
    - When you use ps command it outputs all processes running and their IDs
    - Note: for .so endings inside of the map file inside of PID directory, it stands for “shared object” which is usually a dynamically linked library file shared by all programs --> thus the memory map range will be the same as if you looked at another program using the same library.
* /temp contains status/log information about actions taken in the system since boot (lost/overwritten when system reboots)
* /var directory also contains log information but will survive between boots.
* /user directory is where user programs are installed
* more v less command
  + does same thing
  + but less only pulls in partial data into memory for you to page through (faster if you only need to look through a few pages or if you will not need to view pages rapidly)
  + more pulls entire file into memory for you to page through (faster if you need to page a lot, unless file size is very large causing thrashing to occur).
* How to access history of all of your commands that you can go back to using up arrow
  + Use *history* command
  + You can find the memory buffer or file where the history function stores the commands user typed in during a session (its open source so go look and find out!)
  + Alternatively you can simply use history command and redirect output to a .txt file so you can copy the script essentially.
    - Then take script and paste it into terminal and run all commands at once! (may need to provide some input values to programs called by the script in order for entire script to run)
* Special redirect to /dev/null 🡪 the bit bucket; a file where nothing is stored or saved.
  + Example: command > /dev/null
    - Sends output of the command program to null file
    - Better than using cntrl^S which suppresses all output of all programs.
* Professor showing redirect demonstration of ‘cat > somefile.txt’:
  + This is what he typed using the keyboard so that cat would redirect output to somefile.txt:
    - “hi there, get out of here, bye cruel world”)
* Combining (stacking) commands using pipe operator/symbol 🡪 |
  + It is similar to a neuro network
  + Very similar to redirect operators > and <, but combines them with a buffer file so you do not need as many commands for using the output of one program as the input of another.
  + Example:
    - Cat file.txt | grep hello | wc -l
    - Note: cat file.txt > grep hello is no the same!
      * You would need to create a bunch of temporary files and use a lot of < and > operators (|operator does the (temporary) file creation and redirection in the background)
* How send output to a file and also view it on the screen:
  + Use combination: first send output to a file, then pipe that file to cat (and/or tail -f) command to show its contents on the console.
* Note for permissions: permissions do not travel from machine to machine
  + When a file is copied to a new machine, then the umask of the target machine is used to assign default permissions to the file.
* Permissions use octal (2^3), since u g or o need only 3 bits to say whether they have read, write or execute (saves space to just use octal).
  + Example: chmod 600 🡪(6) for owner, 0 for groups, 0 for others
    - 6 = 110 (rw-), 0 = (---)
    - So owner can read and write, groups and others cannot rw or x.
* Note: you can take away all permissions for everyone, including the owner, but the owner can still change permissions back since they own the file.
  + This can be useful as an owner if you have a file that you do not want yourself or anyone else to accidentally change/execute the file in any way.
* Note, use sudo (super doer) only when you need it.
  + Even if your username has an administrative account (sudo account), it can help protect you to not to do a dangerous/powerful command without being aware of it – so sudo can help you be more aware/reminded that you are possibly doing a command that is very powerful and consequential.
  + NOTE: THERE IS NO PROTECTIONS: for example, if you are in /bin folder as a root user (or using pseudo command to briefly change to your root profile if one exists under your username), then if you issue rm \* command, everything in that folder will be deleted from the disk!
    - So root users need to add safety on their own if they want it: for example, map rm command to move “deleted” contents to a “trash” directory.
* Note for passwords:
  + Stored as encrypted inside of the /etc/shadow file, which is root owned, and when you log in, the OS will put your plaintext ***into the algorithm*** and generate the ciphertext and compare it to the ciphertext stored inside of /etc/shadow file.
  + Solution to change password since /etc/shadow is root owned:
    - Give a program root privileges.
    - More on this later…

# Lecture 2 – Buffer Overflow (BOF) (lec2-bof slides)

* Java compiles first into java byte code and sent to a jvm (java virtual machine) that checks all memory access 🡪 improved security 🡪 but no pointers and more overhead (for both compilation and runtime) because each memory access goes through jvm.
* C++ on the other hand is faster, has pointers, but has the big risk of buffer overflow.
* With buffer overflow, you can insert code at that location; and then jump to that location, and execute the code you inserted.
* Also remember: PAGING 🡪 frequently/recently accessed (loaded into RAM) disk files are placed in the cache (much faster memory) and periodically written to disk 🡪 so if you can overflow that cache buffer, you can corrupt open files or frequently accessed files that are periodically synced from cache with the disk if the file changes.
* Many vehicular software are c/c++ based; also Facebook, Google, SQL, and many beefy applications.
  + Python and java can be more safe, but they suffer from performance because of their architecture; also they are not as efficient when it comes to hardware resource use (c++ has a much smaller memory footprint because of its capabilities).
  + c/c++ is the best performing language, but with the great power comes great responsibility (it can be dangerous).
* c/c++ does not perform bounds checking explicitly because we do not want to be like jvm (which slows things down).
* Remember, the OS handles memory management (including I/O).
* Remember, with virtualization (VMs), the *hardware* handles (in tandem with OS) they paging and virtual paging tables.
  + Very complex 🡪 but necessary for VMs to work well.
    - Hardware control allows for VTx and VTd (much faster than only using host OS trap methods to keep memory segmented, which is essentially then just emulation, which causes VM to be extremely slow).
* Python is a very good machine-learning and data science language.
* Notable BOF Attacks:
  + Originally, internet was called ARPANET (used by scientists mostly) 🡪 was not designed with security in mind
  + Morris Worm (1988) (first worm of the internet 🡪 originally ARPANET)
    - Morris made a worm to infect ARPANET
    - Exploited the fingerd vulnerability (which had a 512 byte username upper limit)
    - He passed in more than 512 bytes; and it was not a username he passed in, but code.
      * this caused the fingered machine to replicate itself on other machines that were caused to finger via the first malicious finger.
    - Morris wanted to be sure he was successful, so he made the worm replicate itself multiple times on each machine it infected.
    - He made it so every 7 checks, the worm would replicate itself and run (thus machines became bogged down with thousands of worm processes running).
    - He brought down the entire ARPANET.
    - They had to go node-by-node and bring each node offline to clean them then put them back on ARPANET because if even 1 node was still infected, every node on the ARPANET would start to get infected again.
  + Code Red (2001)
    - HTTP GET request 🡪 implanted worm in payload of the GET request.
    - Whitehouse simply mapped DNS to another IP to get around the attack.
  + SQL Slammer (2003)
    - SQL was originally Microsoft’s major cash cow
    - Used UDP (connectionless, best effort transport layer 4 protocol)
    - Attack could’ve been prevented for many SQL servers if machines installed the available patch.
  + Conficker Worm (2008-2009)
  + Flame (2010-2012) 🡪 new shift in worm malware.
    - similar to Stuxnet (believed to be precursor to Stuxnet actually).
    - Large payload to include many capabilities so that you could cyber spy.
    - Changed its bytes constantly so that its hash id value would be very hard to track its signature.
      * Made it very hard to analyze

# Lecture 3 – Memory Layout (lec2-bof slides)

* Remember, the OS gives every process the illusion that it owns the entire memory address space of the system.
  + It does this through virtual addressing through TLB (Translation Lookaside Buffer), which is a hardware method of virtual addressing (very fast) (remember CIS-450 Operating Systems).
  + Also, can do this through paging by switching pages that processes (and are made up of) needs in and out of memory via kernel function (OS functionality).
* Remember the data segments:
  + Global variables 🡪 data segment
  + Local function variables 🡪 stack
  + Static variables 🡪 data segment (or BSS=BootStrapSegment segment if static variable is uninitialized; OS uses BSS by initialize all data to 0=null so that any uninitialized static variable go here and are initialized with 0).
    - Remember, for static variables, they are only initialized once; so if function is called again, that initialization line will be ignored.
  + Heap🡪 pointer memory allocated here
    - The pointer variable itself goes on the stack.
  + Text (code) segment 🡪code for all functions.
  + BSS, Data segment, and text (code) segment are known at compile time
  + Heap and Stack known at runtime
  + Cmdline & env known when process starts.
* Stack and Heap grow in opposite directions
  + Heap grows up (lower addresses to higher addresses)
  + Stack grows down (higher to lower addresses)
  + This makes sure processes don’t waste space by growing in opposite directions towards each other.
* Heap is managed through OS by malloc() (memory allocation) function.
* Remember, OS kernel manages programs, including which line to execute next, and how to jump back to other functions (return), by causing all functions that call another function to push their return address on the stack.
  + Uses eip (instruction pointer), ebp (base pointer), and esp (stack pointer) pointers..
* Why do we call sizeof(instruction) when trying to figure out offset for the next instruction?
  + One reason is that so the same code can run on different architectures (64 or 32) where instruction sizes are different; also different architectures have different sizes for instructions, why others always have fixed sized:
  + Sizeof(instruction): This refers to the number of bits making up an instruction.
  + The size or length of an instruction varies widely, from as little as four bits in some microcontrollers to many hundreds of bits in some horizontal microprogrammed machines. However, most CPUs have instruction sizes between 8 and 64 bits.
  + Computers with a RISC architecture typically have fixed instruction length (often 4 bytes = 32 bits), although ARM processors with a Thumb extension can have either 16 or 32-bit instructions.
  + Computers with a CISC architecture typically have instructions of widely varying length, for example 1 to 15 bytes for x86.
  + The 6809 processor, for instance, was a CISC processor with a variety of addressing modes. The smallest instruction was one byte, e.g. just an opcode with no operands such as ABX (add B to X). Most instructions were two bytes; one for the opcode and one for an 8-bit operand, which could be either a single byte literal, an offset for an index register, or a zero-page address. Three-byte instruction had a 16-bit operand, used for two-byte literals or extended addressing to anywhere within the 64 KB memory. In addition, the 6809 had a number of instructions with two-byte opcodes, creating some instructions as long as four bytes.
  + In von Neumann architecture computers, where instructions and data are located in the same memory space, instructions are some multiple (or sub-multiple) of the memory data width.
  + In Harvard architecture machines, where programs and data reside in separate address space, instructions do not have to be the same width as data. For example, in Microchip’s 8-bit PIC16 microcontrollers, the RAM is 8-bits wide, but the instructions are 14-bits wide.
* Complex Instruction Set Computer (CISC 🡪 can have varying instruction sizes = very complex; not used very much anymore; can be used to make one-liners that can do many things in one line 🡪 could have smaller and bigger sizes depending on how much information you put into an instruction).
  + This makes decoding very complicated for the compiler, and made it take compiling take a long time.
  + So then RISC (Reduced Instruction Set Computer architecture) came along to make it much more portable and easy for compilers and high performing and replaced CISC.
    - This allows you to issue many instructions in a short amount of time 🡪 this helped the RISC architecture to be very fast.
  + X86 is CISC, but since they are stuck with CISC instruction set because of backwards compatibility, they have special hardware to convert CISC to RISC instructions.
  + ARM-based (android) are RISC-based architectures.
  + However, CISC still has its use: it can save memory by allowing instruction sizes to be smaller if possible (but at the cost of more complexity).
* %eip = 4(%ebp) means load whatever ebp is pointing to + add 4 into eip register.
* %ebp register just acts as a reference point on the stack so we know where to get the return address of the calling function; see the screenshots of the slides below.
* Function calls:
* Diagram

  Description automatically generated
* A picture containing diagram

  Description automatically generated
  + New %ebp refers to the current memory location that the %ebp register is referencing (pointing to).
    - Everything is referenced relative to where the ebp pointer is pointing to. That is why we say ebp is the origin (base) of a stack frame.
* Graphical user interface, text, application, email

  Description automatically generated
* Graphical user interface, text, application

  Description automatically generated
* Strcpy() function only copies data up to null terminator (all strings == char arrays terminate in \0 ==null terminator).
* Buffer overflow:
  + Could cause ebp and eip values pushed on stack for a function (ebp controls reference to the stack frame of the function and eip stored pointer back to caller function) to be overwritten and thus when you return you can end up in protected memory (memory that is a part of another process).
    - This causes segmentation fault (hopefully, if not then someone may be able to exploit this and the program may not crash and they can access protected memory (memory from other processes).
* NOTE: You need %eip register to point to next instruction in memory (of a particular function/process); eip jumps around the code segment.
  + The %ebp register is needed so we can reference local variables that are a part of a stack frame of a given function; each function has its own stack frame, and thus needs their own %ebp so that it can reference its own local variables on the stack appropriately.
* Most programs require user input 🡪 that is why buffer overflow via user input or input from another program is often a vector of attack.
  + Attackers want to do code injections especially via buffer overflow.
  + Goal is to overwrite the eip so that a function returns to malicious code injection.
  + One challenge for attackers is that they need to load executable machine code directly into memory.
    - Also, machine code must not contain any 0’s (all 8 bits of a byte as 0 == 0x00 == 00000000) (== avoid null terminators, since most buffer overflows use functions that stop at null).
    - Goal is to run a general purpose shell that provides attacker access to system resources (kernel-level privileges needed).
    - In order to not have null pushed, we use code to push 0s onto the stack instead of the buffer input itself.
    - You need to use a special compiler that does do optimization so that you can create machine code (executable) that is exactly the length and order of machine code that you need so your instructions are not altered and you can be more surgical about what is passed into the buffer.

# Lecture 4 – Code Injection (lec2-bof-slides)

* Remember: you cannot have any 0x0 in your machine code in order to successfully do a bof attack - 0x0s must be passed via another code that will wrap that 0x0 to pass it through.
* Challenge 1 for is not passing 0s because input will terminate for buffer.
* Challenge 2: we can only write memory sequentially
  + This includes changing value of eip pointer to point to instruction relative to ebp where you have also overflowed the memory with the rest of the shell code that you want to inject and run.
* Challenge 3: we need to determine return address of the current function on the stack and find where %eip is saved (memory location) so we can overwrite it.
  + We don’t know how far %eip or %ebp is from end of the memory of the current function’s stack.
  + Need to brute force the program (if possible) and crash it as many times necessary until they find the %eip
    - Requires stack start location to be the same every time system boots; this will make brute forcing all 2^32 addresses much more efficient.
    - How to improve chances? Use NOP Sleds
  + NOP Sleds
    - NOP = no operation: an instruction that does nothing; simply executes nothing, moves eip, and burns energy in the process.
    - The more NOP added, the better chance you have at succeeding the attack.
    - As long as you land between eip and a NOP, you will be successful.
    - A picture containing chart

      Description automatically generated

## USERS AND GROUPS for LINUX:

* Two primary users: root and non-root.
* Attackers want to be root.
* The uid integer value associated with the char array username is what is actually used to determine if a user has certain privileges and can execute certain code.
* Most of the time when a security measure becomes more useable, it becomes less secure.
* A “salt” = a random string of characters that is generated every time an account is re-enabled or created; it is added to the password of the user in order to make the password stronger when it is hashed so an attacker cannot get the hash and brute force hash to find a hash that matches
  + Remember hash functions are one-way; and also (essentially) collision free so that every input generates only one unique output.
  + By adding salt, you increase password size, and also you make it more difficult to brute force and compute hash code and compare it to a hash lookup table (dictionary/common password attack).
  + Salt can invalidate these pre-computed hash lookup tables.
  + But: if user inputs password, hash is generated, and checked against hash code stored in memory, what about the salt added to their password? If the OS automatically adds salt, to the password, then isn’t the OS the only one who knows what the salt is and therefore must have the salt stored somewhere in memory?
  + So salt is not bullet proof but slows down the brute force process.
* Users and Groups
  + groups allow users to access a common resource
  + user can be a member of multiple of multiple groups
  + owner, groups, others: rwxrwxrwx (3 fields for each type of category); the example means all categories can read, write, and execute.
    - Thus to represent those 9 total fields = need 9 total bits (1 = can, 0 – cannot)
    - Represented as an octet: example, 644 = 110100100 = 9 bits; each digit in the octet number represents 3 bits.
      * Unmask will set permission opposite; so by default all permissions are set (0 = can, 1 = cannot).
  + in Linux, file extensions mean nothing.
  + Most popular Linux distro is REL = REDHAT distro. Actually, in terms of industry standards, most companies use Linux (particularly, REDHAT distro (distribution)).
* Security related commands in Linux
  + You actually want to avoid using a root user to run commands – use it only when you need it.
  + You can use sudo under a particular user so that it associates their account for the given command as a root user.
    - You need to grant superuser privileges through an actual root user (this is an example when you must login as a root user) to make a user a sudoer in order that they can use sudo command (which also helps track which user did what privileged commands).
    - Also, you need to actually be logged in as a root user (cannot just use sudo) to write to the /etc/shadow file and other /etc files.
* Set-uid
  + Set-uid program allows the user’s privileges to be temporarily elevated (changes their actual uid to a privilege user) and allows them to write to /etc/shadow file.
    - Most programs need privileges or they would be useless; this is why programs are commonly the target of gaining shell access to a machine, since it likely needs/uses elevated privileges at times during its execution.
  + passwd program is a set-uid program under the root (uid=0) which allows the passwd executable to elevate its privileges (change uid = 0) temporarily when necessary so it can have root privileges.
    - So it is similarly to “sending” execution to root (uid=0) user to handle privileges that it needs.

# Malware (lec4 malware slides)

* There is such a thing called a power virus: cause CPU to run at high speed to damage the circuitry by overheating it.
* There is such a thing as crypto mining: infecting a user by hijacking their resources to do mining for you.
* APT malware: Advanced Persistent Threat
  + Stay in background, collect data
  + Hard to detect
  + Similar to crypto mining attack but even more undetectable because it consumes less resources.
  + This can be done even in firmware by supply chain hits so that firmware code can be loaded on thousands of machine to spy (exfiltrate) in the background.
* New encryption is a double-edged sword: if it works great, it can help secure data, but then also that same encryption technology can be used to secure data by maliciously encrypting data and creating a ransom for the security key.
* Man in the middle attacks 🡪 program that maliciously sits between kernel and user mode.
* Virus v worm (similar to real life definition of virus v bacteria)
  + Virus need a host program (they are parasitic)
  + Worms are standalone programs (relatively speaking); no user intervention needed (i.e., a bot)
  + Although the line between virus and worm is blurry as some malware can be hybrid.
* Virus main goal: avoid detection as long as possible.
* Worms main goal: spread (and spread extremely quickly) to outpace human response.

# Lecture 4.1 (missed 🡪 he finished malware slides)

# Lecture 5 – Cryptography (lec5 slides)

* For algebraic attacks, if you have enough outputs, you can try to derive the equations used to encrypt the text so you can decipher it.
* Frequency analysis (using statistical analysis 🡪 distributions) because of the human language syntax makes pruning very easy to implement when trying to break monoalphabetic ciphers.
  + Question: is there a way to make cipher text output more uniform?
    - Maybe impossible 🡪 which is why frequency analysis is (seemingly) unbeatable).
* The Hill Cipher is very fast because it uses matrix operations 🡪 but that also means it can be subject to brute force attack (except for when you use very large matrices to rule out people with enough processing power to decipher by brute force).
  + But, you can collect Plaintext and Ciphertext pairs and calculate a possible key and try the key (so the more P and C you have the better chance you can calculate the key).
* The remaining 8 bits not used for the key in DES can be used for parity (data integrity check).
* MIT project showed that they could obtain DES key within a day via brute force attack.
* AES introduced to replace DES
  + Symmetric key: same key to encrypt and decrypt.
  + Diagram

    Description automatically generated
  + Diagram

    Description automatically generated
  + For 128-bit AES, each w\_i = 32 bits, but not treated as bits but as a 32-bit integer.
    - So the 128-bits is treated as 4 32-bit integers.
  + Graphical user interface, text

    Description automatically generated
    - So, it appears that developers AES has found a way to make the encryption output uniform so that it is very difficult to do frequency analysis on – except for one issue: the same plaintext always generates the same cipher text.
    - If you change one bit in the key (a bit has 50% being 1 or 0), it has an avalanche effect – so that if you have a 256-bit key, then you would nearly have to actually do 2^256 permutations to find the key.
  + The one vulnerability of AES is the lookup table that has to be stored somewhere.
  + Also the same plane text will always give the exact same cipher text: AES becomes not secure.

# Lecture 6 – Cryptography (lec5 slides) (cont’d)

* IV only purpose is to not leak information; therefore it does not need to be kept secret
* AES mode: CBC (Cipher Block Text) fixes the information leak issue of AES by adding in an initialization vector (IV) that is randomly generated and added to plaintext, so that identical plaintext texts will not produce the same cipher text value.
  + Problem with encryption: later input depends on previous input, so you cannot run the algorithm in parallel.
  + Decryption however can be parallelized.
  + Requires padding: cipher block must be 128-bit blocks (because it uses a matrix that takes 128 values and does transformations on it), so any unused bits out of a 128-bit block must be filled (padded) with 0s.
    - Usually, padding only needed for last block, since last block of 128 bits of the data likely cannot be split into exactly 128 bits.
* AES mode: CFB (cipher feedback mode)
  + No padding required.
  + Note, this symbol:
  + Diagram

    Description automatically generated
  + Means XOR operation.
* AES mode: OFB mode (output feedback mode)
  + Cellphones use OFB and CFB mode.
  + Encryption and decryption can be done in parallel; encryption of IV with block cipher key can be done offline for several blocks so when an actual block of data comes in it is ready to be XORed with several pre-executed IV cipher texts.
    - Diagram

      Description automatically generated
    - Also, notice above, next step does not depend on plaintext from previous step.
* AES CTR mode (counter mode):
  + Encryption and decryption can be done in parallel.
* Public Key Cryptography (RSA)
  + Public Key and Private key are inverse algorithms of each other: so private key can be used to encrypt while public is used to decrypt; OR, private key can be used to decrypt while public key used to encrypt.
* Euler’s theorem
  + “relatively prime to one another” = numbers that only share a common factor of 1. (i.e., gcd(a, b) = 1.
* Extended Euclidean Algorithm
  + RSA uses this algorithm.
* RSA key generation
  + Very expensive operation
    - Exponentiation, modulo, multiplication , division.
      * Costs a lot of clock cycles.
  + Thus, we do hybrid encryption: RSA + AES.
    - AES is very fast encryption algorithm
  + Parameters for the RSA key are sent SEPARATELY and encrypted using AES so that if message is intercepted it cannot be compromised since a person may only have 1 of the necessary parameters for RSA key.
* For digital signature: private key is used to sign (add data to end of a file and then encrypt) data so that receivers of the data know that the data is the correct version and has not been compromised. Cannot use ONLY Hash verification because then the data can be derived even if you only send the hash key: solution 🡪 hash the file, encrypt the hash value and then send it; then receiver will download the encrypted data, decrypt it AND THEN hash it, and they should get the same hash value as the hashed file+signature value that was hashed, encrypted, and sent.
* HTTPS and TLS (TLS used to be SSL) protocols uses AES and RSA encryption algorithms.
* NOTE: We need RSA to do AES secret key exchange because if AES key is intercepted, then AES fails completely to be secure.
* Graphical user interface, application

  Description automatically generated with medium confidence
* Hash MACs (HMACs)
  + Used to prevent man-in-the-middle attacks in network applications.
  + Used on top of encryption to check for integrity between hosts without needing a middle man (but costs additional overhead).
  + Classic HASH requirements: collision free/resistant, one-way function (pre-image resistant).
  + SHA (secure hash algorithm) hash functions: SHA-2 SHA-256, SHA-512, etc.
    - For example, SAH-256 means the output is 256-bits (takes an input that is twice the output value it gives).
  + Hash function has avalanche effect: if even one bit is changed in the input, output will be completely different.
  + Opad=output pad
  + Ipad= input pad
  + After Keyed-Hash MAC (HMAC) function gives its output (AES key was the input), that value is then encrypted using AES key.
* Transport Layer Security (TLS)
  + Can be implemented on top of TCP (and UDP, but most often implemented with TCP).
  + Heartbeat (a TLS protocol) allows sessions to stay alive: for example, if you have an HTTPS connection, you do not want to do the handshake every time you make a GET request, otherwise webpages would run very slow.
  + TLS Handshake protocol
    - Cipher suites = types and versions of hash and encryption that a client supports.
    - Typo under server hello; should say: “server random” not “client random”
    - Random values and pre-master key used to derive master key, and master key is used to generate the HMAC and AES key values.
    - All previous messages used in handshake exchange is hashed and encrypted by both parties and sent in the final message encrypted with AES session key to make sure there is no man in the middle. They both decrypt the message and hash their own message exchange and check it against the encrypted hash it received to ensure integrity (no man in middle). Note: client random and server random messages are sent for this purpose at the final integrity check.
  + Need PKI for key exchange to make sure a client is getting correct public key.
  + We use separate keys for client-side and server-side write and MAC keys (so 4 keys total); both client and server has access to all session keys of course; but in the event that one of the keys is compromised, at least the whole of the communication will not be fully compromised unless an attacker obtains all 4 keys.
  + Content type in the TLS header would be which TLS protocols are being sent (i.e., handshake, heartbeat, etc).
  + TLS fragments the data into “records” and then does the encryption on each block it fragments.
    - Remember: From TCP’s point of view, it fragments data into “segments”, but it cares nothing of the format of the data; it simply fragments it according to its own protocol.
  + COLLEY LINUX distro is best for traffic checking (includes Wireshark)

Public Key Infrastructure:

* Deals with how to verify a public key sent to a client is the correct public key for communication with the other client.
* Mallory will re-encrypt the data using Alice’s public key and send it to her so she can decrypt using her private key and never know there was a man in the middle (MITM).
* Root of trust: you have to put your trust in something/someone. It is unavoidable when it comes to security: there is no way to tell whether the public key actually came from the client you intend to talk to; requires a trust database (CA = Certified Authority).
  + CA’s signature: signed by CA’s private key.
    - Then you will use the trusted public key to decrypt a message a message to make sure it finds the CA’s signature.
    - Browsers can maintain a cache so this whole process does not have to be replicated often using some integrity check methods.
  + You can verify the correct domain (IP) where you check for the CA’s public key information by generating a random number and telling the purported client to place that random number in the domain for the client to go to proper domain and see if that random number has been added.

# Lecture 7 – Web security

* php was based off of old Pearl programming language.
* Developed by Linux developers
* ? in a URL means: arguments start here
  + each argument separated by & symbol.
  + This is how you send arguments through the URL.
* http and ftp are text-based (unsecure)

## HTTP and HTTPS

* https 🡪 s = secure; made secure by TLS (sort of sits between the application🡪HTTP and transport layer🡪TCP)
* tilde in URL refers to Linux-based system and refers to “home” directory just like how you can use tilde to go to home directory of the currently logged in user.
* Heard inside of HTTP request contains system/browser information so that the receiver of the request can tailor the connection and protocols based on the requesters specifications.
* GET request is not secure because often passwords and other important data is sent via GET requests which is embedded in the URL link itself.
* POST also has plaintext data which can be easily decoded.

## Cookies (bread crumb session authentication token value)

* Add state to web applications.
  + By default HTTP is stateless (messages are independent and do not know/are not aware/does not have a history of the other messages)
    - Each message treated independently at the protocol level
  + Cookies is not a part of HTTP because HTTP was designed with scalability so it does not get bogged down with too much overhead and applications can be written to opt into certain functions but not have to and can use bare bones HTTP with minimum overhead.
* Cookies provide statefullness (in the application layer; HTTP uses TCP so the connection can be maintained as established and not have to constantly reconnect, but from HTTP application layer’s view once a request is received and served that interaction “closes” and is not tracked as part of HTTP; so you need to use Cookies, which is another application that works with HTTP and is often ran by servers and web browser applications).
* Statefullness via cookies maintained by storing state data on the requester’s local machine and on the server’s machine.
* Can set lots of options on cookies (such as expiration date of a cookie, who has access to the cookie, etc.).
* HTTP supports the cookies in its HTTP header but it does not remember the information.
* You do not want to rely on “security by obscurity” by making the name and authentication values the same value; for example, cookies uses separate username and authentication value.
* So essentially, cookies act as a temporary password authentication token so a user/device does not have to continuously log in 🡪 cookies maintain the authentication for a time and thus maintain the session (like when you go to a website and you are already logged in (automatically by the cookie token)).

## Session fixation attack

* Craft a GET request to send anonymous token to many users so one of them eventually clicks the link (which automatically sends GET using the anonymous token the sent) and when the user logs in they will elevate the token).
* Solution: always overwrite anonymous token with a new token when elevating that anonymous token (new token will be sent to user who logged in).

## CSRF (cross-site request forgery) attacks

* URL side-effects
  + Attacker can send a get request but change the source of the getter.
  + They can do this in order to get cookie tokens f
  + rom a destination server, which stores user credentials.
* Websites need cross-site requests, but this can be exploited. How can we defend against this?
  + Need to use a second token: a secret/hidden token.

## Cross-site scripting (XSS): Stored XSS attack

* Pass in script code instead of ASCII code (comments, ratings, some text-based input site that receives its input from users) to a server.
* Stored XSS attack (can be used to perform crypto mining attacks and use a client’s resource to mine for crypto currency, because JavaScript supports Dynamic Programming whereby browsers are allowed to execute code from authenticated (trusted) pages):
  + Infect a server so it stores a script
  + User does GET and the infected server will send the script
  + Script will execute on the client’s machine because the client trusts the URL it has a session with and gives that site privileges to send messages and execute code from that server IP/Domain.
  + Side note: JavaScript is the primary web development coding language.

## Dynamic web pages (JavaScript)

* Can cause a lot of problems because the power that it provides for Web 2.0.
* Power needs to be confined.
* Same Origin Policy (SOP)
  + Browsers provide isolation for JavaScript via SOP.
  + Keeps sessions in a browser between different domains separated (keeps their code separated so one browser session does not affect the other browser sessions connected to different domains despite being from the same origin 🡪 IP/Client machine and browser).
* Browsers do not enforce strict parsing (to make sure code is not passed in when not expected) because no pages would work because most pages have loose ends/errors/issues/bugs (and also browsers would rather websites be blamed for issues and not the browser) so browsers do their best to load the page no matter what with a more limited page security once (application layer).

## Cross-site scripting (XSS): Reflected XSS attack

* Echo attack: takes advantage of server-client echo messages (such as searching a site 🡪client input is echoed back to them as well as the results).
* Instead of passing in legitimate search text, pass in code instead which will be echoed back to the user/client.
* XSS defenses:
  + Use opensource libraries that filter (sanitize) the input from users and consider past and current attacks.

## XSS v CSRF

* XSS 🡪Exploits client’s trust in a server (infected server sends malicious content).
  + Stored attack (infect server first): attacker goes straight to server, passes in their JavaScript code in hopes that server will not parse it and hope code will be embedded in the website (store it on the server) 🡪 then when a client visits the site, the infected server will exploit the client’s trust in the infected server.
  + Reflected (infect client first): craft a link that includes the JavaScript which a client becomes “infected” by when they click the crafted URL link with code embedded to send to server and then be echoed back to client 🡪 client trusts server so it will execute the code now that it is received (via an echo) from the server.
  + Both stored and reflected attacks essentially achieve the same goal: exploit client’s trust in a server in order that code will be executed on the client in their browser (usually via a browser that supports and is running JavaScript).
* CSRF 🡪 Exploits server’s trust in a client.

# Lecture 8 – SQL (Structured Query Language) Databases

* SQL databases are basically structures based on columns and rows (matrices) that allows you to manipulate values row and column wise through defined row and column relationships (that is what the language/syntax is built off of).
* Stack is only meant to contain data, however you can still inject code into that region which can be executed. Just like buffer overflow, you can use the same exploitative techniques to inject code into an SQL database.

## SQL countermeasures

* Problem: users pass in input that is not expected (i.e. code).
* Similar buffer overflow issue can occur in SQL databases.
* We need to sanitize and/or check the input to make sure code is not being passed in.
  + Sanitize = remove (or replace) characters on a defined blacklist.
  + You could do the opposite: whitelisting = make sure characters are on the safe list.
  + We could whitelist by sending input to safe (duplicated) environment to test it (see if the input is code or causes undefined behavior); if no issues then hash the value into a whitelist hash table to mark it safe and avoid reducing performance by constantly having to do this redundantly for already tested input.
    - Problem: if any statements in the database changes, you would need to nullify the whitelist hash table to be safe because all tested inputs would now have to be retested.
    - Possible solution: fuzzing (sending input to another duplicate database in the background and testing inputs and building the whitelist instead of waiting for input from users to build the whitelist and test inputs).
    - Another solution 🡪 use Prepared statements: decouple the code from the data to isolate the two to allow us to compile now before binding the data. In doing this, we can bind user input to a data type and insure it cannot be code. Binding causes slower performance. Even if a user passes in code, the input is treated as code already and bind as a string.
    - Performance is very important for databases (measured usually in Max or average queries per second that a database can maintain).

# Networking and the Internet

* During the cold war, US military wanted a communication network that could withstand attacks.
* It needed to be robust and have strong failover (redundancy).
* Primary structure: break up data into smaller chunks and can send them out of order. Data can be retransmitted. Called the “packetized approach”.
* Data is split into smaller chunks: “packets”
* Packets contains a header(meta data) and a payload (and also a trailer often times).
  + Payload = the actual data.
* A network connection structure is referred to as a network topology.
* Computing devices that are a part of the network are called “host nodes” (can be source and/or destination nodes).
* Routers in a network forward packets to these communication nodes.
* Computers that are in close proximity are in a LAN = Local Area Network.
* Internet = a collection of LANs = WAN (wide area network).
* AS = Autonomous System 🡪 a network or group of networks that communicate using routing protocols that allows inter and intra networks to autonomously negotiate communications and establish security and efficiency.
* AS can be a collection of segmented WANs and LANs.

## Internet Protocol Layers (layer 3 of the network stack)

* 5 layer model:
  + 5) Application Layer
  + (TLS lies here between Application and Transport layer)
  + 4) Transport Layer
  + 3) Network Layer
  + 2) Link Layer
  + 1) Physical Layer (Phy Layer)
* Physical layer (layer 1)
  + Moving bits
  + Pre amplify and de amplify signals
  + Modulate (translate signals from digital to analog and vice versa) signals.
  + Circuitry 🡪 clock.
* Link Layer (layer 2)
  + Concerned with grouping bits into a logical structure.
  + Transfers data between machines on a LAN.
  + i.e. Ethernet link layer standard.
    - Group bits into **frames**.
    - Identify port destination by 48-bit Media Access Control address (MAC address).
      * Burned-in address.
      * Requires universally unique addresses worldwide.
  + Has collision issues (distorted bits)
    - This is caused when data is transmitted by several hosts close enough in time that they collide and become destroyed or enhanced (destructive or constructive interference of light) so as to cause a distortion on the receiving end 🡪 at the network interface card (NIC) of the hosts.
  + Switch v Hub
    - Hub is a circuit that simply broadcast all messages it receives to all other ports.
      * Cheap, fast, but limited bandwidth (inefficient) because of collisions require a lot of waiting.
      * Faster in the sense that it is a pure circuit with no software circuitry.
    - Switch has some circuitry (such as RAM and a CPU) that allows it to run software to not have to broadcast every message it receives. Initially, switches act as a hub but converges (build MAC address to port table and learns network topology) to allow point to point communications.
      * Uses the 48-bit MAC addresses.
      * Every port has to have a MAC address burned into its NIC (network interface card).
      * The IEEE uses rollover method where first 24 bits are reserved for manufacturers; after a certain number of years, reserved addresses given to companies will expire and be assigned to another manufacturer.
  + Frame layout
    - Preamble (7 bytes) used for synchronization.
    - MAC dest
    - MAC src
    - Ethertype
    - Payload
    - Footer (trailer) CRC
  + ARP (address resolution protocol)
    - Find the hardware address (MAC address) of a given host and map it to the host’s IP address.
    - Once a device receives ARP response from a host, it caches the mapping for the IP and MAC associated with the host that responded.
    - Problem: ARP spoofing
      * Device responds to an ARP broadcast and claims it has the IP of a device that it does not have; and ARP does not provide any authentication.
      * Often, spoofing occurs as man in the middle between local hosts and the gateway router 🡪 the spoofer pretends essentially that it is the router and simply forwards all the traffic to the router so that the hosts can still get to the internet, but the man in the middle is able to see/copy all the data before forwarding it to the router.
* Network Layer (layer 3)
  + Move packets across the internet
  + Movement between LAN networks
  + Allows for hop-to-hop movement between router gateways.
  + Tasked with routing packets.
  + Nodes have 32-bit address (IPv4) or 128-bit address (IPv6).
  + Main protocol of network layer = IP (intern protocol).
* Transport Layer (layer 4)
  + Uses a 16-bit address (2^16 - 1 addresses since port 0 is not used)
  + Combination of IP + 16-bit port number to identify what applications data should go to for a given host.
  + TCP = transmission control protocol (connection-oriented protocol).
    - More overhead.
    - More security and data integrity and reliable transport.
  + UDP = user datagram protocol (connectionless, best-effort protocol).
    - Less overhead = much faster than TCP.
    - Used for real-time application.
* Application layer (layer 5)
  + Provide protocols for specific services.
  + Application layer protocols
    - HTTP
      * Uses TCP layer 4 protocol to support it.
    - SMTP
      * Also uses TCP layer 4 protocol to support it.
    - VoIP
      * Uses UDP layer 4 protocol to support it.
* Each layer adds its own header data (encapsulation) when data is traveling down the network layers to finally be transported on the wire.
  + Example:
  + Layer 5) 🡪 DATA
  + Layer 4) 🡪 add TCP header
  + Layer 3) Add IP header
  + Layer 2) add FRAME header + CRC trailer
    - CRC = cyclic redundancy check.
      * Very expensive.
      * Can check integrity of the frame and detect errors.
* De encapsulation occurs when data is travelling up the network layers and headers are torn off to final become raw data to be used by application layer.

# Lecture 10 – Networking and Internet (continued)

* Remember, L3 packets have TTL field in the header to limit total number of hops a packet can make before it is dropped from the network by one of the routers; each time it reaches a hop TTL is decremented and thus checksum needs to be recomputed each time.
* Remember networks are divided by subnets 🡪 done through an IP address and corresponding subnet mask.
  + Subnet mask will tell you which bits of an IP address are the network bits 🡪 remaining bits are host bits.
  + Network ID = A ^ B, where A is IP address and B is subnet mask network bits
    - Host bits = A XOR C, where C is subnet mask non-network bits (0 bits).
* ICMP (internet control message protocol)
  + Use mainly for dialogues for determining if a host is alive and to find the path taken by the packet to reach a target host.
  + Types of ICMP messages:
    - Echo request 🡪 asks destination to ack receipt of packet
    - Time exceeded (error notification that packet expired)
    - Destination unreachable: error 🡪 packet could not be delivered 🡪 probably host denied request or target machine is not alive.
  + Examples of ICMP functions:
    - Ping
    - Traceroute (tracert).
      * Traceroute function makes use of TTL field 🡪 sends echo requests with a incrementally larger TTL value and every time it receives a response it increases starting TTL for sending the next request until destination is reached 🡪 1st request.
        + Note, this does not guarantee however that each packet took the same path when it was able to reach a further hop.
        + Also note: the response messages will not necessarily take the same path.
  + ICMP can be used to perform DOS attacks by flooding networks with ICMP requests.
    - Need to use machine learning and look at the entire network stack (all layers) to detect these kinds of attacks so that a firewalls can drop these packets (such as checking source field or using a blacklist and data predictions or databases to decide to drop a given packet). Costs more overhead however.
    - Also, you could just temporarily disable ICMP packets to stop the attack.
    - DOS attacks in general are very difficult to stop, especially because IoT devices with limited resources and security are often hijacked to assist in the DOS attack.
  + ICMP can also be used for packet sniffing.
    - Listen to traffic on the network.
    - When ethernet frames are transmitted, every node connected to a hub (which is a single broadcast domain.
    - Switches have single broadcast (collision) domain for each port; but they often have a Spam Port to mirror all data sent on all other ports (for QoS and network troubleshooting)
      * Spam ports present vulnerability
    - Another spoof attack: The receiving end (attacker/spoofer) can set their NIC card to be configured in promiscuous mode to listen to all traffic in a network (for example, data sent from spam port of a switch).
      * Then you can use tools such as WireShark to listen to all network traffic.

## Transport Layer

* Allows applications (processes) within the same machine to share a single IP
  + Data segments are distinguished by a 16-bit port number.
* 2 main protocols: TCP (transmission control protocol) and UDP (user datagram protocol).
* TCP is the main protocol used for the internet.
  + TCP is connection oriented and reliable (all segments are acknowledged).
  + Segments tracked through sequence numbers so that out-of-order arrival of segments can be reordered in a sliding window protocol so that it can send data to application layer and so that receiver is not overwhelmed.
  + Example: segments 1 2 3 4 and 5 transmitted to a host who can only buffer 5 segments total, then cumulative ack=6 is sent be receiver saying it acknowledged all previous segments and is ready for segment 6 (the 6th segment).
  + For flow control with TCP, if one of the segments in the sliding window is not ACKed but rather host ACKs a missing segment in that window, then the entire window is retransmitted and then if host receives the segments it was missing it will then send cumulative ACK for the whole window.
    - This teaching of Dr. Bacha appears to conflicts with our learning form Dr. Song from CIS-427.
* Transport layer contains checksum, source and destination port numbers, communication flag bits (SYN, ACK, FIN, etc.) etc.
* TCP attacks (from transport layer perspective)
  + Random sequence number value is sent (and SYN bit set denoting a SYN segment is being sent for establishing connection in 3-way handshake 🡪 SYN = random sequence number)
    - Host responds with ACK = (random sequence number + 1).
  + We do not start sequence at 0 because but use a random number so connection establishment is not predictable and thus easier to attack.
  + Servers uses TCB (transmission control block) to maintain half-open connection (handshake is occurring with a client to try and establish a connection) 🡪 remember socket programming I did in CIS-427.
  + TCB queue maintains which connections are being established.
    - TCBs remain in queue for quite some time (40s = an eternity from a computer attack perspective) before finally the attempted connection times out/closes.
  + We can exploit this TCB queue with a DOS attack 🡪 The SYN flood attack
    - Goal is to exhaust TCB queue.
    - Attacker is going to spoof the IP using random IPs
    - Send as many SYN requests so that TCB queue becomes full of attack nodes that do not really intend to connect.
  + Netwox synflood on our Ubuntu VM is an attack tool we can use to test synflood on a server we can run on the VM.
* TCP reset attack
  + Exploits the TCP wave (the normal way is using the FIN bit for a formal, agreed upon close of connection between to machines using TCP for connection using the FIN bit).
  + Attacker eses the RST bit, which causes abrupt closing of a connection.
  + Allows a DOS attack.
  + How? 🡪constantly send spoofed RST TCP messages to deny actual host connected to the server from maintaining its connection.
  + Key thing that the attacker needs: the correct sequence number that the server expects 🡪 RST = correct\_sequence\_#.
    - attack will succeed also if a sequence number is sent that is within the receiver Window (increases chance of success at attacker guessing sequence number since it is out of a range).
* TCP session hijacking
  + A computer can have multiple established connections.
  + TCP relies on TCP signature to uniquely ID a segment associated with the right session.
  + A signature consist of a 4-tuple:
    - Source IP address
    - Dest IP address
    - Source port number
    - Destination port number
  + Question: what if an attacker spoof this signature?
  + Answer: server has no way to distinguish between the imposter and the real client.
  + Result: attacker can hijack the session; server will not know if the packet came from the real client of an attacker.
  + For this to work: it is critical still to have the correct (or within receiver window range) sequence number.
  + Attacker can then inject data into the server even if it guesses correct sequence number within range that can be buffered on server (via a socket where server is receiving, thus has a receiving window) 🡪 the injected data will be buffered on server until all other data is ACKed, then that data will be used, but the data could be from attacker who now has privileges on the server using privileges from the spoofed real client.
    - With session hijacking, we can send commands to the server with clients privileges (net cat on port).
* ICMP attack - The Ping Flood attack (DOS attack)
  + You can send echo requests to an IP which will echo reply (let’s say, a server)
  + You can adjust size to 4KB
  + An attacker could flood the server with ping requests (DOS)
  + The above is not an efficient way to implement this attack because you would need many spoofed IP addresses 🡪 we have a better way to do this…
  + Solution: Smurf Attack 🡪 broadcast
  + Attacker sends a spoofed broadcast ICMP echo request packet
    - Specify source to be that of the victim so that all of the machines who receives the broadcast will reply to that source (the victim) who will be flooded with replies.
    - Requires that you leverage a large network.
  + Note: broadcast ICMP echoes are sent to default gateway which then the router will broadcast the request to all machines on the network.
* UDP protocol
  + “fire and forget” JDAM drop a bomb on em don’t care where it land or if it hit the target.
  + Header fields: source port, dest port, length, checksum, data.

# Lecture 11 – Final lecture - Networking and Internet (continued) – DNS (domain name system)

* DNS provides IP 🡪 ASCII based name translation
  + DNS is hierarchical
  + UDP based
* We have ROOT servers, then TLD (top level domain, such as .com, .edu, etc), then 2nd-level domains
  + Each level is responsible to keep track of the branch directly connected to it at the next level.
* DNS Query Process
  + 1) local DNS cache (locally stored on the machine)
  + 2) local DNS server (such as ISP) (or local router first that is directly connected to ISP)
  + 3) DNS server on internet (starting at root DNS server and working down structure until you get a hit).
* More on local DNS resolution
  + Stored in /etc/hosts file on Linux 🡪 stores static IP to hostnames mappings
  + One of the default mappings in the file is: 127.0.0.1 localhost (loopback address)
    - so ssh localhost will work because host has the mapping.
    - Special note: add 127.0.0.1 doubleclick.net mapping in local DNS cache file in order to block/drop adds from the large advertising site. GET request won’t return anything since you specified the wrong IP address (on purpose).
* Note: www means “web server”; you could have for example 🡪 [ftp.example.com](ftp://ftp.example.com) instead of [www.example.com](http://www.example.com).
* DNS attacks
  + Man in the middle attacks.
  + DOS attacks.
  + DNS requests are in plaintext
  + Attack (modify) /etc/hosts file entries.
    - Could be part of code from some malware you unknowingly installed that exploits your machine to get root access in order to edit that file.
  + Also, we could attack the /etc/resolve.conf file and modify it so that if no local DNS hit, then your machine goes not to ISP DNS but to whatever address was placed in the resolve.config file.
    - Service with root access manages this file as well as other /etc/hosts files that require root access.
* Local DNS server attacks
  + DNS is fire and forget (UDP) so it is easier to attack
  + DNS requests are not encrypted
  + Thus they can be easily snooped (listened to).
  + We can do man in the middle against the client, or we can do DNS server cache poisoning against the DNS server.
    - Attacker machine sends the request and reply to the targeted DNS server in order to poison the cache.
    - Also as another attack: could do DOS attack by requesting lots of different addresses so that the cache is full of irrelevant cached values that true users actually need.
    - Query ID (randomly generated 16-bit integer) is used as a defense against DNS poisoning but attacker can still try to guess the query ID by sending multiple requests and responses.
    - Problem: if true response server wins the race, then you have to wait for a while for cache to be cleared to start doing the attack again; solution is to use Kaminsky attack.
  + The Kaminsky attack
    - Use invalid addresses to try to poison server since name is not valid so there is no race to beat the root server that gives a response to local DNS server that we want to poison.