Overview

Tuesday, April 4, 2023 3:05 PM

Attacks target:
- Confidentiality: protected data stays protected
- Integrity: can't modify protected data
- Availability: protected data can be served

Types of attacks
- Software bugs
- Hardware bugs
- Humans
- Unintended characteristics
    - Side channel, bad randomness

Thinking as an attacker
- Look at things as they are, not as others see them
- See the gaps and weaknesses in the security

Thinking as a defender
- Data to protect, and what properties to enforce?
- Who are the attackers, motivation? What kind of attack to prevent
- Countermeasures: cost vs benefit, technical vs non-technical
- Minimize complexity - high complexity more prone to failure

Threat Model: set of assumptions about the attacks that a security system is trying to protect against

Iterative Secure Design: Identify the weaknesses of the system and focus on correcting

X86, Instruction Format, Syntax, Endianness, Arrays

Tuesday, April 4, 2023 4:27 PM

Control a process = get the privileges of its UID

How to control a process?
- Send specially formatted input to process

X86
- Variable-length instructions
- Register poor: 8 reg, 6 GP

32-bit	16-bit	8 high bits	8 low bits	Description
EAX	AX	AH	AL	Accumulator
EBX	BX	BH	BL	Base
ECX	CX	CH	CL	Counter
EDX	DX	DH	DL	Data
ESI	SI	N/A	SIL	Source
EDI	DI	N/A	DIL	Destination
ESP	SP	N/A	SPL	Stack Pointer
EBP	BP	N/A	BPL	Stack Base Pointer

  EIP: program counter (instruction pointer)
- Memory layout:
  .text : machine code
  .data : initialized global variables
  .bss : uninitialized global variables
  heap : dynamic variables
  stack : local variables, function call stack
  env : environment variables (OS env vars), program arguments

- Syntax: opcode src, dst
  - Constants (to be loaded into a register) prefixed by $
  - Registers preceded by %
  - Indirection using ()
  - Examples:

```
sub $16, %ebx
movl (%esi) %eax

pushl %eax
popl %eax

jmp -20
call FOO // saves instruction pointer to the stack and jumps to the argument value
ret      // pops the stack into the instruction pointer
leave    // copy EBP to ESP and restore the old EBP from stack

lea (%ebx + 8), %eax // adds 8 to ebx and stores to eax
```

- Function calls:
  - Locals are organized into stack frames
    - Callees exist at lower address than the caller
  - On call:
    - Caller: Save %eip so you can restore control
    - Callee: Save %ebp so you can restore data
  - Implementation details are largely by convention

- Somewhat codified by hardware

X86 is Little Endian:

0x12345678 =

0x78	0x56	0x34	0x12

Arrays: written in the same order as expected

Buffer Overflow Attack

Thursday, April 6, 2023 4:09 PM

Idea:
In the following code:

```
#include<stdio.h>
#include<string.h>

int main(int argc, char *argv[]) {
    char buf[100];
    strcpy(buf, argv[1]);
    printf("Hello %s\n", buf);
    return 0;
}
```

=> 

If size of argv is more than size of buf, then strcpy will overwrite the EIP stored on stack, which allows use to jump to a value we define and start executing a code.

Uses:
- Denial of service (run some code which will crash)
- Control flow hijacking (run some code to hijack the control flow)

Example: Shellcode

Note: We don't know where these addresses will be

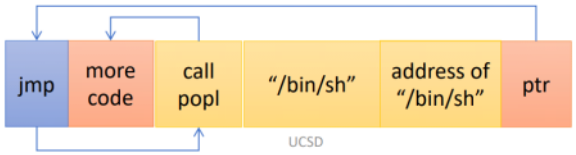
Core logic:

```
jmp    offset-to-call              # 2 bytes
popl   %esi                        # 1 byte
movl   %esi,array-offset(%esi)     # 3 bytes
movb   $0x0,nullbyteoffset(%esi)   # 4 bytes
movl   $0x0,null-offset(%esi)      # 7 bytes
movl   $0xb,%eax                   # 5 bytes
movl   %esi,%ebx                   # 2 bytes
leal   array-offset,(%esi),%ecx    # 3 bytes
leal   null-offset(%esi),%edx      # 3 bytes
int    $0x80                       # 2 bytes
movl   $0x1, %eax                  # 5 bytes
movl   $0x0, %ebx                  # 5 bytes
int    $0x80                       # 2 bytes
call   offset-to-popl              # 5 bytes
 /bin/sh string goes here.
empty bytes                        # 4 bytes
```

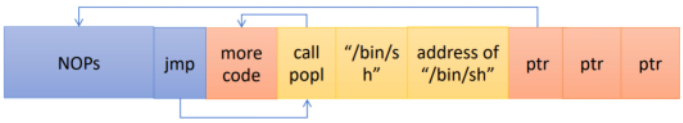
syscall 0xb = load a program
store pointer to "/bin/sh"
store pointer to pointer to "/bin/sh"
store pointer to NULL
interrupt 0x80
syscall 0x1 = exit
exit with code 1
interrupt 0x80
pushes "/bin/sh" pointer to stack

} loads the /bin/sh program into process memory

} safely exit the program to shell.



Problems:
- Code may not be the right size to overrun the buffer, pad with 0s
- Strcpy stops when it hits 0, use alternative machine code which avoids 0s
- How to know what value to override ptr? Try to leak some address in stack



- We can use a nop sled to make the arithmetic easier
- Land anywhere in NOPs, and we are good go
- Instruction "xor %eax,%eax" which has opcode \x90
- Can also add lots of copies of ptr at the end

What about small buffers? Just override ptr and jump to ENV

Defense:

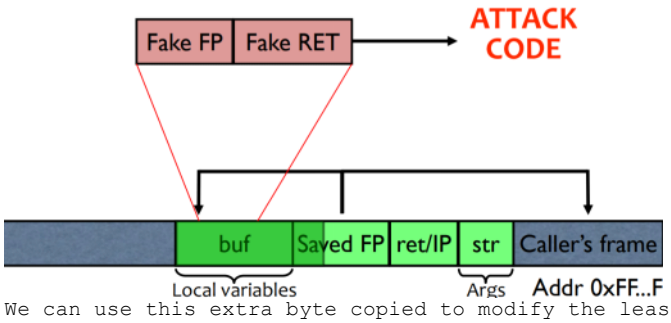
- Bounded buffer copies (strncpy, strncath)
- Sanitize input (only accept characters which are expected)

Format String Attacks

Tuesday, April 11, 2023 3:27 PM

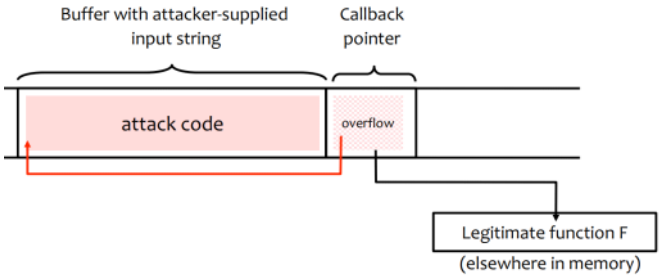
Idea: A poorly written program with an off by one error:

Frame Pointer Overflow

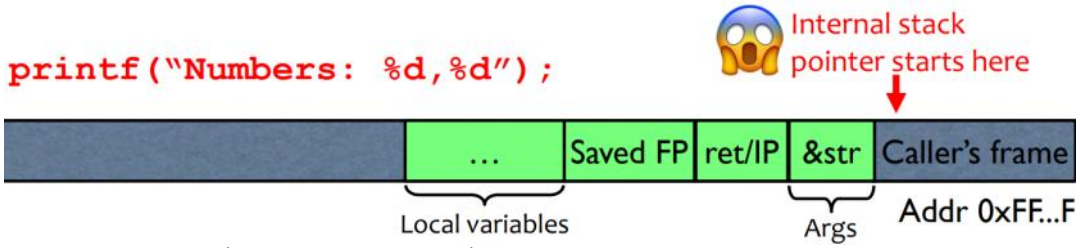


We can use this extra byte copied to modify the least significant byte of the saved FP so that the function returns to a fake frame.

Function Pointer overflow



Format String:
Idea: if you have printf(string), what happens if string contains a format character?



Can be used to print out the values in the caller's frame
We can use this to look at values in memory

Attacking: using %n with argument, printf will write the number of characters into the argument address
We can use this to write memory

- Use % to advance the internal stack pointer until we reach the caller's return pointer
- Make the input string the right size so that
- When %n is called, the length so far is copied into a byte of the return address
- Repeat 4 times until the return pointer is fully overridden

Integer Overflow

Thursday, April 13, 2023 3:52 PM

Idea: we can use integer overflow to control the size of buffers

Low Level Mitigation, Stack Canaries

Thursday, April 13, 2023 3:54 PM

Canary: push a canary value to the stack, compare value before and after function call to ensure the stack was not overwritten
- Also ensure that sensitive local variables are at the top of the stack, and buffers are near canaries
- We can chain vulnerabilities: exploit one vulnerability to read canary, exploit another to perform stack overflow
- We can brute force: keep guessing canaries until it succedes

Separate Control Stack: Control data is stored to a separate stack

Address Space Layout Randomization: Randomize addresses so that exploiters have a harder time to guess addresses

Write XOR Execute: writable pages should not be executable

Return to lib.c, Return Oriented Programming

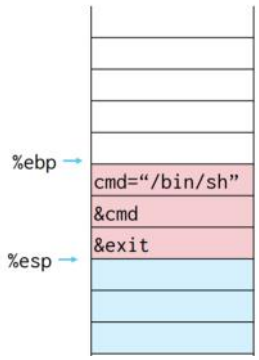
Tuesday, April 18, 2023 3:34 PM

Return to lib.c - Idea: lib.c contains the function system(), which executes the argument specified

Payload:

What we want to get to

- Transfer control to address of system() in libc
- Setup stack frame to look like a normal call to system()
  - **int system(const char *command);**
  - &exit() system call is in the slot where the return address would be
  - &cmd is the argument
  - It points to the string "/bin/sh" stored further down the stack



ROP - Idea: stitch together code using the last lines of functions just before return instruction
- ROP Gadgets: code sequences ending in ret instruction.
- Commonly added by compiler (at end of function)
- But also (on x86) any sequence in executable memory ending in 0xC3 (ret).
  - x86 has variable-length instructions
  - Misalignment (jumping into the middle of a longer instruction) can produce new, unintended, code sequences

Control Flow Integrity

Tuesday, April 18, 2023 4:16 PM

ROP and other return control protection: Only allow returns, calls, etc to go to known good targets

- Restrict control flow only to legit paths
  - Direct control transfer (known at compile time)
  - Indirect control transfer (depends on registers, can't predict)
- We note that we do not need to monitor direct control transfers
- Assign all indirect jump targets with labels

Operating System Security

Tuesday, April 18, 2023 4:44 PM

Trust boundaries: Each interface may have a different level of trust
- Processors: memory reference, privileged instructions
- Software: System calls, file accessors, etc.

Process Abstraction

Thursday, April 20, 2023 3:51 PM

Process abstraction: Abstract running program into processes, which have ID and permissions
- Isolation: Use permissions, paging and segmenting to protect process memory
- Process/kernel separation: privileged kernel operations can only be performed by the kernel

Unix user permissions:
- Permissions in UNIX granted by UID
- Each file has Access Control List (ACL)
    - Grants permissions according to UID and roles (owner, group, other)
    - Everything in UNIX is a file
- Use two IDs: Real UID, and Effective UID
    - RUID - typically the same as the user ID of parent process, determines which user started process
    - EUID - determines current permissions of process, used in security checks
    - Program can have setuid, which sets the EUID of the process temporarily to the file owner
        - Can use to do things like set passwd or call sudo
        - Allows user called process to temporarily elevate privilege

Process Isolation: limit processes to write only their own memory
- Virtual memory: abstract memory space for each process that only it can access
- Translate virtual addresses to physical addresses
    - Keep track of processes and their permissions for addresses
- Paging: translate blocks of addresses (usually 4KB or multiple)
    - Use a tree to store page mappings
    - Page descriptors describe how a process can access memory (read/write/execute)

Process/kernel separation:
- Privilege level: Increase processor privilege when kernel is running
    - Higher privilege allows process to perform sensitive actions
- Elevating privileges:
    - Prepare arguments including ID of desired entry point
    - Execute special instruction that initiates transfer
- System calls: expensive to flush TLB, page table base register, etc - but are used frequently
    - Map kernel virtual memory to every process memory, but is inaccessible when in user mode
        - Use privilege elevation to control kernel calls

Kernel Security and Exploits

Thursday, April 20, 2023 4:28 PM

Idea: Kernel memory and control flow should be protected from usermode processes
- Assume all usermode processes are untrusted and potentially malicious
- Avoid kernel being manipulated into abusing its privileges

Example: read(int fd, void * buf, size_t count)
- Reads count bytes from fd into buf
- What if buf is the address of a sensitive data structure in the kernel

Mitigation:
- copy_to_user(), copy_from_user(): Safely copy between user and kernel buffers, prevents values changing during execution
- Hardware: Instructions to mark pages as inaccessible to kernel or mark pages as inaccessible to processes

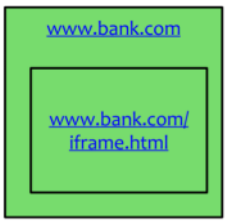
Web Security

Tuesday, April 25, 2023 3:27 PM

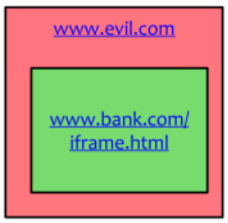
Browser security model:
- Sandbox: protect local system from web attacker
- Same origin policy: protect web content from other web content

Sandbox:
- No direct file access
- Limited access to OS, network, browser data
- Limited access to data from other sites
- Tabs and iframes (run in their own process)

Same origin policy:
- Only allow requests from the same origin = (scheme, domain, port)
- DOM: Only code from the same origin can access HTML elements on another site or iframe



www.bank.com (the parent) **can** access HTML elements in the iframe (and vice versa).



www.evil.com (the parent) **cannot** access HTML elements in the iframe (and vice versa).

- Cookies: Sites can only read/receive cookies from the same domain
- Scripts: included a script, it runs in the context of the site and has access to all HTML elements

Cross Origin Resource Sharing:
- Commonly: Access-Control-Allow-Origin: *
- Allows certain other origins to make requests, but commonly * used to denote any other origins

Plugins: enable functionality beyond browser sandbox, increases the attack surface

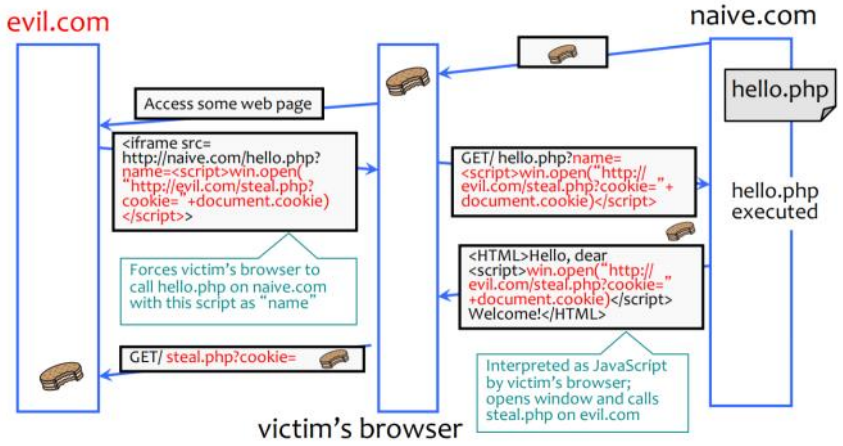
Extensions: extends functionality of the browser, use privilege separation and least privileges to contain malicious extensions

XSS

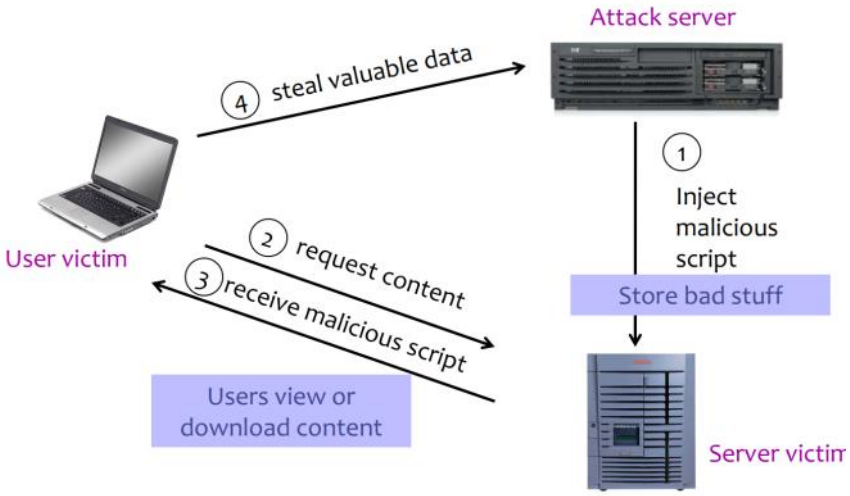
Thursday, April 27, 2023 3:45 PM

XSS: Site receives input from user and blindly trusts the validity of the input

Reflective - Use XSS against site to reflect malicious code back to the user:



Stored - Manipulate site directly and inject malicious code directly into site



Mitigation: sanitize or escape problematic characters or strings, tricky and workarounds often found

- Blocking "<" and ">" : event listeners
- Blocking certain tags : often required but can have js loaded
  -
- Blocking "javascript" : java\nscript
- Content security policy - Allow list for executable scripts

SQL Injection

Thursday, April 27, 2023 4:22 PM

SQL queries made by appending input data to query string - can inject other SQL commands as part of input

Ex: Login to database

```
set UserFound = execute(
    "SELECT * FROM UserTable WHERE
    username= ' " & form("user") & " ' AND
    password= ' " & form("pwd") & " ' ");
```

User supplies username and password, this SQL query checks if user/password combination is in the database

```
If not UserFound.EOF
    Authentication correct
else Fail
```

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database

User gives username ' OR 1=1 --

Web server executes query

```
set UserFound=execute(
    SELECT * FROM UserTable WHERE
    username= ' ' OR 1=1 -- ... );
```

Always true!

Everything after -- is ignored!

Mitigation: Validate inputs
- Filter out special characters
- Prepared statements: prevent inputs from being interpreted as statements

Prepared Statements

```
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, quantity, order_day "
            + "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParamenter("month")));
ResultSet res = ps.executeQuery();
```

Bind variable (data placeholder)

- Bind variables: placeholders guaranteed to be data (not code)
- Query is parsed without data parameters
- Bind variables are typed (int, string, ...)

Cross Site Request Forgery

Thursday, April 27, 2023 4:36 PM

Idea: We want to steal authentication such as cookies

- Users logs into bank.com, forgets to sign off
  - Session cookie remains in browser state
- User then visits a malicious website containing

```
<form name=BillPayForm
action=http://bank.com/BillPay.php>
<input name=recipient value=badguy> …

<script> document.BillPayForm.submit(); </script>
```

- Browser sends cookie, payment request fulfilled!
- Lesson: cookie authentication is not sufficient when side effects can happen

Network Protocols

Tuesday, May 2, 2023 3:53 PM

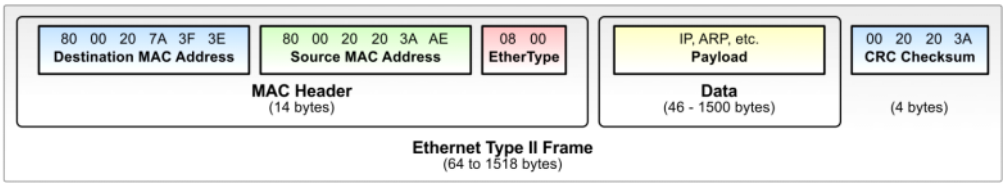
Syntax: how communication is specified and structured
- Format, order messages are sent and received

Semantics: What communication means
- Actions taken when transmitting, receiving

Packet: single unit of data, each layer will add headers to the packet

Stack of layers: define abstraction boundaries

Link: connects hosts to local network (ethernet)
- Messages organized into frames



- Every node has globally unique 6 byte MAC address
- Originally broadcast protocol, now is switched (wifi still broadcast)

IP: Connectionless delivery model
- Best effort: no guarantees about delivery
- Hierarchical addressing scheme

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note: there are no security measures except the checksum

ARP: Address Resolution Protocol
- Maps IP address to MAC address

BGP: Border Gateway Protocol
- Allows routers to exchange routing information to connect local networks together

TCP: Controls transmission of multiple packets
- Ports: identifies a specific application

UDP: Offers no service quality guarantees
- Essentially a transport layer protocol that is a wrapper around IP

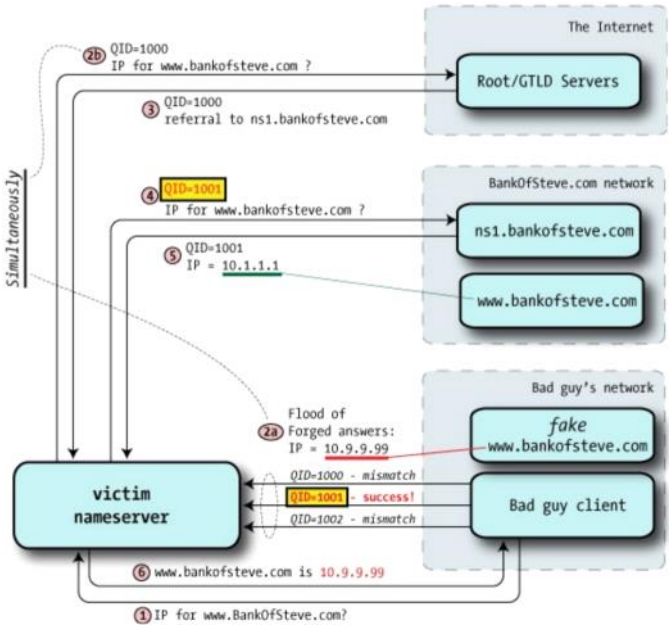
DNS: handles mapping between domain names and IP addresses

DNS Attacks

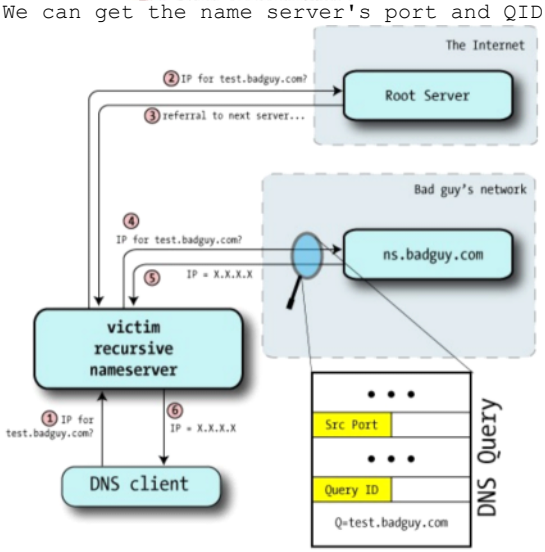
Tuesday, May 9, 2023 3:34 PM

DDoS attacks: Denial of service
- Take down DNS servers so clients can't use internet

DNS Cache Poisoning
- Send a DNS request
  - DNS server makes a request to Root server and gets a referral to nameserver
  - DNS makes a request to the name server
- Race the real name server in responding to the nameserver request
  - Guess the QID for nameserver request
  - Guess the source port for the request
  - Respond with the fake IP address
- DNS server receives the fake response and caches the fake IP

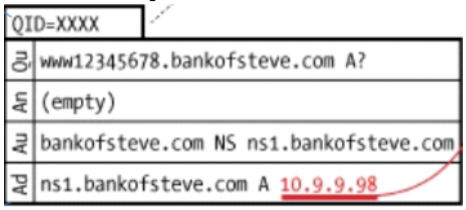


We can get the name server's port and QID range by using a test query to a name server under attacker's control



How to predict the query ID and source port?

Variation: poison cache for NS record instead:



Defenses:
- Randomize QID
- Randomize UDP port
- DNSSec: Cryptographically sign DNS responses, verify via chain of trust from roots on down

Phishing Attacks

Tuesday, May 9, 2023 4:03 PM

Pretend to be a trustworthy source

- Typo squatting:
  - www.qpple.com
  - www.goggle.com
  - www.nytmes.com

- Other shenanigans:
  - www.badguy.com/(256 characters of filler)/www.google.com

- Phishing attacks
  - These just trick users into thinking a malicious domain name is the real one

IP Prefix (CIDR), BGP, IP Hacking

Tuesday, May 9, 2023 4:08 PM

IP Prefix = CIDR: a.b.c.d/n means the first n bits are fixed. Effectively represents a range of addresses

BGP: Policy based routing across autonomous systems
- Each AS has an IP ranged assigned to it
- BGP routes traffic across AS graph, does not respond well to frequent node outages
  - Nodes broadcast packets to all connected nodes

IP (Prefix) Hacking:
- Any AS can advertise any prefix, attacker can advertise a more specific prefix and intercept BGP routing

  Ex:
  youtube = 208.65.152.0/22
  youtube.com = 208.65.153.238
  Pakistan ISP advertises 208.65.153.0/24
  Because Pakistan ISP is more specific, all traffic to youtube rerouted to Pakistan
  - Pakistan ISP broadcast more specific route across BGP, AS prefers this better path because more efficient

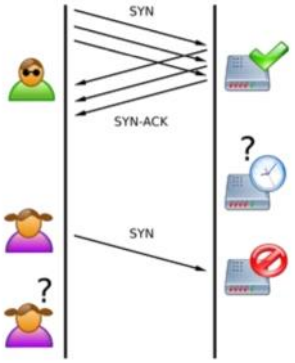
  Mitigation: BGPSec - cryptographically sign route announcements
  - AS can only advertise at itself
  - Cannot advertise for IP prefixes it does not own

IP Spoofing, ARP Security

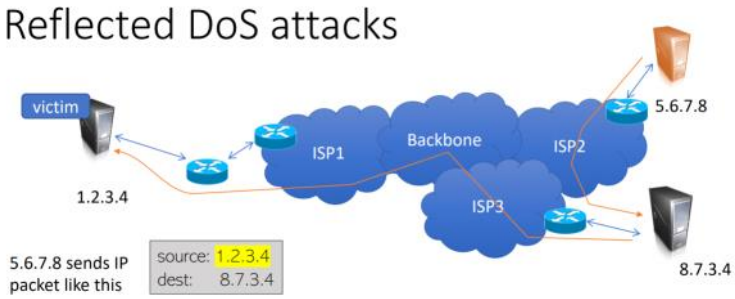
Tuesday, May 9, 2023 4:34 PM

Denial of Service: prevent users from accessing victim site
- Application: lock all user accounts by repeatedly guessing passwords
- DDoS: get pool of machines to send many malicious traffic
- Reflected: Send spoofed IP packets to random server who floods to victim with a lot of data

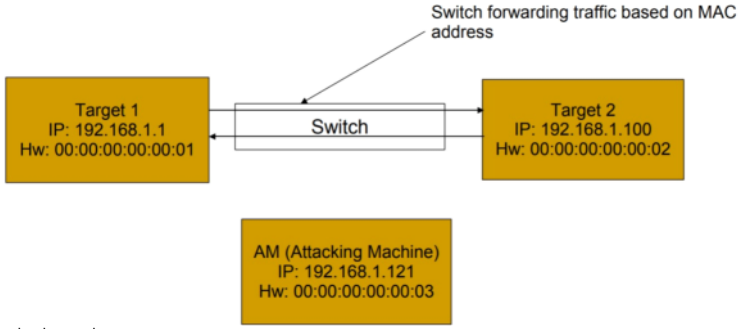
TCP SYN Flood: Attacker floods target with TCP SYN requests, but does not send ACK, target waiting for ACK but never comes



Reflected DoS:



ARP Poisoning: Intercept ARP requests to forward fake IP to MAC address mappings to forward packets to attacker's machine



Mitigation:
- Static ARP map for critical services
- ARPWATCH: logs ARP mapping changes
- Antidote: daemon service that monitors for unusually large ARP requests

General Mitigations

Thursday, May 11, 2023 4:10 PM

Firewalls:

- Personal firewalls
  - Run on end-hosts
  - Has application/user-specific information
- Network firewalls
  - Intercept communications from many hosts
- Filter-based
  - Operates by filtering on packet headers
- Proxy-based
  - Operates at the level of the application
  - e.g. HTTP web proxy

Access Control Policies:
- Default allow: permit all services, block specific ones
- Default block: block all services, permit specific ones known to be good

Packet-filtering firewalls can take advantage of the following information from network and transport layer headers:
- Source IP
- Destination IP
- Source Port
- Destination Port
- Flags (e.g. ACK)

Keep state on known open tcp connections: allows response packets to outbound requests

Circumventing:
- Iodine IP over DNS
- SSH Tunnel
- VPN

NAT: Network Address Translation

Idea: IP addresses do not need to globally unique

NAT maps between two address spaces

- NAT maintains a table of the form:
  `<client IP> <client port> <NAT ID>`
- Outgoing packets (on non-NAT port):
  - Look for client IP address, client port in mapping table
  - If found, replace client port with previously allocated NAT ID (same size as port number)
  - If not found, allocate a new NAT ID and replace source port with NAT ID
  - Replace source address with NAT address
- Incoming packets (on NAT port)
  - Look up destination port as NAT ID in port mapping table
  - If found, replace destination address and port with client entries from the mapping table
  - If not found, the packet should be rejected
- Table entries expire after 2–3 minutes of no activity to allow them to be garbage collected

Proxies: Man in the middle application
- Enforce policies for specific protocols:
  - SMTP: scan for viruses
  - SSH: Log authentication
  - HTTP: Block forbidden URLs

Network Intrusion Detection Systems: Passively monitor network traffic for signs of attack

- No understanding of higher level protocols
- Must be able to track packets of various multi packet connection standards
- Benefit:
    - No need to modify or trust end systems
    - Cover many systems with single monitor
    - Centralized
- Cons:
    - Expensive: 10Gbps = 1M packets/sec = ns/packet to check
    - Vulnerable to evasion attacks: incomplete analysis and imperfect observability

Log analysis: run scripts to analyze system log files

Vulnerability Scanning: probe internal systems and launch attacks on yourself

Honeypot: Deploy sacrificial system with no operational purpose

Cryptography

Thursday, May 18, 2023 3:22 PM

Symmetric Key Cryptography, MAC, Hashing

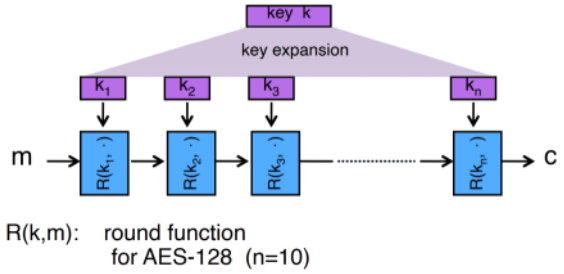
Thursday, May 18, 2023 3:31 PM

Idea: use the same key to encrypt and decrypt data
- Encrypt(key, plaintext) -> ciphertext
- Decrypt(key, ciphertext) -> plaintext
- Correctness: Decrypt(Encrypt(m)) = m
- One-time key: use key once per message
- Multi-use key: use the same key for multiple messages

One Time Pad:
- Generate key as long as plaintext
- ciphertext = plaintext XOR key
- Pros: theoretically information secure, ciphertext reveals no information about plaintext
- Cons: can only use key once, key must be as long as the message

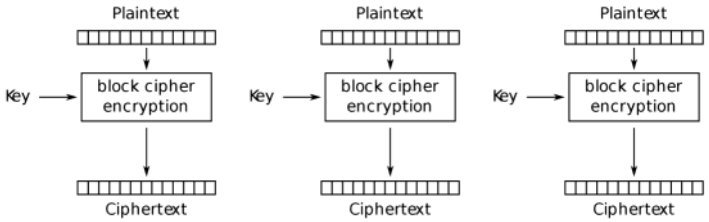
Stream Cipher:
- use pseudo random to generate a long key from a shorter seed
- Cons: can only use key once because attacker can generate messages from multiple uses of same key

Block Cipher:
- Operate on fixed block sizes with fixed size key

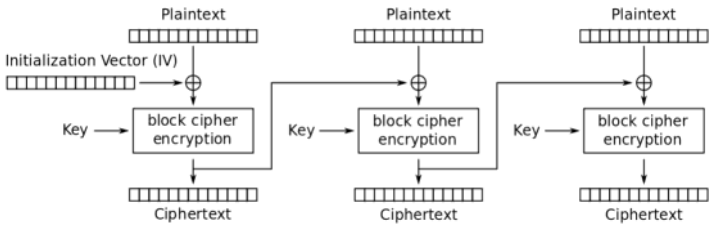


R(k,m): round function for AES-128 (n=10)

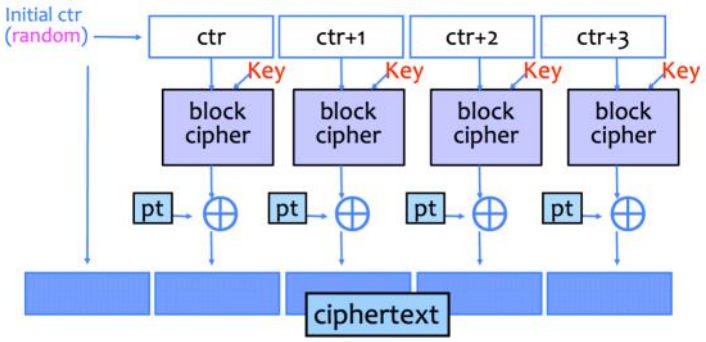
- Electronic CodeBook: Apply same operation on each block



- Cipher Block Chaining: Use previous ciphertext to encrypt next block



- Counter: use a counter as input to each block



- Identical blocks of plaintext encrypted differently
- Still does not guarantee integrity; Fragile if ctr repeats

Hashing: h(x) - Lossy compression function, map input message to output digest, should appear to be uniform
- One-way: Given y it should be very difficult to find $x \to h(x) = y$
- Collision resistance: should be difficult to find $x \neq x' \to h(x) = h(x')$
  - Birthday paradox means that brute-force collision search is only $O(2^{\frac{n}{2}})$, not $O(2^n)$
- Weak collision resistance: given an arbitrary x, hard to find $x' \to h(x) = h(x')$

- Complexity: $O(2^n)$

Common Hash Functions:

MD5:
- 128-bit
- Collision broken in 2004

SHA-1
- 160 bit
- Collision broken in 2017

SHA-256, 512, 224, 384

SHA-3

Message Authentication Code: Compute code when sending message, check code against received message to confirm integrity
- Use secret key to perform MAC calculation
- HMAC: Construct MAC from hash function, where K is the encryption key and message m
  || means concatenation

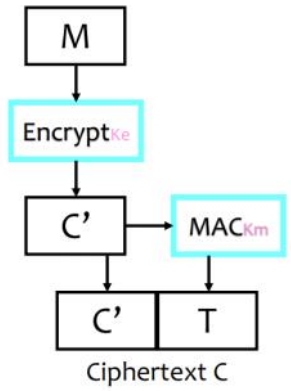
$$\mathrm{HMAC}(K, m) = \mathrm{H}\left((K' \oplus opad) \parallel \mathrm{H}\left((K' \oplus ipad) \parallel m\right)\right)$$

$$K' = \begin{cases} \mathrm{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

Ipad is block-sized inner padding, 0x36 repeated

Opad is block-size outer padding, 0x5c repeated

- Integrity and encryption: encrypt then MAC



Encrypt-then-MAC

Asymmetric Key Cryptography, Digital Signatures

Tuesday, May 23, 2023 4:18 PM

Problem: we want to establish a shared key safely with a recipient

Idea: Use two keys
- Public key: used to encrypt or verify
- Private key: used to decrypt

Diffie-Hellman:

Public parameters:
p a large prime number $\geq 2048\ bits$
$g = i \bmod p$ for some integer i

Alice chooses a number a and send $g^a \bmod p$ to Bob
Bob chooses a number b and send $g^b \bmod p$ to Alice
Alice computes $(g^b \bmod p)^a = g^{ab} \bmod p$
Bob computes $(g^a \bmod p)^b = g^{ab} \bmod p$

Textbook RSA: Generate public and private key using factoring and discrete log hardness

Attack: Malleability - Given c = Enc(m) = m^e mod N, forged ciphertext Enc(Ma) = ca^e mod N
Attack: Chosen ciphertext - Given c = Enc(m) for unknown m, attacker asks for Dec(ca^e mod N) = d and computes m = da^-1 mod N

RSA PKCS #1 v1.5: encrypter pads message

Digital signatures: want to attach signature to message to verify message

Sign(sk, m) = s = m^d mod N
Verify(pk, m, s) = true|false = m = s^e mod N

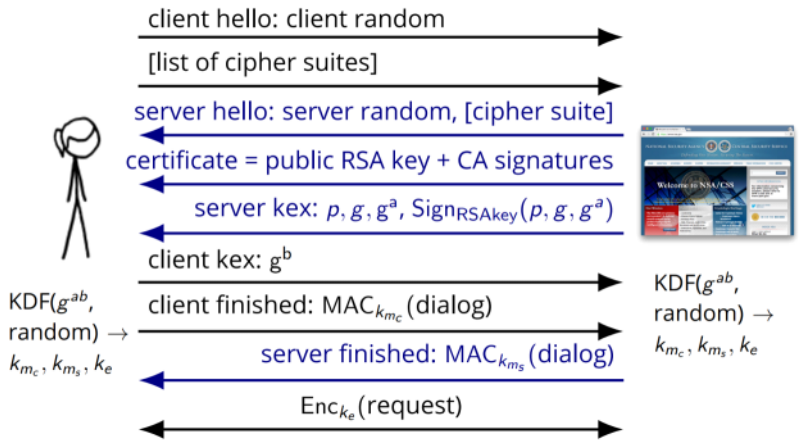
Sign by padding a hashed message

Transport Layer Security

Tuesday, May 30, 2023 3:31 PM

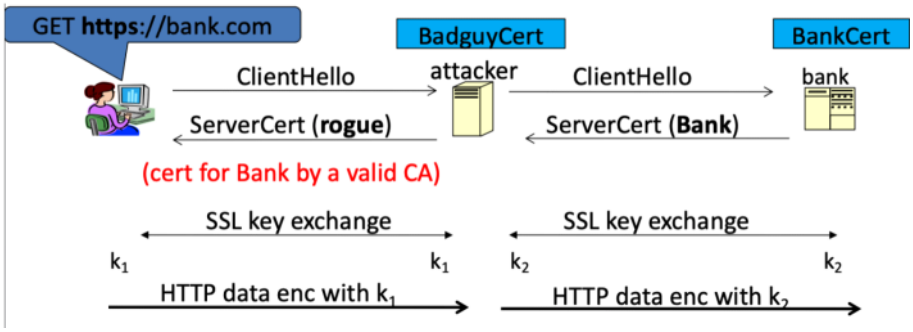
Provides confidentiality, integrity, authenticity

Steps:
- Negotiate with server to establish a key exchange protocol
- Use certificates to verify authenticity of server's public key
- Use Diffie-Hellman key exchange and Key Derivation Function (KDF) to establish symmetric keys
  - ke, km = KDF(g^ab)
- Use RSA signature to sign the two DH params and ensure authenticity of endpoint
  - Add nonces to avoid signature reuse



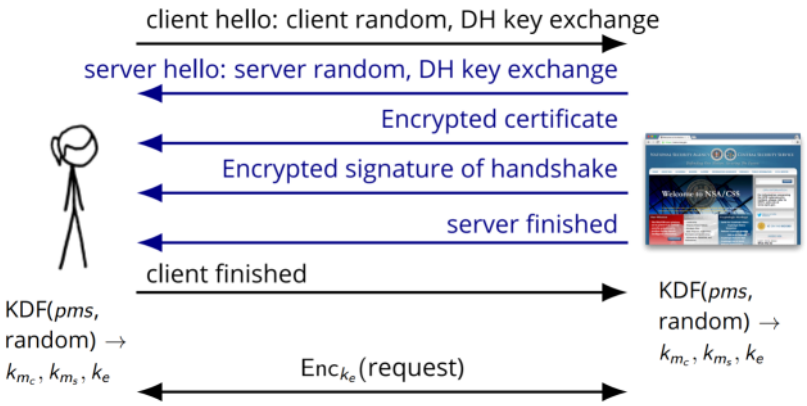
How to trust keys? Certificates
- Certificate Authorities: trusted intermediary
  - Verify public keys and sign them in exchange for money
  - Transitively trust keys signed by authority

Attack: Proxy connection using rogue certificate



Mitigations:
- Hard code popular websites certificates into browsers
- Certificate transparency - Public append-only log of certificate issuances to track rogue certs

TLS 1.3: Encrypt handshake after establishing key exchange


client hello: client random, DH key exchange
server hello: server random, DH key exchange
Encrypted certificate
Encrypted signature of handshake
server finished
client finished
KDF(pms, random) → k_{m_c}, k_{m_s}, k_e
KDF(pms, random) → k_{m_c}, k_{m_s}, k_e
Enc_{k_e}(request)

User Authentication

Thursday, June 1, 2023 3:30 PM

Prove to a computer who you are, use:

- What you know
  - Password
  - PIN
  - Security Questions
- What you are
  - Fingerprint
  - Hand geometry
  - Face
- What you have
  - Email
  - Phone number
  - TFA hardware

Password Based Authentication: User sets password, server checks request for equality
- Attack: Remote guessing attack
  - Untargeted: guess probably passwords against all user accounts
  - Targeted: target specific user by using tailored guess based on user's name, email, DoB, etc
- Problem: should not store passwords as plaintext
  - Hashing: hash password and store to database, compare hashes of request and stored hash
    - Can be brute forced by hashing common passwords and comparing to hashed passwords
  - Salting: add random value to end of password, random for each user
    - Attacker would have to guess common passwords with each user's salt
  - Pepper: Global secret stored separately from database
- Problem: Hashing is quick, attackers can brute force
  - Slow hash function to slow down brute force attacks
    - PBKDF2 - multiple iterations of hash function to slow down, still not slow enough!
    - Scrypt - requires a lot of memory to compute, slows down computation and increases cost of brute force hardware
- Problem: poor usability leads users to pick weak passwords and reuse passwords
  - Password managers: store passwords and encrypt using master password
    - Trust issues
    - Can be hacked
    - UI issues, does not work everywhere

Two Factor Authentication: Add an additional method of authentication
- OTP: send one time password through SMS, email
  - Insecure to SIM-swap attack
- Time Based OTP: Create OTP based on current time, sync server and TOTP using secret key

Biometrics: Use physiological or behavioral traits
- Types:
  - Faceprint
  - Fingerprint
  - Keystroke dynamics
  - Signature

- Universality: everyone should be able to use it
- Distinctiveness: should be unique to everyone
- Permanence: should not change over time
- Collectability: should be easy to measure trait
- Performance: should be easy to match
- Acceptability: should be acceptable (socially) to use
- Circumvention: should be hard to spoof or bypass

- Challenges:
  - Low accuracy: High false accept rate, high false reject rate

- Can't hash biometrics - fingerprint reading may change slightly each time
  - Need to encrypt data for biometrics
  - Can use TPM hardware support

- Attacks:
  - Spoofing: fake fingerprint, MasterPrint leverages error in matching algorithms

- Can’t be changed if compromised

Token Based Authentication: Use physical tokens to generate code
- Use public/private cryptography
  - Server & Client generate public key
  - Client generates private key
  - Server challenges login attempt to prove it has the private key

Mobile Security

Tuesday, June 6, 2023 3:25 PM

Attack vectors:

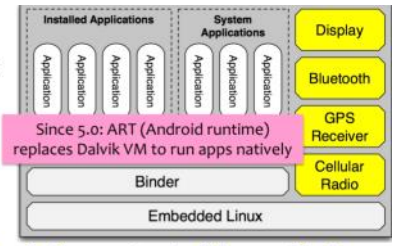
- Unique to phones:
  - Premium SMS messages
  - Identify location
  - Record phone calls
  - Log SMS
- Similar to desktop/PCs:
  - Connects to botmasters
  - Steal data
  - Phishing
  - Malvertising

Mobile platforms:

- Applications isolated:
  - run in separate execution context
  - No default access to file system, devices
- App store:
  - Vendor controlled
  - App signing: vendor issued or self-signed
  - Permissions are user approved

Android architecture:

- Based on Linux
- Application sandboxes
  - Applications run as separate UIDs, in separate processes.
  - Memory corruption errors only lead to arbitrary code execution in the context of the **particular** application, not complete system compromise!
  - (Can still escape sandbox – but must compromise Linux kernel to do so.) ← allows rooting



- Try to prevent attacks by limiting application access to resources

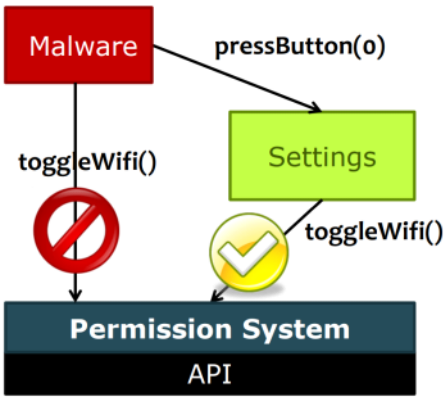
Rooting: leverage vulnerability in firmware to install su, can run programs as root

Permissions: Asked at use or asked at install

Permission Re-Delegation:

- An application without a permission gains additional privileges through another application.

- Settings application is deputy: has permissions, and accidentally exposes APIs that use those permissions.




```
Application Signing:
  - Often self-signed certificates
  - Signed application certificate defines which user ID is associated with which applications
      ○ Different apps run under different UIDs
  - Shared UID feature
      ○ Apps with the signed with the same key can declare a share UID
```

```
Other Mitigations:
```

- Address Space Layout Randomization to randomize addresses on stack
- Hardware-based No eXecute (NX) to prevent code execution on stack/heap
- Stack guard derivative
- Some defenses against double free bugs (based on OpenBSD's dmalloc() function)
- etc.