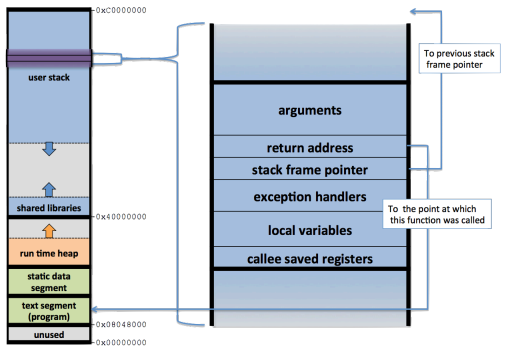
Computer Security Field Guide

Written by: Krishna Parashar
Published by: Atrus

Memory Safety

Memory Layout



All of the following uses the IA-32 (Intel 32-bit systems) Notation.
Instructions are formatted at `inst src dst`
Memory addresses are 4 bytes long
Registers are prefixed with %
Constants are prefixed with $
($exx) means accessing memory at register %exx
l suffix for instructions that are 32-bit (long) instructions
SFP is the saved %ebp on the stack
OFP is the old %ebp from the previous stack frame
RIP is the return address on the stack
The **text** contains executable code of program
The **heap** stores dynamically allocated data

The **stack** stores local variables/info for each func call

Registers

General Purpose Registers - %eax, %ebx, %ecx, %edx, %edi, %esi (%eax stores return value)
%ebp - base pointer, indicates start of stack frame (gen const for given function).
%esp - stack pointer, indicates bottom on stack (can and does change).
%eip - instruction pointer, indicates instruction to run.

Instructions

mov a, b - copies value of a into b
push a - pushes a onto the stack (decrements stack, copies value over)
pop a - pop data from stack on a (copies value over and increments stack)
call func - pushes address of next instruction onto stack and transfers control to func
ret - pops return address of next instruction onto stack and transfers control to func
leave - mov %ebp, %esp then pop %ebs (restores previous stack frame)
pushl %ebp is part of the prologue for instructions that move the stack pointer to the current top of the stack. You then call movl %esp, %ebp to move $ebp to where %esp is. You do this at the beginning of each function call.
You push arguments in memory in reverse order.
Indexes in array are stored with the highest index immediately under the SFP (which is under the return address, and lower values under that.

```
bottom of                                                             top of
memory                                                                memory
           buffer2       buffer1    sfp   ret   a     b     c
<------   [             ][        ][    ][    ][    ][    ][    ]

top of                                                             bottom of
stack                                                                  stack
```

C Specifics

`char` is 1 byte, `int` is 4 bytes, 1 byte is 8 bits
`void *memcpy(void *dest, const void *src, size_t n);`
`size_t: typedef unsigned int size_t;`
`size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)` Reads data from the given stream into the array pointed to, by ptr.
`strlen(s)` calculates the length of the string s, not including the terminating ”`\0`” character.
`strcpy(dst, src)` copies the string pointed to by src to dst, including the terminating ”`\0`” character
`strlcpy(dst, src, n)` avoids writing more than n bytes into the buffer.
`sprintf` works exactly like `printf`, but instead writes to the string pointed to by the first argument – terminates the characters written with a ”`\0`”
`snprintf(buf, sizeof buf, src)` is more secure than `sprintf(buf, src)`.
`fgets(char str, int num, FILE *STREAM)` is more secure than `gets()`.

You can access an element before an array with a negative index.

Buffer Overflows

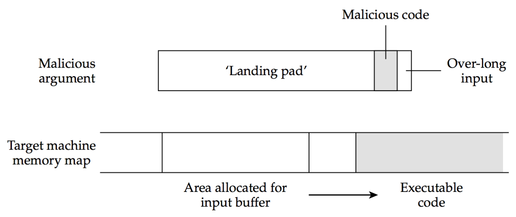
Simple Overflow

An input with 80+ bytes will overflow `buf`.

```
    char buf[80];
    int (*fnptr)();
    void vulnerable() {
      gets(buf);
    }
```

Stack Smashing

Half of all technical attacks on operating systems that are reported come from memory overwritting attacks or smashing the stack. The basic idea behind it is that programmers are often careless about checking the size of arguments, so an attacker who passes a long argument to a program may find that some of it gets treated as code rather than data.



Format String Vulnerability

This arises when a machine accepts input data as a formatting instruction (ex %n in the C command printf()). This can allow the string's author to write to the stack.

```
    void vulnerable() {
        char buf[80];
        if (fgets(buf, sizeof buf, stdin) == NULL)
            return;
        printf(buf);
    }
```

Should be printf("%s", buf). May crash/core dump. If attacker knows contects functions stack frame : %x:%x, reveals first two words of stack memory. %x:%s: treats next word as address and prints what is at that string.

Integer Conversion

If you don't check for data sizes before writing, the integer manipulation attack uses an overflow, underflow, wrap-arround or truncation that can result from writing an inappropriate number of bytes to the stack. C will cast negative value to large pos integer `memcpy()` will copy huge amount into `buf` causing overflow

```
    char buf[80];
    void vulnerable() {
        int len = read_int_from_network();
        char *p = read_string_from_network();
        if (len > 80) {
            error(”length too large”);
    return; }
        memcpy(buf, p, len);
    }
```

Defenses

Stack Canaries places a random value on the stack before return address, and then checks to see if that values overwritten, and if it is, it raises an erro. **ASLR (Address Space Layout Randomization)** starts the stack at random place in memory rather than a fixed point the idea that if someone is using hardcoded malicious code, ASLR can get around it, but you can get around it by instead of return to a hardcode address, return a relative address (known address + offset to where you want to go) to where you are on the stack right now.

In general make sure enough space is allocated, look for unchecked buffer writes, off-by-one errors (usually in indexing), inputs an attacker controls (they don't use a nullterminator or newline that you expect), malloc or calloc (initialized malloc) lengths. Keep tight bounds and restrictions for your program inputs.

Defensive programming means each module takes responsibility for checking the validity of all inputs sent to it. Use libraries to automatically detect potential bugs. And use **Static Analysis** tools which scan for potential memory leaks. False negatives are better than false positives. If code does not trigger any warnings, then it is guaranteed to be free of bugs. **Runtime Checks** automatically inject checks everywhere there is a chance of exploitation, which slows you down (10–150%) and can be tedious to run on legacy code.

```
    char digit_to_char(int i) { // BETTER
            char convert[] = ”0123456789”;
            if (i < 0 || i > 9)     # Check array access bounded
                return ”?”; // or, call exit()
            return convert[i];
```

Fuzz Testing

Random Given nothing, input into the program random values and see where it crashes.
Mutational Given a template as input, modify stuff (like bits) and run, to figure out which values can cause problems.
Generational Given some constraints, and fuzz those value to figure out which can cause problems.

Security Methodologies

Software Security

We want our systems to maintain **Confidentiality**, **Integrity**, and **Availability**.

Leveraging Modularity (privilege separation) is strengthening security by providing forms of isolation that is keeping potential problems localized, and minimizing the assumptions made between components.

Threat Models are the environments and context in which your security was designed. As time goes on they have a knack for changing, like what happened to the Internet (University Network to Now).

Kerkhoff's Principle is that Cryptosystems should remain secure even when the attacker knows all internal details of the system. The key should be the only thing that must be kept secret, and the system should be designed to make it easy to change keys that are leaked (or suspected to be leaked). Changing the key is easier than replacing software.

Access Control

A *Subject* tries to access an *Object* as is restricted by the *Policy*.
Authorization is who should be able to perform which actions. **Authentication** is verifying who is requesting the action. An **Audit** is a log of all the actions, so that there is **Accountability** to hold people to their actions.

In web we can have have **Centralized Enforcement** where the database centrally checks policy for each access. There is also **Integrated Access Control** which verifies the policy wherever there is data access. It is more flexible but perhaps more prone to error.

Trusted Computing Base

This is the minimum set of components to ensure security. TCB is one that is able to violate our security goals if it misbehaves. Ensure the security of your TCB. TCB must be large enough so that nothing outside the TCB can violate security. For example Authorized users allowed to enter system using SSH so the TCB is the SSH daemon, Operating System, CPU. A **Reference Monitor** is TCB specialized for access control. Design Principles for TCB are **Unbypassable** - no way to breach system security by bypass TCB (you have complete mediation), **Tamper-resistant** - protected from tampering from other systems and **Verifiable** - verify the correctness of the TCB. Use **Least Privilege** to only give each part of the system the minimum amount of access rights needed. Smaller systems are easier to secure. Isolate privileges to specific components as much as possible.

Race Conditions

Race conditions occur when a transaction is carried out in two or more stages, and it is possible for someone to alter it after the stage which involves verifying access rights.

TOCTTOU Vulnerability

Time-Of-Check To Time-Of-Use is a issue that arises when meaning of variable changed from the time when it is checked and the time when it is used such as in UNIX filesystem calls. Can arise anywhere there is mutable state that is shared between two or more entities.

Core Ideas

Security relies on the economic constraints, **Design security from the start**, give the **Least Privileges** possible, have **Fail Safe Defaults**, **Layer Security**, Ensure people **psychologically** accept security (shitty passwords), go for **Complete Mediation** (end to end control), know and track your **Threat Model**, ensure that if you can't protect you can at least **Detect** vulnerabilities, don't rely on **Obscurity** to keep your system safe, assume **system architecture** is known, try to plan for the **Worse Case** and **Study Attacks**. You are only as secure as the weakest link.

Web Security

The web allows for sharing applications and information between a client (your browser) and a source (a web server). One of the main ways to do this is through HTTP (Hypertext Transfer Protocol). The allows you to send requests and get responses.
URL (Uniform Resource Locator) such as
`http://atrus.co:2000/fg?search=security#web` is separated into
`protocol://hostname.tld:port/path?query#fragment`
HTML + CSS + JS = DOM –Painter–> Bitmap
The web is a good example of **bolt-on security**. Security model was added later.

We want **Integrity** where no malicious site should tamper with my computer of info on other sites (Sandbox JS, Access Control, Updates), **Confidentiality** none of my confidential info should be given to malicious sites (Same Origin Policy), and **Privacy** where malicious websites should not be able to spy on me or my activities.

Same Origin Policy

Stipulates that each site is isolated from other sites, where origin is protocol + hostname + port. One origin should no be able to access he resources of another origin. Cross origin communication is allowed through a API called `postMessage` where the receiving origin see the message's origin and can accept or deny message.

SQL Injection

These commonly arise when a careless web developer passes user input to a back-end database without checking to see whether it contains SQL code and doesn't escape `'+–;`. You can get around it using frameworks such as ParameterizedSQL that sanitize input. You can also use **Prepared Statements** which are preset allowed commands. Exploits like the following allow you to see or remove all the users.

```
    ok = execute(SELECT ... where user=' ' or 1=1 --)
    ok = execute(SELECT ... where user=' '; DROP TABLE Users)
```

Cookies

Since good HTTP is stateless (no info is maintained between requests and requests are independent), we use cookies. Browser stores cookies that web sites ask to store a set of key value pairs such as (domain, when to send), (path, when to send), (secure, SSL), (expires, time). **HTTPOnly** attribute means no JS access. Cookies can be accessed by parent domains (except TLD), but they cannot set cookies for their subdomains. Sessions allow you do set a time period for authentication and are tied to user. Tokens are generated by the server for a set tie period. Session Cookies send the cookie with every request and are thus vulnerable to CSRF, putting the token in URL can leak the token, and putting it in hidden form fields only work for short sessions. Thus we use a combination of these for optimal effect.

Privacy

A third party cookie, say from an advertiser that has a relationship with a website, can follow your activity and thus form a trail of you. Google Analytics sends a ton of data from websites to their platform. Facebook can track all the sites you are on if that site has a like button. This all makes Internet content free (ads), but there is concern how easy it is to get your data and we don't know how it can be misused. Plus you don't really know what you are giving away (image location data). Don't trust private browsing, cookies are still used within session.

Cross Site Scripting (XSS)

Very common web vulnerability. Same origin policy does not protect against XSS. Two main types:

- **Stored XSS** is when an attacker leaves malicious JS on a server that is then sent to a client (subverts SOP) which then does something malicious to the client. For example executing a URL that sends me money through a request. You can steal cookies and operate on session cookies. Example is storing HTML/JS on a Database that doesn't escape it like Samy's MySpace worm.

- **Reflected XSS** is when a user is tricked into clicking a malicious link, submitting a specially crafted form, or browsing to a malicious site, the injected code travels to the vulnerable website and the web service reflects it back. For example stealing a cookie by "searching" a script that requests it form the server (`document.cookie`) and sends it. This works when there is no form validation (such as escaping HTML tags).

You can protect against these by having a whitelist of accepted scripts (CSP) and disallowing all other scripts. Use proper form validation.

Cross-Site Request Forgery (CSRF)

Is when a session cookies remains in browser state after a user successfully authenticates. Malicious scripts from specific sites can ask for the cookie and forge requests with full authentication (such as sending money using a payment service). To combat this we use **secret validation tokens** (secret tokens that server looks for to make sure request is coming from authorized source). Often we use the user's **session ID** for validation, but an attacker can impersonate these if they can see the content of the page. We can also use a **nonce** (single purpose random number) that is sent with cookie and is hidden but they can be replaced by an attackers CSRF token. Else we can use a **session dependent nonce** which binds the user's CSRF token to their session ID, but this requires the server to maintain a table of these pairs.
Referrer validation is when includes a referrer header that indicates where the request came from. This can be used to differentiate cross site requests. You have to set the tolerance for which referers to allow. There are also privacy concerns with being able to see referer header URL such as search queries. **Custom Headers** allow for sites to send headers amongst themselves, and since they are custom they don't even need cookies values, the server just has to reject requests without the custom header. They are limitations, such as data export format and it not working across domains.

Authentication

Server should authenticate client (Passwords, Key, Fingerprint) and client should authenticate web servers (Certificates). Two-factor authentication is using two of (Knowledge, Possession, Attributes). After authecication as session cookie with my session ID is stored on my browser and the webserver maintains a list of active sessions. HTTP uses the cookie to authenticate every request automatically. Vulnerable to CSRF attacks.
Phishing is when an attacker fakes a site to trick a user into giving up their credentials or data. Prevent this by checking the URL and certificate. Don't click on unknown links.

UI Exploitation

Clickjacking is creating a link that seem sto link to one website, but with some tricky (usually using JS) can do things like like something on Facebook or redirect to another URL.
iframes can be used to link one website within another, which can cause one site to appear like the normal one but can have a layer of maliciousness on it such as stealing userid and password. Frames follow same origin so you can't run malicious JS on the content of the frame, but you can use clickjacking to add a partial overlay to make it seem like you are entering your info in the right place. This compromises the **visual integrity** of the page (things look different than they are). You can also use fake cursors to trick users into clicking the wrong thing. User confirmation while bad UX helps to alleviate these attacks. UI Randomizations are good as well but are susceptible to multi-click attacks. Removing cursor customization (16% attack rate), freezing screen around the pointer (4%), adding a lightbox (3%), or delay (1%), or disabling clicks unless pointer is over target (0%).
Framebusting is code on a website that prevents other sites from framing it, usually with a conditional statement and a counter action such as
`if (top != self) { top.location = self.location; }`. However these are often defeated due to small bugs in the framebusting code (like not using HTTPS) or with a XSS filter that removes the scripts from the page. We can also use X-Frame-Options which either deny rendering in framed context or only render if the top frame is the same origin as framed page.

Cryptography

Confidentiality Prevent adversaries reading data, **Integrity** prevent attackers from altering some data, **Authenticity** determines who created a given document.

Attacks

Ciphertext Only is Eve intercept encrypted message and wants to recover plaintext.
Known Plaintext is when Eve intercept encrypt msg and has partial info about plaintext
Chosen Plaintext is when Eve trick Alice to encrypt messages $M_1, M_2, \ldots, M_n$ of Eve's choice. Eve can observe the resulting ciphertexts. Try to recover new message M.

Chosen Ciphertext is when Eve can trick Bob into decrypting some ciphertexts $C_1, \ldots, C_n$ and then use this to learn decryption of some other ciphertext C. Combining the last two is a very serious threat.
Block Ciphers Encrypt with $E_K(M) = E(K, M)$; **Decrypt** with $D_K(E_K(M)) = M$
Symmetric Key Crypto is where the same secret key us used by sender and receiver and is reusable, it is built on block ciphers.
Public Key Crypto is where sender and receiver use different keys.

One-Time Pad (OTP) can only be used once. Alice and Bob share n-bit secret key. (XOR of the same bit is 0 and different bits is 1) **Encrypt** with $c_j = m_j \oplus k_j$. **Decrypting** is done with $m_j = c_j \oplus k_j$

$$\begin{aligned}\Pr[b=0|\text{ciphertext}=C] &= \frac{\Pr[b=0 \wedge \text{ciphertext}=C]}{\Pr[\text{ciphertext}=C]}\\ &= \frac{\Pr[b=0 \wedge K = M_0 \oplus C]}{\Pr[\text{ciphertext}=C]}\\ &= \frac{1/2 \cdot 1/2^n}{1/2^n}\\ &= \frac{1}{2}.\end{aligned}$$

Cryptanalysis Against Public-key Cryptosystems

Chosen-Plain Text Attack (CPA)

- An attack chooses a plaintext messages and gets encryption assistance to obtain the corresponding ciphertext messages. The task for the attacker is to weaken the targeted cryptosystem using the obtained plain-text pairs.
- The attacker has an encryption box.
- All public key encryption systems must resist CPA otherwise it is useless.

Chosen-Ciphertext Attack (CCA)

- An attacker chooses ciphertext messages and gets decryption assistance to obtain the corresponding plaintext messages. The task for the attacker is to weaken the targeted cryptosystem using the obtained plaintext-ciphertext pairs. The attacker is successful if he can retrieve some secret plaintext information from a ”target ciphertext” which is given to the attacker after the decryption assistance is stopped. That is, upon the attacker receipt of the target ciphertext, the decryption assistance is no longer available.
- The attacker is entitled to a conditional use of a decryption box. The box turns off before the ciphertext is sent.

Adaptive Chosen-Ciphertext Attack (CCA2)

- This is a CCA where the decryption assistance for the targeted cryptosystem will be available forever, except for the target ciphertext.
- The attacker has the decryption box for as long as he wishes, except they can’t decipher the original message.

RSA Cryptosystem

- Key Setup
  1. Choose two random prime numbers p and q (typically done by applying a Monte-Carlo prime number finding algorithm.
  2. Compute $N = pq$
  3. Compute $\phi(N) = (p-1)(q-1)$
  4. Choose a random integer $e < \phi(N)$ such that $\gcd(e, \phi(n)) = 1$, and compute the integer d, such that,
     $$ed \equiv 1 (\texttt{mod}\ \phi(N))$$
  5. Publicize (N,e) as the public key, safely destroy p, q, and $\phi(N)$, and keep d as the private key.
- Encryption
  To send a confidential message $m < N$ to Alice, the sender Bob creates the ciphertext c as follows,
  $$c \leftarrow m^e (\texttt{mod}\ N)$$
- Decryption
  To decrypt the ciphertext c, Alice computes,
  $$m \leftarrow c^d (\texttt{mod}\ N)$$

Symmetric Key Encryption

Poly-Alphabetic Ciphers

- Vernam Cipher
  - Message is m bits and the key is n bits.
  - Bitwise xor the message and the key, if m is greater than n, then use the key multiple times.
- One-Time Pad
  - Same idea as the Vernam Cipher except we use a key that is the same length or greater than the length of the message, then discard it after each use.

Advanced Encryption Standard (AES)

AES is a block cipher with variable block size and variable keysize. (block size can be 128, 192, 256 bit)
AES has 4 states:

1. Sub Bytes State: nonlinear substitution on each byte
2. Shift Rows State: Transposition rearranges the order of elements in each row
3. Mix Columns State: Polynomial multiplication after converting column to polynomial.
4. Add Round Key State: adds elements of round key to the state, basically bitwise ”OR”

Decryption is the inverse of these steps.

Advanced Encryption Standard (AES)

AES is a block cipher with variable block size and variable keysize. (block size can be 128, 192, 256 bit)
AES has 4 states:

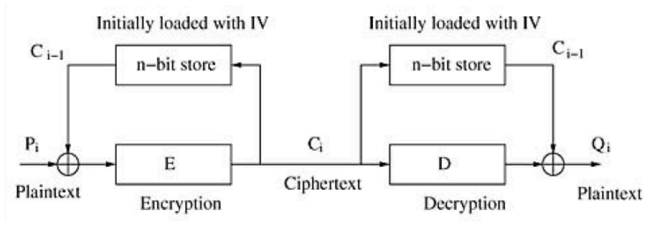
1. Sub Bytes State: nonlinear substitution on each byte
2. Shift Rows State: Transposition rearranges the order of elements in each row
3. Mix Columns State: Polynomial multiplication after converting column to polynomial.
4. Add Round Key State: adds elements of round key to the state, basically bitwise ”OR”

Decryption is the inverse of these steps.

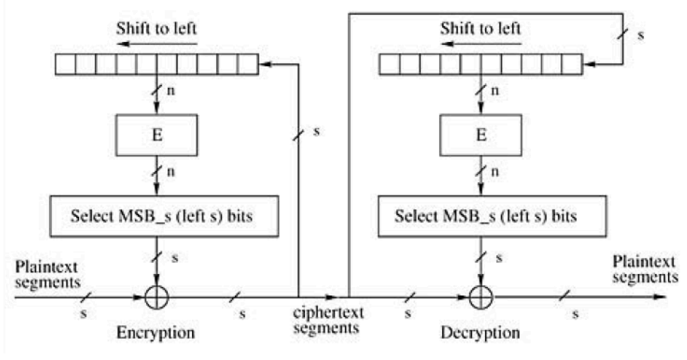
Confidentiality Modes of Operation

Different modes of operation have been devised on top of an underlying block cipher algorithm

- Electronic Codebook (ECB) Mode This mode encrypts and decrypts every block seperately. It is deterministic and leaves patterns in the cipher text. (for example images.)
- Cipher Block Chaining (CBC) Mode
  - This is the most common mode of operation. In this mode the output is a sequence of n-bit cipher blocks which are chained together so that each cipher block is dependent on all the previous data blocks.
  - Decryption can be done in parallel
  - CBC cannot prived data integrity protection.
  - If the CBC claims data integrity protection, Eve can use (Bomb Oracle Attack) a Decryption Oracle to figure out the padding scheme and eventually the last byte of the cipher text.



- Cipher Feedback (CFB) Mode
  - CFB mode of opration features feeding successive cipher segments which are output from the mode back as input to the underlying block cipher algorithm.
  - CFB requires an IV as the initial n-bit input block



- Counter (CTR) Mode
  - The CTR mode features feeding the underlying block cipher algorithm with a counter value which counts up from an initial value. With a counter counting up, the underlying block cipher algorithm outputs successive blocks to form a string of bits. This string of bits is used as the key stream of the vernam cipher, that is, the key stream is XOR-ed with the plaintext blocks.
    $C_i \leftarrow P_i \oplus E(Ctr_i, i = 1, 2, \ldots, m$
    $P_i \leftarrow C_i \oplus E(Ctr_i, i = 1, 2, \ldots, m$

Public Key Encryption

Diffie-Hellman Key Exchange Protocol

`Common Input`	$(p, g) : p$ `is a large prime, g is a generator element in` F_p^*
`Output`	`An element in` F_p^* `shared between Alice Bob.`

1. Alice picks $a \in U(1, p-1)$; computes $g_a \leftarrow g^a (\texttt{mod}\ p)$; sends g_a to Bob.
2. Bob picks $b \in U(1, p-1)$; computes $g_b \leftarrow g^b (\texttt{mod}\ p)$; sends g_b to Alice.
3. Alice computes $k \leftarrow g_b^a (\texttt{mod}\ p)$

4. Bob computes $k \leftarrow g_a^b (\texttt{mod}\ \ p)$

Alice and Bob both compute the same key,

$$k = g^{ba}(\texttt{mod}\ \ p) = g^{ab}(\texttt{mod}\ \ p)$$

P is a public 2048 bit prime number.

RSA Euler's Theorem

$m^{(p-1)(q-1)} \equiv 1(\texttt{mod}\ \ pq)$

Insecurity of Textbook RSA Encryption

Eulers Theorem

given a,n are coprime,
$a^{\phi(n)} = 1(\texttt{mod}\ \ n)$

Fermat's Little Theorem

given a coprime to N,
$a^{(N-1)} = 1(\texttt{mod}\ \ N), a \in \mathbb{Z} < N$

- The RSA Cryptosystem is "all-or-nothing" secure against CPA if and only if the RSA assumption holds, meaning if the attacker has some prior knowledge of the contents of a message (ex a number or bid), they may be able to successfully bruteforce a solution.

  For a plaintext m (<N), with a non-negligible probability, only $\sqrt{m}$ trials are needed to pinpoint m if $\sqrt{m}$ size of memory is available, exploiting,

  $$(m_1 \cdot m_2)^e \equiv m_1^e \cdot m_2^e (\texttt{mod}\ \ N)$$

- Let $c = m^e(\texttt{mod N})$ such that Malice knows $m < 2$. With non-negligible probability m is a composite number satisfying,

  $$m = m_1 \cdot m_2 \texttt{ with } m_1, m_2 < 2^{\frac{l}{2}}$$

  and with RSA's multiplicative property, we have,

  $$c = m_1^e \cdot m_2^e (\texttt{mod}\ \ N)$$

  Malice can build a sorted database

  $$\{1^e, 2^e, \ldots, (2^{\frac{l}{2}})^e\}(\texttt{mod}\ \ N)$$

  Then they can search through the sorted database trying to find $c/i^e(\texttt{mod}\ \ N)$ for $i = 1, 2, \ldots, 2^{\frac{l}{2}}$ a finding signaled by,

  $$c/i^e \equiv j^e(\texttt{mod}\ \ N)$$

  Acheiving a square-root level reduction in time complexity.

- Let Malice be in conditional control of Alice's RSA decryption box. If the ciphertext submitted by Malice is not meaningful, then Alice should return the plaintext to Malice. This is reasonable because of the following:

  1. "A random response for a random challenge" is a standard mode of operation in many cryptographic protocols. (including in the Needham-Schroeder protocol)
  2. This random-looking decryption result should not provide an attacker with any useful information.

  Malice wants to know the plaintext of a ciphertext $c \equiv m^e(\texttt{mod}\ \ N)$ which he has eavesdropped.

  He picks a random number, $r \in Z_N^*$, and computes $c' = r^e c(\texttt{mod}\ \ N)$ and sends his chosen cipher text, c', to Alice. Alice will return,

  $$c'^d \equiv rm(\texttt{mod}\ \ N)$$

  Which will appear completely random to Alice.
  Then because Malice has r, he can obtain m with a division modulo N.

Public Key Infrastructure (PKI)

Encryted key exchange (EKE)

- The EKE protocol protects the password against online eavesdropping and offline dictionary attacks.
- Premise: User, U and Host, H share a password P_u; The system has agreed on a symmetric encryption algorithm, K() denotes symmetric encryption keyed by K; U and H also agreed on an asymmetric encryption scheme, E_u denotes asymmetric encryption under U's key.
- Goal: U and H achieve mutual entity authentication, they also agree on a shared secret key.

1. U generates a random "public" key E_u, and sends to H: U, $P_u(E_u)$
2. H decrypts the cipher chunk using P_u and retrieces E_u; H generates random symmetric key, K, and sends to U: $P_u(E_u(K))$
3. U decrypts the doubly encrypted cipher chunk and obtains K; U generates a nonce N_u, and sends to H: $K(N_e)$
4. H decrypts the cipher chunk using K, generate a nonce, N_H, and sends to U: $K(N_u, N_h)$
5. U decrypts the cipher chunk using K, and return to H: $K(N_h)$
6. If the challenge-response in 3,4,5 is successful, logging-in is granted and the parties proceed further secure communication

Integrity

A MDC is another term for a Message Authentication Code (MAC). A Mac can be created an verified using a keyed hash function technique, or using a block cipher encryption algorithm.

Cryptographic Hash Functions

A Hash function is a deterministic function which maps a bit string of an arbitrary length to a hashed value which is a vit string of a fixed length.

Properties of a Hash Function

- Mixing-transformation
  On any input x, the output hashed value h(x) should be computationally indistinguishable from a uniform binary string.
- Collision Resistance
  It should be computationally infeasible to find two inputs x,y with $x \neq y$ such that $h(x) = h(y)$
- Pre-image resistance
  Given a hashed value h, it should be computationally infeasible to find an input string such that $h = h(x)$
- Practical efficiency
  Given input string x, the computation of $h(x)$ can be done in time bounded by a small-degree polynomial (ideally linear)

Hash Function Applications

- In digital signatures, hash functions are generally used for generating "message digests" or "message fingerprints."
- In public-key cryptosystems, has functions are widely used for realizing a cipher text correctness verification mechanism. This is necessary to protect an encryption scheme from active attackers.
- Hash Functions are used in wide range of applications, pseudo-randomness is required.

MAC based Keyed Hash Function

In a shared-key scenario a hash function takes a key as part of its input.

$$\texttt{MAC} = h(k \| M)$$

Where k is a secret key shared between the transmitter and receiver.
$(\| = \texttt{concatenation})$
The receiver should then recalculate the MAC from the message. If they match, then the message is believed to have come from the transmitter. In order for the receiver to verify itself to the sender, we must compute the HMAC,

$$\texttt{HMAC} = h(k \| M \| k)$$

MAC based on Block Cipher Encryption

Let $E_k(m)$ denote a block cipher encryption algorithm keyed with the key, k, on the input message, m. In order to authenticate a message, M, we need to divide M.

$$M = m_1 m_2 \ldots m_l$$

Where each sub-message block $m_i, i = 1, 2, \ldots, l$ has the size of the input of the block cipher algorithm. Let $C_0 = IV$ be a random initializing vector. Here the transmitter applies the CBC encryption:

$$C_i \leftarrow E_k(m_i \oplus C_{i-1}), i = 1, 2, \ldots, l$$

Then pair,

$$(IV, C_l)$$

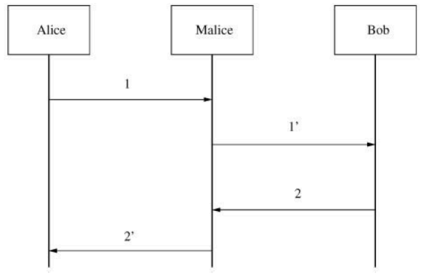
Key Management

Signing

Password Hashing

Mistakes

Man in the Middle Attack on Diffie-Helman



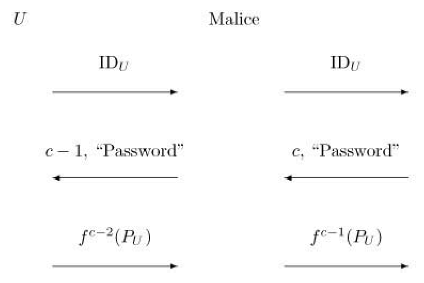
1. Alice picks a $\in_u [1, p-1)$, computes $g_a \leftarrow g^a(\texttt{mod}\ \ p)$ she sends g_a to Malice("bob");
2. (1') Malice("Alice") computes $g_m \leftarrow g^m(\texttt{mod}\ \ )$ for some $m \in [1, p-1)$; he sends g_m to Bob;
3. (2) Bob picks $b \in_U [1, p-1)$, computes $g_b \leftarrow g^b(\texttt{mod}\ \ p)$; he sends g_b Malice("Alice");
4. (2') Malice("Bob") sends to Alice: g_m;
5. (3) Alice computes $k_1 \leftarrow g_m^a(\texttt{mod}\ \ p)$;
6. (4) Bob computes $k_2 \leftarrow g_m^b(\texttt{mod}\ \ p)$;

Man-in-the-Middle Attack

Malice intercepts messages between Bob and Alice

Parallel Session Attack

The parallel session attack consists of two or more runs of a protocol executed concurrently under Malice's protection
An Attack on the S/KEY Protocol:



Result: Malice has $f^{c-2}(P_U)$ which he can use for logging-ing in the name of U

Reflection Attack

A reflection attack is when an honest principal sends a message to an intended communication partner. Malice intercepts the message and reflects it back at the host

Interleaving Attack

In an interleaving attack, two or more runs of a protocol are executed in an overlapping fashion under Malic's orchestration. Malic may compose a message and send it out to a principal in one run from which he expects to receive and answer.

Attack Due to Type Flaw

Malice uses a flaw, including a principal being tricked to misinterpret a nonce, a timestamp or an indentification into key

Attack Due to Name Omission

Name omission is a serious problem that could allow exploits

Network Security

Structure

TLS

Building end-to-end channels: communication protections achieved all the way from originating client to intended server and no need to trust intermediaries. Deals with Eavesdropping (encryption, session keys) manipulation(integrity, use MAC, and replay protection), and Impersonation (Signatures) SSL - Secure Sockets Layer . TLS: Transport Layer Security. Used interchangeably. Security for any application that uses TCP : guarantees encryption/confidentiality + integrity + authentication(of server but not of client). Puts s in https.
Diffie-Hellman has perfect forward secrecy. If private key found out, old session keys aren't exchanged, so old messages can't be read. Certificates: Signed statement about someone's private key. Simply states that given public key Kbob belongs to Bob Validating Identity: Browser compares domain name in cert w/URL as opposed to IP address. Browder accesses separate cert belonging to issuer. Hardwired into browser and trusted (could be chain). Browser applies issuer's public key to verify signature. Compares with it's own SHA-1 hash of cert. Assuming cases match, high confidence. Powerful Protections:

- Attacker runs sniffer to capture Wifi session? But encrypted communication unreadable
- DNS Cache Poisoning or DHCP spoofing? Client goes to wrong server but detects impersonation
- Attacker hijacks connection? Injects new traffic but data receiver rejects due to failed integrity check
- Attacker manipulates routing to run by eavesdropper or wrong server? But detect impersonation and can't read
- Slips MITM? Can't read, can't inject, can't even replay previous traffic

Limitations:

- Hassle of buying certs, integrating with sites that don't use https, latency: 1st page slower to load,
- TCP-level denial of service: SYN flooding, RST injection.
- SQL injection/XSS/server-side coding/logic flaws, User flaws: weak passwords, phishing

TCP Summary Attacker who can observe TCP connection can:

- Terminate by forging RST packet
- Inject (spoof) data into either direction
- No security against on-path attackers. Reasonable TCPsecurity against off-path attackers, but UDP and IP are not.

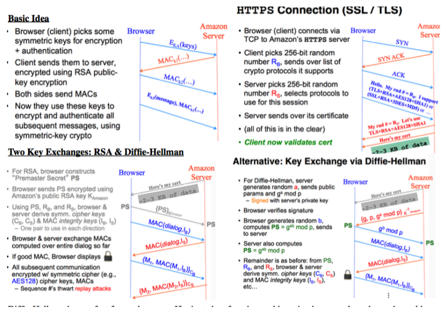
Source port and Destination and IP addr: Define conn Sequence Number - Defines where this packet fits within the sender's byte stream. Data Injection: If attacker knows ports and seq numbers (on-path attackers), attacker can inject data into any TCP connection. On-path attacker can shut down TCP connection if they see traffic - infer port and seq numbers and insert fake data (China). Off-path attacker: If they can infer/ guess pot and seq number. Blind Spoofing: On and Off path attackers can create fake TCP connection if they can infer/guess TCP initial seq numberss.

SSL/TLS

- Process: *=optional

1. $C \to S$: ClientHello
2. $S \to C$: Server Hello, ServerCertificate*, ServerKeyExchange*, CertificateRequest*, ServerHelloDone
3. $C \to S$: ClientCertificate*, ClientKeyExchange, CertificateVerify*, ClientFinished
4. $S \to C$: ServerFinished

- Hello Message Exchange: Server and Host let each other know what protocols they are capable of running
- handshake with key exchange
- crypto-suite selection, certificates
- use of nonces and random secrets



Basic Idea

- Browser (client) picks some symmetric keys for encryption + authentication
- Client sends them to server, encrypted using RSA public-key encryption
- Both sides send MACs
- Now they use these keys to encrypt and authenticate all subsequent messages, using symmetric-key crypto

HTTPS Connection (SSL / TLS)

- Browser (client) connects via TCP to Amazon's `HTTPS` server
- Client picks 256-bit random number R_B, sends over list of crypto protocols it supports
- Server picks 256-bit random number R_S, selects protocols to use for this session
- Server sends over its certificate
- (all of this is in the clear)
- *Client now validates cert*

Two Key Exchanges: RSA & Diffie-Hellman

- For RSA, browser constructs "Premaster Secret" PS
- Browser sends PS encrypted using Amazon's public RSA key K_{Amazon}
- Using PS, R_B, and R_S, browser & server derive symm. *cipher keys* (C_B, C_S) & MAC *integrity keys* (I_B, I_S)
  - One pair to use in each direction
- Browser & server exchange MACs computed over entire dialog so far
- If good MAC, Browser displays 🔒
- All subsequent communication encrypted w/ symmetric cipher (e.g., AES128) cipher keys, MACs
  - Sequence #'s thwart replay attacks

Alternative: Key Exchange via Diffie-Hellman

- For Diffie-Hellman, server generates random a, sends public params and $g^a \bmod p$
  - Signed with server's private key
- Browser verifies signature
- Browser generates random b, computes PS = $g^{ab} \bmod p$, sends to server
- Server also computes PS = $g^{ab} \bmod p$
- Remainder is as before: from PS, R_B, and R_S, browser & server derive symm. *cipher keys* (C_B, C_S) and MAC *integrity keys* (I_B, I_S), etc...

Browser
Amazon Server
E_{K_A}(keys)
$MAC_{k1}(\ldots)$
$MAC_{k2}(\ldots)$
E_{k3}(message), $MAC_{k1}(\ldots)$
SYN
SYN ACK
ACK
Hello. My rnd # = R_B. I support (TLS+RSA+AES128+SHA1) or (SSL+RSA+3DES+MD5) or …
My rnd # = R_S. Let's use TLS+RSA+AES128+SHA1
Here's my cert
~2-3 KB of data
PS
$\{PS\}_{K_{Amazon}}$
$MAC(dialog, I_B)$
$MAC(dialog, I_S)$
$\{M_1, MAC(M_1, I_B)\}_{C_B}$
$\{M_2, MAC(M_2, I_S)\}_{C_S}$
$\{g, p, g^a \bmod p\}_{K^{-1}_{Amazon}}$
$g^b \bmod p$

Network Attacks

DNS Attacks

DNS translates a website from `www.google.com` to `74.125.25.99`. It basically is a giant dictionary that your local Internet provided holds and checks. Your computer checks with the local network and then a TLD DNS server to find the correct IP. Currently ICANN runs the DNS servers for each TLD. Most of it is cached. DNS is a critical part of the Internet (SOP relies on this). Thus there are several attacks that can take place. DOS (seen next section) could work but the systems are overengineered to prevent this.

- Malicious DNS sever can lie to you can send you to a malicious website. **Cache poisoning** is when a server sends back IPs for web servers that are not in the network, poisoning the cache. This has been patched.
- An on path attacker can see the query and the 16 bit transaction identifier to easily pull a man in the middle attack on you. Not much you can do here, and it is what the NSA uses for QuantumDNS.
- An off path attack entailed forcing the attacker to visit a site, and since we know what they are looking up we simply need to guess the id number and reply faster than the actual sever. This attacker was easier when id fields were incremental. Now it is randomized and only possible if the attacker can send thousands of replies a second. But you could have also done Kaminsky (blind spoofing) in which the attacker doesn't care what you lookup, it just used the Additional field in the DNS header to spoof the domain. The defense of this was the client use a source port randomizer.

Denial of Service (DOS)

Prevent legitimate users from using a computing service by disrupting the service to them. You can do this by exploiting a **program flaw** or though **resource exhaustion** - overloading the server with requests. These attacks happen at an network level. **DDos** distributes the attack over multiple servers so that it is hard to trace and stop the attack. Doing this only requires you to send the capacity of the bottleneck link of the target's Internet connection. **Amplification attacks** are when an attacker leverages the system's own structure to pump up the load the induce on a resource. An example of this is exploiting a mail delivery server by sending large files to many recipients that don't exist which will force the mail server to send all the files back individually, overloading it (this has since been patched).

Firewalls

Firewalls: Security Policy: Inbound: External users connect to services running on internal machines. Outbound: Internal users contact with external services. Default-allow - every network service permitted, unless denied. Default-deny: Every network service denied, unless specifically listed as allowed. Need Central Chokepoint: Single place to monitor, enforce policy. Stateful packet filter: Check each packet against access control policy. If allows, fwd on, otherwise dropped. Keeps track of all open connections. Impt: Allows firewall to check whether packet part of already-open connection.

```
allow tcp *:* -> 1.2.3.4:25
drop * *:* -> *:*
```

Allows anyone to open a TCP connection to port 25 on machine 1.2.3.4. (It blocks all other connections. Notation: 1.2.3.4:25 indicates IP address 1.2.3.4 and port 25; * is a wildcard, which may appear in any field.

```
allow tcp *:*/ext -> 1.2.3.4:25/int
allow tcp *:*/int -> *:*/ext
drop * *:* -> *:*
```

Allows inbound connections to port 25 on 1.2.3.4 (rule 1) and allows all outbound connections (rule 2); all other connections are dropped (rule 3).
Sender might sneak: root might span packet boundaries. Packets might be reordered. Must have sequence number. Else, discard. Stateless Packet Filters Cruder, operate at network level. Look at TCP, UDP , IP Headers. Rules specify whether to drop. Require hacks to handle TCP Application-layer firewalls: Restrict traffic according to contents of data fields. Secure External Access: VPN to tunnel traffic form outside network to inside network. Firewalls successful because central control, easy to deploy, address security vulnerabilities in network services. Disadvantages: Functionality (some apps don't work), malicious insider problem, establish security perimeter

Common Names

The Common Name (also CN) identifies the fully qualified domain name(s) associated with the certificate. It is typically composed by an host and a domain name it looks like (e.g. www.example.com or example.com).

Detection

NIDS table of all active connections and maintains state for each. RST packet: Reset takes effect immediately, no ack needed, only accepted if has correct sequence number. if NIDS sees RST packet, assume RST won't be received.
Approach 1: Look at network traffic. No need to touch/trust end systems.
Approach 2: Instrument web server: Works for encrypted and no problems with http complexities like % escapes. But have to add code to each web server, consider unix semantics, sensitive files. **Approach 3:** Log files: Analyze at night. Cheap, but detected delayed. **Approach 4:** Monitor system call activity backend processes. No issues with complexities, but maybe False positives. False Positives: Alerting about problem when no problem. False Negatives: Failing to alert, when there was problem. Signature-Based Detection: Look for activity that matches structure of a known attack. Build up library, conceptually simple, but blind to novel attacks. Vulnerability Signatures: Don't match on known attacks, match on known problems. Can detect variants of known attacks, but can't detect new vulnerabilities. Anomaly-Based :Something looks peculiar. Develop model of normal behavior. Potential detection of wide range of attacks. Can fail to detect known and novel attacks. High FP rate. Specification-based:Specify what's allowed. Nice: detect novel attacks, can have low FP, but expensive. Behavioral-Based:Look for evidence of compromise, Ex: code injection. Can detect wide range of novel attacks, low false positives, cheap. But no opportunity to prevent it, post facto detection.

DNSSEC

How can we trust that when clients look up names with DNS, they trust the answer they receive? **DNSSEC** - Standardized DNS security extensions currently being deployed. Idea 1 - Why can't just secure DNS using SSL/TLS: Slow. Doesn't protect against malicious DNS servers. Caching: crucial for DNS scaling, but then authentication assurances? Security: must trust the resolver. Idea 2: make DNS results like certs. Public key for each domain. As a resolver works from DNS root down, each level gets signed statement regarding keys used by next level.

DNSSEC vs TLS

TLS provides channel security for communication over TCP. It ensures confidentiality, integrity, authentication, but the client and server must agree on crypto and session keys. It is also dependent on trusting a Certificate Authority and their decisions as well as the implementors design flaws
DNSSEC provides object security for DNS results. It ensures integrity and authentication but not confidentiality but there is no client/serve setup dialog and it must be tailored to be caching friendly. It is also dependent in trusting the Root Name Server's key and all the signing keys.

Miscellaneous

Bitcoin

Hash-Chaining: Bitcoin relies on hash chaining to prevent malicious actors from modifying events which occurred in the past. This works by starting from some root x0, and sending xi+1 = H(xi, message). BitCoin: Problems with simple scheme: initial balance arbitrary, broadcasting is expensive, transactions aren't authenticated, double-spending, synchronization, sybil attacks (someone take over network), graph cut (malicious neighbors). Solutions: Use public keys to identify users, linearize history in public log, store transaction log in hash chain(honest nodes will reject any broadcasts that do anything other than append), longest chain wins, Reward miners: each item appended must be a proof of work : hash starts with 33 0 bits first to successfully append wins money,

Electronic Voting

In an age where most computation and data is stored in the cloud, we can retain privacy through policy to minimize data collection and access and secure the data using more powerful technology.

Policy mitigations too ensure and Technology Merkle Tree

Tor

Anonymity is hard, and almost impossible to achieve just on your won. The best technique for anonymity is to **ask someone else to send the packets for you**.

A **Proxy** is a trusted proxy that acts as an intermediary between you and normal Internet traffic, it hides your IP and a trace of the packets to you. However this is slow and you must trust the proxy server as they can trace your traffic to your IP. The government can also subpoena the proxy to reveal this. Thus this acts as a single point of vulnerability.

Tor is built using an approach called **Onion Routing** which basically obfuscates the traffic by encrypting with the public key of and passing it through multiple intermediaries. The Tor protocol ensures that the message is passed along to the right recipient, but no one knows both you and the destination. The approach also allows for *bidirectional communication*.