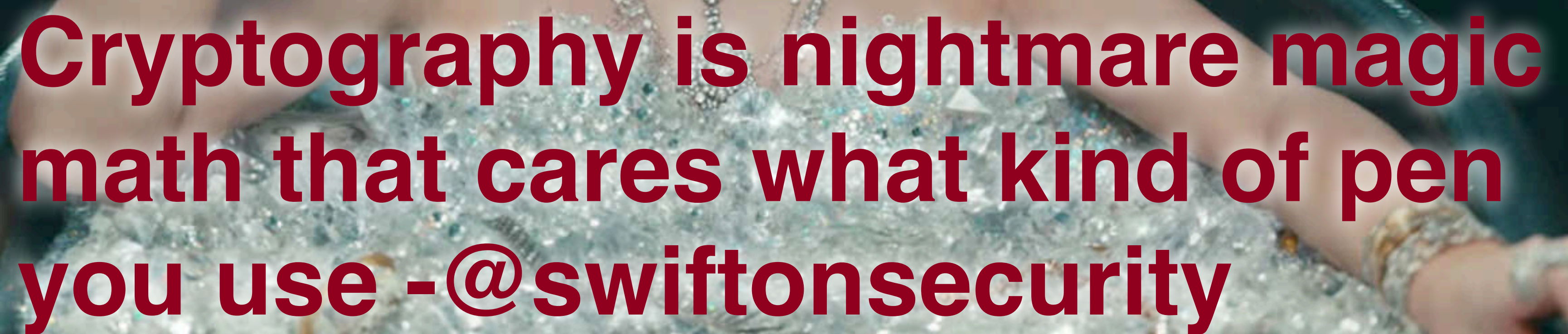


Cryptography is nightmare magic math that cares what kind of pen you use -@swiftonsecurity

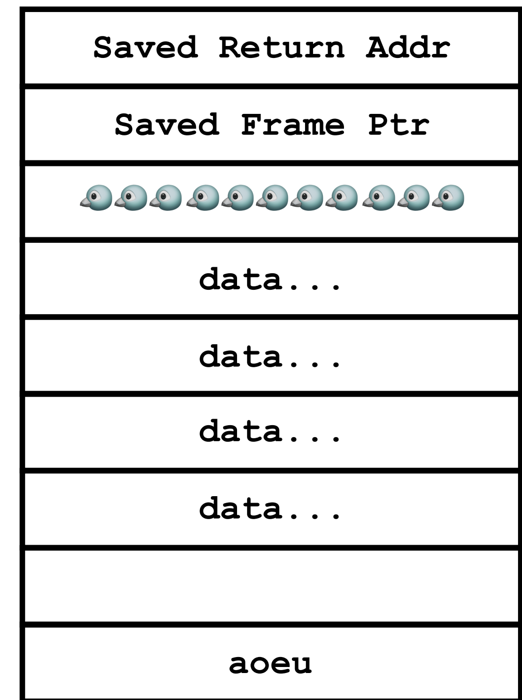
More Defenses & Start of Crypto

But Suppose You Don't Want To Reprogram Things? What Then?

- A large back-and-forth arms race trying to prevent memory errors from being ***exploitable for code injection***
  - An attacker can still use them to crash the program
  - An attempt at defense-in-depth
- Stack Canaries
- Non-Executable Pages
- Address-Space-Layout-Randomization
- And some R&D down the pipe
  - E.g. selfrando

Stack Canaries…

- Goal is to protect the return pointer from being overwritten by a stack buffer…
- When the program starts up, create a ***random*** value
  - The “stack canary”
- When returning in a function
  - First check the canary against the stored value



How To (Not) Kill the Canary…

- Find out what the canary is!
  - A format string vulnerability
  - An information leak elsewhere that dumps it
  - Now can overwrite the canary with itself…
- Write around the canary
  - Format string vulnerabilities
- Overflow in the heap, or a C++ object on the stack
- QED: Bypassable but raises the bar
  - A simple stack overflow doesn't work anymore:
    Need something a bit more robust
  - Minor but nearly negligible performance impact
    - First deployed in 1997 with "StackGuard"
- It requires a compiler flag to enable on Linux, but…
  - ***THERE IS NO EXCUSE NOT TO HAVE THIS ENABLED!!! I'M LOOKING AT YOU CISCO ASA!***

And Canary Entropy…

- On 32b x86 the canary is a 32b value
  - It is 64b on x86-64
- One byte of the canary is always x0
  - Since some buffer overflows can't include null bytes:
    e.g. if the vulnerability is in a bad call to `strcpy`
- But this means you can (possibly) brute-force the canary
  - It would only requires an expected 2^{24} tries or so!
    - Think of this as "you need to try ~16 million times":
      $2^{10} \sim= 10^3$

Non-Executable Pages

- We remember how the TLB/page table has multiple bits:
  - R -> Can Read
    W -> Can Write
    X -> Can Execute
- So lets maintain W xor X as a global property
  - Now you can't write code to the stack or heap
- Unfortunately that is insufficient
  - "Return into libc": Just set up the stack and "return" to exec
    - Especially easy on x86 since arguments are passed on the stack
  - "Return Oriented Programming"

W^X is Somewhat Ubiquitous As Well: Playing games with the page table...

- The OS enforces a simple rule:
  By default, a memory page may be writeable or executable but ***not both!***
- Effectively ***no*** performance impact
  - Synergistic interaction with ASLR
- Does break some code…
  - Stuff which dynamically generates code on the fly ***and doesn't know about W^X***. So basically stuff that deserves to break
  - FreeBSD deployed in 2003, Windows in 2004
  - But don't always have apps supporting it!
- Yet still often not ubiquitous on embedded systems
  - See "Internet of Shit", Cisco ASA security appliances…

Return Oriented Programming...

- The deep-voodoo idea:
  - Given a code library, find a set of fragments (gadgets) that when called together execute the desired function
    - The "ROP Chain"
  - Inject into memory a sequence of saved "return addresses" that will invoke this
- The lazy-hacker idea:
  - Somebody else did the deep voodoo already. I can just google for "ROP compiler" and download an existing tool
- Tools democratize things for attacker's:
  - Yesterday's Ph.D. thesis or academic paper is today's Intelligence Agency tool and tomorrow's Script Kiddie download

Address Space Layout Randomization

- Start things more randomly
  - Especially on 64b operating systems with 64b memory space:
    64b operating systems tend to be significantly harder to exploit
- Randomly relocate everything:
  - Every library, the start of the stack & heap, etc…
  - With 64b of space you have lots of entropy
    - Everything needs to be ***relocatable*** anyway:
      Modern systems use relocatable code and link at runtime
  - 32b? Not-so-much
- When combined with W^X, need an information leak
  - Often a separate vulnerability, such as a way to find the address of a function
  - To find the magic offset needed to modify your ROP chain

ASLR Efficiency...

- The modern OS already has to relocate everything
  - Dynamically load all the desired libraries
- So additional overhead is effectively 0!
  - Just instead of putting things in a sequential order, you just randomize how things are...
- But you need to be page aligned
  - So well less than the full entropy of the memory space:
    This is why it is much more effective on a 64b architecture...
    32b architecture you may have low enough entropy that an attacker can brute force things

Defense In Depth in Practice: Attacker Requirements...

- Attacker first needs to discover a way to ***read*** memory
  - Just a single pointer to a known library will do, however
    - The return address off the stack is often a great candidate
    - Or a `vtable` pointer for an object of a known type
- Armed with this, the attacker now can create a ROP chain
  - Since the attacker has a copy of the library of their own and has already passed it through a ROP compiler, it just needs to know the starting point for the library
- Now the attacker needs to ***write*** memory
  - Writes the ROP chain and overwrites a control flow pointer

These Defenses-In-Depth in Practice...

- Apple iOS uses ASLR in the kernel and userspace, W^X whenever possible
  - All applications are sandboxed to limit their damage: The kernel is the TCB
- The "Trident" exploit was used by a spyware vendor, the NSO group, to exploit iPhones of targets
- So to remotely exploit an iPhone, the NSO group's exploit had to...
  - Exploit Safari with a memory corruption vulnerability
    - Gains remote code execution within the sandbox: write to a R/W/X page as part of the JavaScript JIT
  - Exploit a vulnerability to read a section of the kernel stack
    - Saved return address & knowing which function called breaks the ASLR
  - Exploits a vulnerability in the kernel to enable code execution
- Full details: https://info.lookout.com/rs/051-ESQ-475/images/pegasus-exploits-technical-details.pdf

Safari Exploit: More Details

- Basic idea: can corrupt a JavaScript object (due to interaction with garbage collector) to trigger a use-after-free issue
  - Attacker JavaScript has access to both objects that share the same memory:
    - Newly allocated object is an array of integers
    - Old object changes the length of the array to be 0xFFFFFFFF
- Now attacker has a "read/write" primitive
  - The array can see a huge fraction of the memory space
    - First thing, find out the offset of the array itself, then any other magic numbers needed
- Turning it into execution
  - Take another JavaScript object that will get compiled (the "Just In Time" compiler)...
  - That object's code pointer will point into space that is writeable and executable

Coming Down The Pipe: Selfrando...

- Don't just randomize the location of all libraries...
- Randomize the location of ***every*** function within the library!
  - Slows down program loading considerably, unlike ASLR
- It works, but...
  - To construct a ROP chain you may need more addresses, but...
  - If you have an arbitrary read primitive, you can get that, it is just more tedious

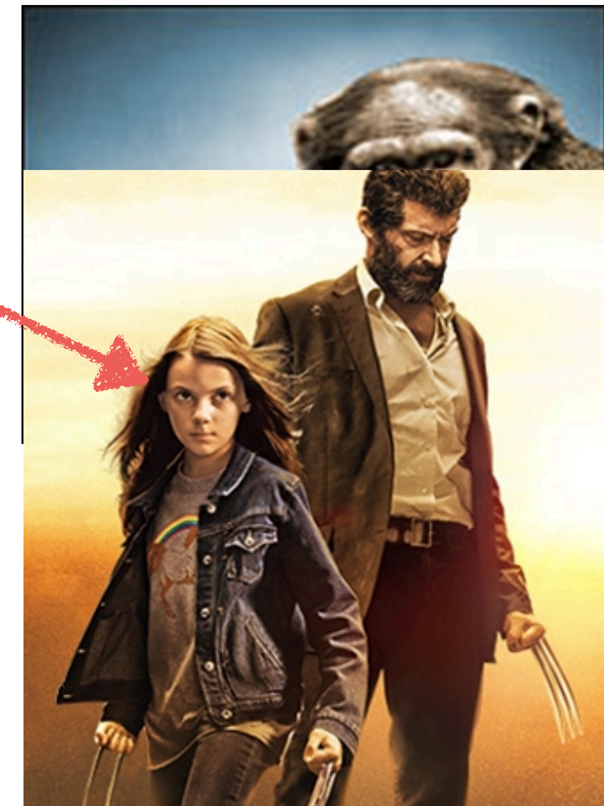
Why does software have vulnerabilities?

- Programmers are humans.
  And humans make mistakes.
  - Use tools
- Programmers often aren't security-aware.
  - Learn about common types of security flaws.
- Programming languages aren't designed well for security.
  - Use better languages (Java, Python, …).

Testing for Software Security Issues

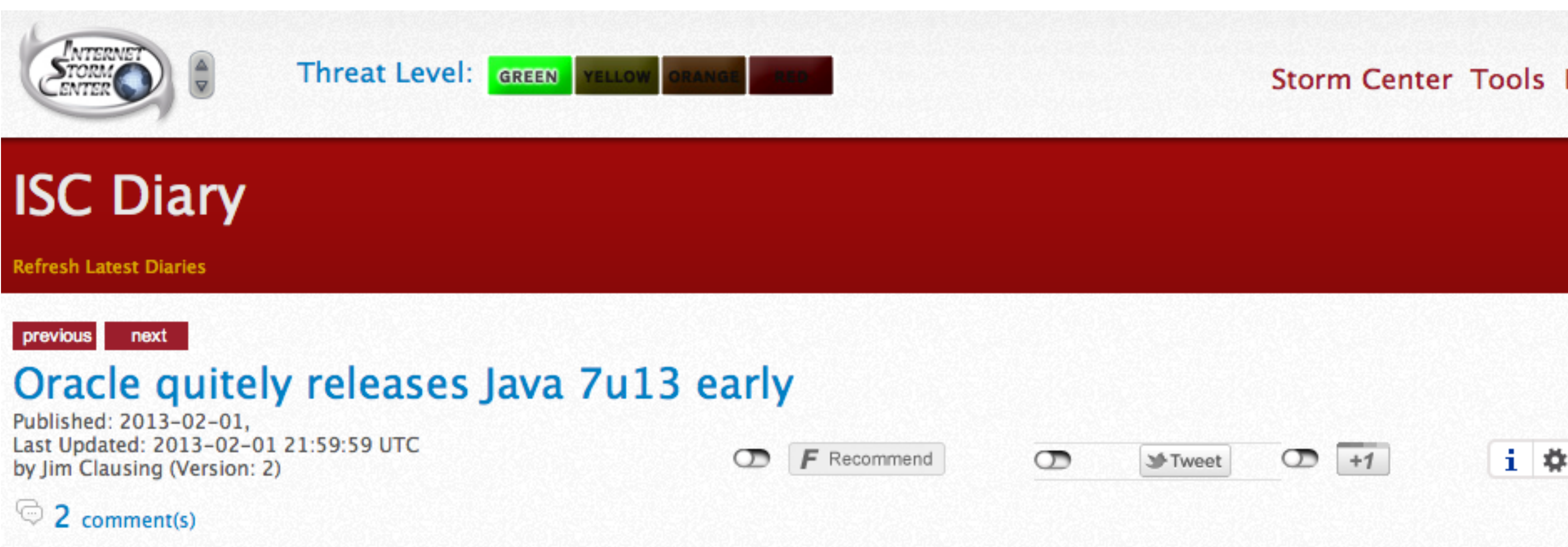
- What makes testing a program for security problems difficult?
  - We need to test for the absence of something
    - Security is a ***negative*** property!
      - “nothing bad happens, even in really unusual circumstances”
  - Normal inputs rarely stress security-vulnerable code
- How can we test more thoroughly?
  - Random inputs (fuzz testing)
  - Mutation
  - Spec-driven
  - Test ***corner cases***
- How do we tell when we’ve found a problem?
  - Crash or other deviant behavior
- How do we tell that we’ve tested enough?
  - Hard: but code-coverage tools can help

Disney's Newest Princess!



Working Towards Secure Systems

- Along with securing individual components, we need to keep them up to date …
- What's hard about patching?
  - Can require restarting production systems
  - Can break crucial functionality



First off, a huge thank you to readers Ken and Paul for pointing out that Oracle has released Java 7u13. As the CPU (Critical Patch Update) bulletin points out, the release was originally scheduled for 19 Feb, but was moved up due to the active exploitation of one of the critical vulnerabilities in the wild. Their Risk Matrix lists 50 CVEs, 49 of which can be remotely exploitable without authentication. As Rob discussed in his diary 2 weeks ago, now is a great opportunity to determine if you really need Java installed (if not, remove it) and, if you do, take additional steps to protect the systems that do still require it. I haven't seen jusched pull this one down on my personal laptop yet, but if you have Java installed you might want to do this one manually right away. On a side note, we've had reports of folks who installed Java 7u11 and had it silently (and unexpectedly) remove Java 6 from the system thus breaking some legacy applications, so that is something else you might want to be on the lookout for if you do apply this update.

Working Towards Secure Systems

- Along with securing individual components, we need to keep them up to date …
- What's hard about patching?
  - Can require restarting production systems
  - Can break crucial functionality
  - Management burden:
    - It never stops (the "patch treadmill") …

News

info security
STRATEGY /// INSIGHT /// TECHNIQUE

IT administrators give thanks for light Patch Tuesday

07 November 2011

Microsoft is giving IT administrators a break for Thanksgiving, with only four security bulletins for this month's Patch Tuesday.

Only one of the bulletins is rated critical by Microsoft, which addresses a flaw that could result in remote code execution attacks for the newer operating systems – Windows Vista, Windows 7, and Windows 2008 Server R2.

The critical bulletin has an exploitability rating of 3, suggesting

Working Towards Secure Systems

- Along with securing individual components, we need to keep them up to date …
- What's hard about patching?
  - Can require restarting production systems
  - Can break crucial functionality
  - Management burden:
    - It never stops (the "patch treadmill") …
    - … and can be difficult to track just what's needed where
- Other (complementary) approaches?
  - Vulnerability scanning: probe your systems/networks for known flaws
  - Penetration testing ("pen-testing"): pay someone to break into your systems …
    - … provided they take excellent notes about how they did it!

ars technica

RISK ASSESSMENT / SECURITY & HACKTIVISM

Extremely critical Ruby on Rails bug threatens more than 200,000 sites

Servers that run the framework are by default vulnerable to remote code attacks.

by **Dan Goodin** - Jan 8 2013, 4:35pm PST

HARDENING 38

Hundreds of thousands of websites are potentially at risk following the discovery of an extremely critical vulnerability in the Ruby on Rails framework that gives remote attackers the ability to execute malicious code on the underlying servers.

The bug is present in Rails versions spanning the past six years and in default configurations gives hackers a simple and reliable way to pilfer database contents, run system commands, and cause websites to crash, according to Ben Murphy, one of the developers who has confirmed the vulnerability. As of last week, the framework was used by more than 240,000 websites, including Github, Hulu, and Basecamp, underscoring the seriousness of the threat.

"It is quite bad," Murphy told Ars. "An attack can send a request to any Ruby on Rails sever and then execute arbitrary commands. Even though it's complex, it's reliable, so it will work 100 percent of the time."

Murphy said the bug leaves open the possibility of attacks that cause one site running rails to seek out and infect others, creating a worm that infects large swaths of the Internet. Developers with the Metasploit framework for hackers and penetration testers are in the process of creating a module that can scan the Internet for vulnerable sites and exploit the bug, said HD Moore, the CSO of Rapid7 and chief architect of Metasploit.

Maintainers of the Rails framework are urging users to update their systems as soon as possible to

Some Approaches for Building Secure Software/Systems

- Run-time checks
  - Automatic bounds-checking (overhead)
  - What do you do if check fails? Probably controlled crash...
- Address randomization
  - Make it hard for attacker to determine layout
  - But they might get lucky / sneaky
- Non-executable stack, heap
  - May break legacy code
  - See also Return-Oriented Programming (ROP)
- Monitor code for run-time misbehavior
  - E.g., illegal calling sequences
  - But again: what do you if detected?

Approaches for Secure Software, con’t

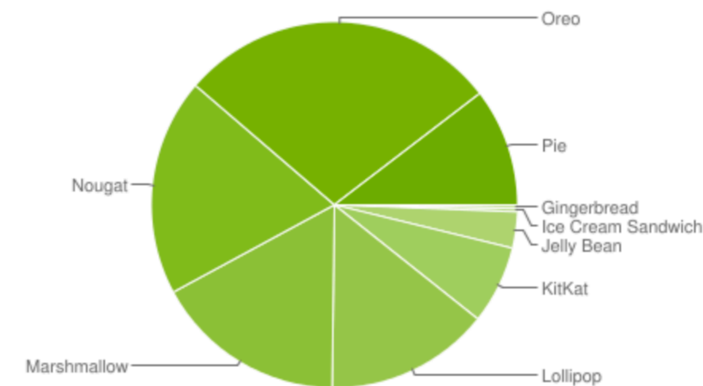
- Program in checks / “defensive programming”
  - E.g., check for null pointer even though sure pointer will be valid
  - Relies on programmer discipline
- Use safe libraries
  - E.g. **`strlcpy`**, not **`strcpy`**; **`snprintf`**, not **`sprintf`**
  - Relies on discipline or tools …
- Bug-finding tools
  - Excellent resource as long as not many false positives
- Code review
  - Can be very effective … but expensive

Approaches for Secure Software, con't

- Use a safe language
  - E.g., Java, Python, C#, Go, Rust
  - Safe = memory safety, strong typing, hardened libraries
  - Installed base? Programmer base? ~~Performance~~?
- Structure user input
  - Constrain how untrusted sources can interact with the system
    - Really key later when we get to SQL injection...
  - Perhaps by implementing a reference monitor
- Contain potential damage
  - E.g., run system components in jails or VMs
  - Think about privilege separation

Real World Security: Securing your cellphone... Look on the back:

- Does it say "iPhone"?
  - Keep it up to date and be happy
- Does it say "Nexus" or "Pixel"?
  - Keep it up to date and be happy
- Does it say anything else?
  - Toss it in the trash and buy an iPhone or a Pixe



Data collected during a 7-day period ending on May 7, 2019.
Any versions with less than 0.1% distribution are not shown.

- Why? The Android Patch Model...
  - "Imagine if your Windows update needed to be approved by Intel, Dell, and Comcast... And none of them cared or had a reason to care"

Cryptography: Philosophy...

- This part of the class is really ***don't try this at home***
  - It is ***incredibly easy*** to screw this stuff up
- It isn't just a matter of making encryption algorithms...
  - Unless your name is David Wagner or Ralcua Popa, ***your crypto is broken***!
- It isn't just a matter of coding good algorithms...
  - Although just writing 100% correct code normally is hard enough!
- There is all sorts of deep voodoo that, ***when*** you screw up your security breaks
  - EG, bad random number generators, side channel attacks, reusing one-use-only items, replay attacks, downgrade attacks, you name it...

Three main goals

- ***Confidentiality***: preventing adversaries from ***reading*** our private data
  - Data = message or document
- ***Integrity***: preventing attackers from ***altering*** our data
  - Data itself might or might not be private
- ***Authentication***: proving who ***created*** a given message or document
  - Generally implies/requires integrity

Special guests

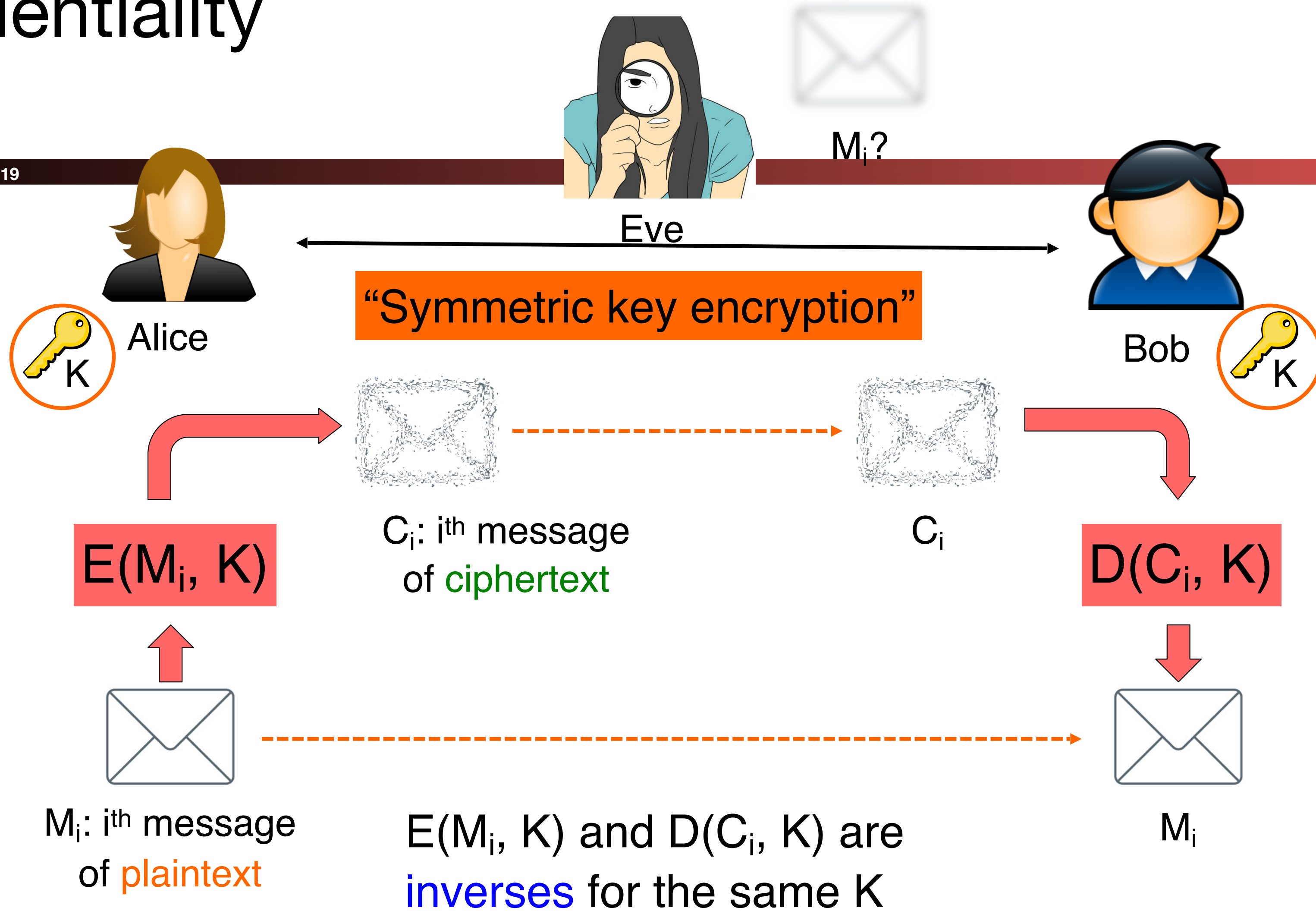
- Alice (sender of messages)

- Bob (receiver of messages)

- The attackers
  - Eve: “eavesdropper”
  - Mallory: “manipulator”

Eve

Confidentiality



The Ideal Contest

- Attacker's goal: any knowledge of M_i beyond an upper bound on its length
  - Slightly better than 50% probability at guessing a single bit: attacker wins!
  - Any notion of how M_i relates to M_j: attacker wins!

- Defender's goal: ensure attacker has no reason to think any $M' \in \{0,1\}^n$ is more likely than any other
  - (for M_i of length n)

Eve's Capabilities/Foreknowledge

- No knowledge of **K**
  - We assume **K** is selected by a ***truly random process***
  - For **b**-bit key, any **K** $\in$ **{0,1}^{b}** is equally likely
- Recognition of success: Eve can generally tell if she has correctly and fully recovered $\mathbf{M_i}$
  - But: Eve cannot recognize anything about ***partial solutions***, such as whether she has correctly identified a particular bit in $\mathbf{M_i}$
    - There are some attacks where Eve can guess and verify
  - Does not apply to scenarios where Eve exhaustively examines every possible $\mathbf{M_i'} \in \mathbf{\{0,1\}^n}$

Eve's Available Information

1. Ciphertext-only attack:
   - Eve gets to see every instance of $\mathbf{C_i}$
   - Variant: Eve may also have partial information about $\mathbf{M_i}$
     - "It's probably English text"
     - Bob is Alice's stockbroker, so it's either "Buy!" or "Sell"
2. Known plaintext:
   - Eve knows part of $\mathbf{M_i}$ and/or entire other $\mathbf{M_j}$s
   - How could this happen?
     - Encrypted HTTP request: starts with "GET"
     - Eve sees earlier message she knows Alice will send to Bob
     - Alice transmits in the clear and then resends encrypted
     - Alex the Nazi always transmits the weather report at the same time of day, with the word "Wetter" in a known position

Eve's Available Information, con't

3. Chosen plaintext
   - Eve gets Alice to send $\mathbf{M_j}$'s of Eve's choosing
   - How can this happen?
     - E.g. Eve sends Alice an email spoofed from Alice's boss saying "Please securely forward this to Bob"
     - E.g. Eve has some JavaScript running in Alice's web browser that is contacting Bob's TLS web server
4. Chosen ciphertext:
   - Eve tricks Bob into decrypting some $\mathbf{C_j}$' of her choice and he reveals something about the result
   - How could this happen?
     - E.g. repeatedly send ciphertext to a web server that will send back different-sized messages depending on whether ciphertext decrypts into something well-formatted
     - Or: measure how long it takes Bob to decrypt & validate

Eve's Available Information, con't

5. Combinations of the above
- Ideally, we'd like to defend against this last, the most powerful attacker
- And: we can!, so we'll mainly focus on this attacker when discussing different considerations

Independence Under Chosen Plaintext Attack game: IND-CPA

- Eve is interacting with an encryption "Oracle"
  - Oracle has an unknown random key **k**
- She can provide two separate chosen plaintexts of the same length
  - Oracle will randomly select one to encrypt with the unknown key
  - The game can repeat, with the oracle using the same key...
- Goal of Eve is to have a better than random chance of guessing which plaintext the oracle selected
  - Variations involve the Oracle always selecting either the first or the second

Designing Ciphers

- Clearly, the whole trick is in the design of **E(M,K)** and **D(C,K)**
- One very simple approach:
  E(M,K) = **ROTK(M)**; **D(C,K)** = **ROT-K(C)**
  i.e., take each letter in **M** and "rotate" it **K** positions (with wrap-around) through the alphabet
- E.g., $\mathbf{M_i}$ = "DOG", **K** = 3
  $\mathbf{C_i}$ = $\mathbf{E(M_i,K)}$ = **ROT3**("DOG") = "GRJ"
  $\mathbf{D(C_i,K)}$ = **ROT-3**("GRJ") = "DOG"
- "Caesar cipher"
  - "This message has been encrypted twice by ROT-13 for your protection"

Attacks on Caesar Ciphers?

- Brute force: try every possible value of K
  - Work involved?
  - At most 26 “steps”

Attacks on Caesar Ciphers?

- Brute force: try every possible value of K
  - Work involved?
  - At most 26 “steps”
- Deduction:
  - Analyze letter frequencies (“ETAOIN SHRDLU”)
  - Known plaintext / guess possible words & confirm
    - E.g. “JCKN ECGUCT” =?

Attacks on Caesar Ciphers?

- Brute force: try every possible value of K
  - Work involved?
  - At most 26 “steps”
- Deduction:
  - Analyze letter frequencies (“ETAOIN SHRDLU”)
  - Known plaintext / guess possible words & confirm
    - E.g. “JCKN ECGUCT” =? “HAIL CAESAR”

Attacks on Caesar Ciphers?

- Brute force: try every possible value of K
  - Work involved?
  - At most 26 “steps”
- Deduction:
  - Analyze letter frequencies (“ETAOIN SHRDLU”)
  - Known plaintext / guess possible words & confirm
    - E.g. “JCKN ECGUCT” =? “HAIL CAESAR” ⇒ K=2
  - Chosen plaintext
    - E.g. Have your spy ensure that the general will send “ALL QUIET”, observe “YJJ OSGCR” ⇒ K=24
- Is this IND-CPA?

Kerckhoffs' Principle

- Cryptosystems should remain secure even when attacker knows all internal details
  - Don't rely on security-by-obscurity
- Key should be only thing that must stay secret
- It should be easy to change keys
  - Actually distributing these keys is hard, but we will talk about that particular problem later.
  - But key distribution is one of the real...

Better Versions of Rot-K ?

- Consider **E(M,K) = Rot-{K_1, K_2, …, K_n}(M)**
  - i.e., rotate first character by **K_1**, second character by **K_2**, up through nth character. Then start over with **K_1**, ...
  - **K** = **{ K_1, K_2, ..., K_n }**
- How well do previous attacks work now?
  - Brute force: key space is factor of $26^{(n-1)}$ larger
    - E.g., n = 7 ⇒ 300 million times as much work
  - Letter frequencies: need more ciphertext to reason about
  - Known/chosen plaintext: works just fine
- Can go further with “chaining”, e.g., 2nd rotation depends on **K_2** and first character of ciphertext
  - We just described 2,000 years of cryptography

And Cryptanalysis: ULTRA

- During WWII, the Germans used ***enigma***:
  - System was a "rotor machine": A series of rotors, with each rotor permuting the alphabet and every keypress incrementing the settings
    - Key was the selection of rotors, initial settings, and a permutation plugboard
    - A great graphical demonstration: https://observablehq.com/@tmcw/enigma-machine
- The British built a system (the "Bombe") to brute-force Enigma
  - Required a known-plaintext (a "crib") to verify decryption: e.g. the weather report
  - Sometimes the brits would deliberately "seed" a naval minefield for a chosen-plaintext attack