# 

[Vocab for MT1](#_y7gql1qp3brd)

[Vocab for test 2](#_eymnv83d9cma)

[Section 4](#_34cl3yg3f4sw)

[Section 5](#_anw03bqzyxa)

[Centralized ledgers may have shortcomings, such as:](#_cko7m1ggsd66)

[Other stuff](#_5k7yi6rg99r5)

[Heap corruption](#_2wc19e1h5uzw)

[Typical Heap Corruption Problems](#_ntoi9encozz8)

[Finding Heap Corruption Errors](#_oe1gsmqg2vc9)

[Uncontrolled format string](#_3kh7uj6kdkka)

[xss vulnerability](#_3iynqbcohcje)

[SQL injection](#_36fh6iloxmop)

[Second order SQL injection](#_swp68kfifjz5)

[Buffer overflow](#_9wkdaddgguxj)

[Buffer overflow protection](#_o527m9gu6t5j)

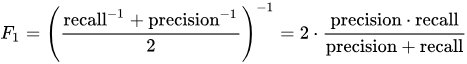
[SQL injection](#_rw8qk4cu7txp)

[SQL injection](#_wxr5tcnr1seo)

[SQL injection](#_2sga9f4gomcf)

# 

# Vocab for MT1

* "ground truth" refers to the accuracy of the training set's classification for supervised learning techniques
* Contextual Anomalies. If a data instance is anomalous in a specific context (but not otherwise), then it is termed as a contextual anomaly (also referred to as conditional anomaly (looks are back and forward of the area. To a odd value in a COS wave is a Contextual Anomaly)
* Collective Anomalies. If a collection of related data instances is anomalous with respect to the entire data set, it is termed as a collective anomaly. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together as a collection is anomalous.
* F1 score  
  
* Harmonic mean  
  
* Intrusion detection system
  + Signature-based IDS refers to the detection of attacks by looking for specific patterns, such as byte sequences in network traffic, or known malicious instruction sequences used by malware.[[3]](https://en.wikipedia.org/wiki/Intrusion_detection_system#cite_note-3) This terminology originates from [anti-virus software](https://en.wikipedia.org/wiki/Anti-virus_software), which refers to these detected patterns as signatures. Although signature-based IDS can easily detect known attacks, it is impossible to detect new attacks, for which no pattern is available.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]
  + [Anomaly-based intrusion detection systems](https://en.wikipedia.org/wiki/Anomaly-based_intrusion_detection_system) were primarily introduced to detect unknown attacks, in part due to the rapid development of malware. The basic approach is to use machine learning to create a model of trustworthy activity, and then compare new behavior against this model. Although this approach enables the detection of previously unknown attacks, it may suffer from [false positives](https://en.wikipedia.org/wiki/False_positives): previously unknown legitimate activity may also be classified as malicious. Most of the existing IDSs suffer from the time-consuming during detection process that degrades the performance of IDSs. Efficient feature selection algorithm makes the classification process used in detection more reliable
* problems of HMM
  + (1)The Evaluation Problem P(O | lamda) = likelihood of sequence occurring
  + (2)The Decoding Problem what is the most likely state sequence in the model that produced the given observations
  + (3)The Learning Problem Given a model LAMDA and a sequence of observations, how should we adjust the model parameters in order to maximize P(O | lamda)
* Type of Anomaly
  + Point Anomalies. If an individual data instance can be considered as anomalous with respect to the rest of data, then the instance is termed a point anomaly. This is the simplest type of anomaly and is the focus of majority of research on anomaly detection. For example, in Figure 1, points o1 and o2, as well as points in region O3, lie outside the boundary of the normal regions, and hence are point anomalies since they are different from normal data points. As a real-life example, consider credit card fraud detection. Let the data set correspond to an individual’s credit card transactions. For the sake of simplicity, let us assume that the data is defined using only one feature: amount spent. A transaction for which the amount spent is very high compared to the normal range of expenditure for that person will be a point anomaly
* Contextual Anomalies. If a data instance is anomalous in a specific context, but not otherwise, then it is termed a contextual anomaly (also referred to as conditional anomaly [Song et al. 2007]). The notion of a context is induced by the structure in the data set and has to be specified as a part of the problem formulation. Each data instance is defined using the following two sets of attributes:
  + Contextual attributes. The contextual attributes are used to determine the context (or neighborhood) for that instance. For example, in spatial data sets, the longitude and latitude of a location are the contextual attributes. In time-series data, time is a contextual attribute that determines the position of an instance on the entire sequence.
  + Behavioral attributes. The behavioral attributes define the noncontextual characteristics of an instance. For example, in a spatial data set describing the average rainfall of the entire world, the amount of rainfall at any location is a behavioral attribute.
* Collective Anomalies. If a collection of related data instances is anomalous with respect to the entire data set, it is termed a collective anomaly. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together as a collection is anomalous. Figure 4 is an example that shows a human electrocardiogram output [Goldberger et al. 2000]. The highlighted region denotes an anomaly because the same low value exists for an abnormally long time (corresponding to an Atrial Premature Contraction). Note that that low value by itself is not an anomaly.

As an another illustrative example, consider a sequence of actions occurring in a computer as shown below: ... http-web, buffer-overflow, http-web, http-web, smtp-mail, ftp, http-web, ssh, smtp-mail, http-web, ssh, buffer-overflow, ftp, http-web, ftp, smtp-mail,http-web …

* Typically, getting a labeled set of anomalous data instances that covers all possible type of anomalous behavior is more difficult than getting labels for normal behavior. Moreover, the anomalous behavior is often dynamic in nature, for example, new types of anomalies might arise
* Output of Anomaly Detection
  + Scores. Scoring techniques assign an anomaly score to each instance in the test data depending on the degree to which that instance is considered an anomaly. Thus the output of such techniques is a ranked list of anomalies. An analyst may choose to either analyze the top few anomalies or use a cutoff threshold to select the anomalies.
  + Labels. Techniques in this category assign a label (normal or anomalous) to each test instance.
* Types of detections
  + Intruder detection
  + Fraud
  + Medical and Public Health Anomaly Detection
  + Industrial Damage Detection
  + Image Processing
  + Anomaly Detection in Text Data
  + Sensor Networks

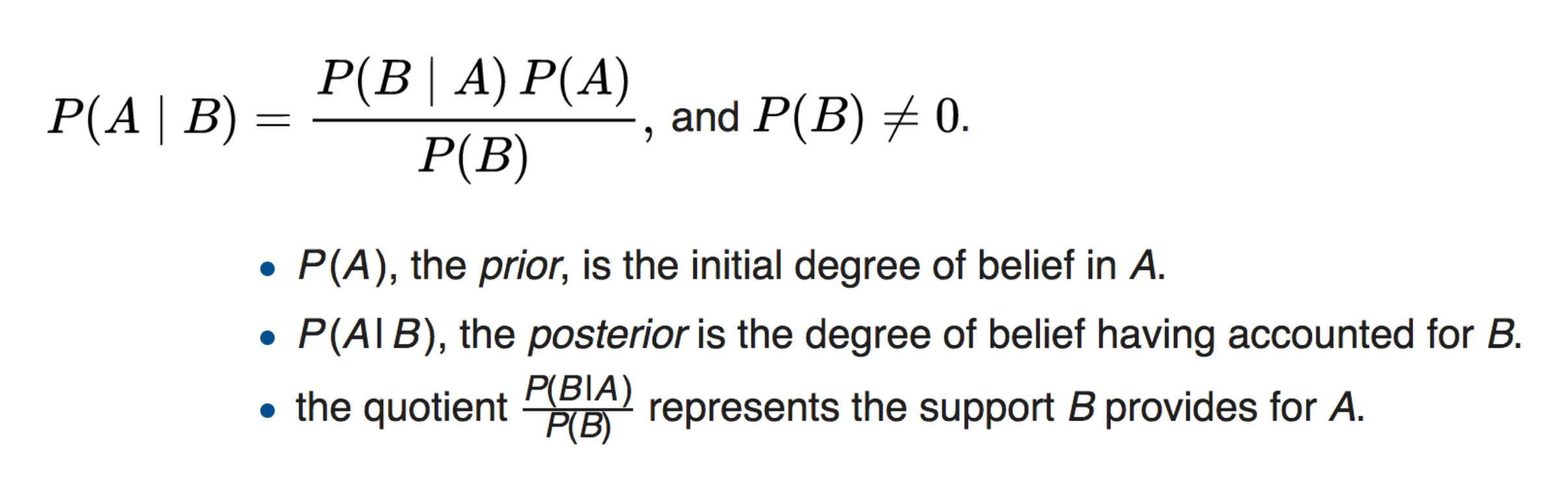
Deterministic vs. Probabilistic???  
3 problems of HMM?

* Anomaly definition Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior. These nonconforming patterns are often referred to as anomalies, outliers,discordant observations, exceptions, aberrations, surprises, peculiarities, or contaminants indifferent application domains.
* Formal anomaly definition an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.

# 

# Vocab for test 2

## Section 4

* measurement as observations that quantitatively reduce uncertainty. Indeed, a mere reduction—not necessarily an elimination—of uncertainty is what one can expect from measurement.
* Shannon proposed a mathematical definition of information as the amount of uncertainty reduction in a signal measured in terms of the entropy3 removed by a signal.
* Assuming that the receiver of information has some prior state of knowledge but has also some uncertainty—meaning, the knowledge is incomplete—, the new information reduces the uncertainty (not necessarily completely) in a quantifiable way
  + This uncertain reduction point of view is what is critical to business. Major decisions made under a state of uncertainty, such as whether to approve large IT projects or new security controls, can be made better— even if just slightly—by reducing uncertainty. Sometimes even small uncertainty reductions can be worth millions of dollars.
* 
* Bayesian (or subjective) interpretation: expectation representing a state of knowledge or as quantification of a personal belief
  + **Reread section 4 slide 41**
* Risk: A state of uncertainty where some of the possibilities involve a loss, catastrophe, or other undesirable outcome.
  + Measurement of Risk: a set of possibilities, each with quantified probabilities and quantified losses.
* Uncertainty: Lack of complete certainty, that is the existence of more than one possibility—the true outcome is not known.
  + Measurement of Uncertainty: a set of probabilities assigned to a set of possibilities. E.g., there is a 35% chance we will have a serious data breach within the next five years
* **Reread section 4 slide 54**
* Inherent risks: normally does not mean a complete lack of **controls** (not a viable alternative) but rather refers to basic and minimal controls that would be considered negligent to exclude. Typical examples include: password protection, firewalls, some frequency of applying security software updates (patches)
* residual risk: after application of controls
* The difference between inherent and residual risks are true discretionary controls that could be considered not absolutely necessary and may be excluded on reasonable grounds.
* Risk should be below risk tolerance. Use controls to achieve this.
* 

## Section 5

* The core ideas behind blockchain technology emerged in 1991. A signed chain of information was used as an electronic ledger for digitally signing documents in a way that could easily show none of the signed documents in the collection had been changed.
* In 2008, the first application to digital cash was described in the original paper on Bitcoin, Bitcoin: A Peer to Peer Electronic Cash System1.
* Features
  + Bitcoin is implemented in a distributed fashion so that no single user controls the currency and no single point of failure exists
  + By using a distributed blockchain and consensus-based maintenance, a self-regulating mechanism was created that ensures that only valid transactions are added to the blockchain.
  + Blockchain technology has become tightly linked to Bitcoin and and other crypto currencies, although it is not restricted to simple fund transfers.
* The security of cryptocurrency ledgers is based on the assumption that the majority of miners are honestly trying to maintain the ledger, having financial incentive to do so
* Bitcoin’s decentralized control verifies and records transactions using a blockchain transaction database in the role of a public distributed ledger.2 The system is peer-to-peer, and transactions take place between users directly, without an intermediary
* Blockchains are immutable digital ledger systems implemented in a distributed fashion, without a central repository and normally also without a central authority.
* **To spend a digital asset, the user must prove possession of the address’s corresponding private key. By digitally signing a transaction with the private key, the transaction can be verified with the public key.**
* Centralized ledgers may have shortcomings, such as:
  + They may be lost or destroyed; a user must trust that the owner is properly backing up the system.
  + Transactions may not be valid; a user must trust that the owner is validating each received transaction.
  + A transaction list may not be complete; a user must trust that the owner is including all valid transactions that have been received.
  + Transaction data may have been altered; a user must trust that the owner is not altering past transactions.
* Transaction are added to the blockchain when a mining node publishes a block. A block contains a set of validated transactions.
* Block contains:
  + The block number, also known as block height
  + The current block hash value
  + The previous block hash value
  + The Merkle tree root hash (defined below)
  + A timestamp
  + The size of the block
  + The nonce value, which is a number manipulated by the mining node to solve the hash puzzle that gives them the right to publish the block
  + A list of transactions included within the block
* Rather than storing the hash of every transaction within the header of a block, a data structure known as a Merkle tree is utilized. A Merkle tree combines the hash values of data together until there is a singular root (a Merkle tree root hash)
* Blockchains are maintained through the consensus of a set of computer systems running blockchain software, known as mining nodes.
* The job of a full node is to store the blockchain data, pass along the data to other nodes, and ensure newly added blocks are valid

# Other stuff

* A **file inclusion vulnerability** is a type of [vulnerability](https://en.wikipedia.org/wiki/Vulnerability_(computing)) that is most commonly found to affect [web applications](https://en.wikipedia.org/wiki/Web_applications) that rely on a scripting [run time](https://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)). This issue is caused when an application builds a path to executable code using an attacker-controlled variable in a way that allows the attacker to control which file is executed at run time. A file include vulnerability is distinct from a generic [Directory Traversal Attack](https://en.wikipedia.org/wiki/Directory_traversal_attack), in that directory traversal is a way of gaining unauthorized [file system](https://en.wikipedia.org/wiki/File_system) access, and a file inclusion vulnerability subverts how an application loads code for execution. Successful exploitation of a file include vulnerability will result in remote code execution on the [web server](https://en.wikipedia.org/wiki/Web_server) that runs the affected web application.
* The File Inclusion vulnerability allows an attacker to include a file, usually exploiting a "dynamic file inclusion" mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

## Heap corruption

Common scenarios include:

* Writing outside the allocated space of an array (char \*stuff = new char[10]; stuff[10] = 3;)
* Casting to the wrong type
* Uninitialized pointers
* Typo error for -> and .
* Typo error when using \* and & (or multiple of either)
* Mixing new [] and new with delete [] and delete
* Missing or incorrect copy-constructors
* Pointer pointing to garbage
* Calling delete multiple times on the same data
* Polymorphic baseclasses without virtual destructors

### Typical Heap Corruption Problems

Due to the dynamic nature of allocating and deallocating memory, the heap is vulnerable to the following typical corruption problems:

* *boundary overrun*: a program writes beyond the malloc region.
* *boundary underrun*: a program writes in front of the malloc region.
* *access to uninitialized memory*: a program attempts to read memory that has not yet been initialized.
* *access to freed memory*: a program attempts to read or write to memory that has been freed.
* *double frees*: a program frees some structure that it had already freed. In such a case, a subsequent reference can pick up a meaningless pointer, causing a segmentation violation.
* *erroneous frees*: a program calls free() on addresses that were not returned by malloc, such as static, global, or automatic variables, or other invalid expressions. See the malloc(3f) man page for more information.

### Finding Heap Corruption Errors

To find heap corruption problems, you must relink your executable with the -lmalloc\_ss library instead of the standard -lmalloc library. By default, the -lmalloc\_ss library catches the following errors:

* malloc call failing (returning NULL)
* realloc call failing (returning NULL)
* realloc call with an address outside the range of heap addresses returned by malloc or memalign
* memalign call with an improper alignment
* free call with an address that is improperly aligned
* free call with an address outside the range of heap addresses returned by malloc or memalign

## Uncontrolled format string

* <https://stackoverflow.com/questions/7459630/how-can-a-format-string-vulnerability-be-exploited>
  + **Uncontrolled format string**[[1]](https://en.wikipedia.org/wiki/Uncontrolled_format_string#cite_note-1) is a type of [software vulnerability](https://en.wikipedia.org/wiki/Software_vulnerability) discovered around 1989 that can be used in [security exploits](https://en.wikipedia.org/wiki/Security_exploit). Previously thought harmless, **format string exploits** can be used to [crash](https://en.wikipedia.org/wiki/Crash_(computing)) a program or to execute harmful code. The problem stems from the use of [unchecked user input](https://en.wikipedia.org/wiki/Unchecked_user_input) as the [format string](https://en.wikipedia.org/wiki/Format_string) parameter in certain [C](https://en.wikipedia.org/wiki/C_(programming_language)) functions that perform formatting, such as [printf()](https://en.wikipedia.org/wiki/Printf). A malicious user may use the %s and %x format tokens, among others, to print data from the [call stack](https://en.wikipedia.org/wiki/Call_stack) or possibly other locations in memory. One may also write arbitrary data to arbitrary locations using the %n format token, which commands printf() and similar functions to write the number of bytes formatted to an address stored on the stack.
  + [src](http://www.cis.syr.edu/~wedu/Teaching/cis643/LectureNotes_New/Format_String.pdf) : /www.cis.syr.edu/~wedu/Teaching/cis643/LectureNotes\_New/Format\_String.pdf

• Viewing the stack printf ("%08x %08x %08x %08x %08x\n"); – This instructs the printf-function to retrieve five parameters from the stack and display them as 8-digit padded hexadecimal numbers. So a possible output may look like: 40012980 080628c4 bffff7a4 00000005 08059c04

• Viewing memory at any location – We have to supply an address of the memory. However, we cannot change the code; we can only supply the format string. – If we use printf(%s) without specifying a memory address, the target address will be obtained from the stack anyway by the printf() function. The function maintains an initial stack pointer, so it knows the location of the parameters in the stack. – Observation: the format string is usually located on the stack. If we can encode the target address in the format string, the target address will be in the stack. In the following example, the format string is stored in a buffer, which is located on the stack.

If we can force the printf to obtain the address from the format string (also on the stack), we can control the address.

printf ("\x10\x01\x48\x08 %x %x %x %x %s");

\x10\x01\x48\x08 are the four bytes of the target address. In C language, \x10 in a string tells the compiler to put a hexadecimal value 0x10 in the current position. The value will take up just one byte. Without using \x, if we directly put "10" in a string, the ASCII values of the characters ’1’ and ’0’ will be stored. Their ASCII values are 49 and 48, respectively.

## xss vulnerability

* XSS enables attackers to [inject](https://en.wikipedia.org/wiki/Code_injection) [client-side scripts](https://en.wikipedia.org/wiki/Client-side_script) into [web pages](https://en.wikipedia.org/wiki/Web_page) viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass [access controls](https://en.wikipedia.org/wiki/Access_control) such as the [same-origin policy](https://en.wikipedia.org/wiki/Same-origin_policy). Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities
* In such case an attacker can easily insert javascript code which would run under the site's context. By doing so the attacker is able to access other pages on the same domain and can read data like CSRF-Tokens or the set cookies.

## SQL injection

* **SQL injection** is a [code injection](https://en.wikipedia.org/wiki/Code_injection) technique, used to [attack](https://en.wikipedia.org/wiki/Attack_(computing)) data-driven applications, in which nefarious [SQL](https://en.wikipedia.org/wiki/SQL) statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).[[1]](https://en.wikipedia.org/wiki/SQL_injection#cite_note-1) SQL injection must exploit a [security vulnerability](https://en.wikipedia.org/wiki/Security_vulnerability) in an application's software, for example, when user input is either incorrectly filtered for [string literal](https://en.wikipedia.org/wiki/String_literal) [escape characters](https://en.wikipedia.org/wiki/Escape_sequence) embedded in SQL statements or user input is not [strongly typed](https://en.wikipedia.org/wiki/Strongly-typed_programming_language) and unexpectedly executed. SQL injection is mostly known as an attack [vector](https://en.wikipedia.org/wiki/Vector_(malware)) for websites but can be used to attack any type of SQL database.
* SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

### **Second order SQL injection**

* Second order SQL injection occurs when submitted values contain malicious commands that are stored rather than executed immediately. In some cases, the application may correctly encode an SQL statement and store it as valid SQL. Then, another part of that application without controls to protect against SQL injection might execute that stored SQL statement. This attack requires more knowledge of how submitted values are later used. Automated web application security scanners would not easily detect this type of SQL injection and may need to be manually instructed where to check for evidence that it is being attempted.

## Buffer overflow

* Buffers are areas of memory set aside to hold data, often while moving it from one section of a program to another, or between programs. Buffer overflows can often be triggered by malformed inputs; if one assumes all inputs will be smaller than a certain size and the buffer is created to be that size, then an anomalous transaction that produces more data could cause it to write past the end of the buffer. If this overwrites adjacent data or executable code, this may result in erratic program behavior, including memory access errors, incorrect results, and [crashes](https://en.wikipedia.org/wiki/Crash_(computing)).
* Exploiting the behavior of a buffer overflow is a well-known [security exploit](https://en.wikipedia.org/wiki/Exploit_(information_security)). On many systems, the memory layout of a program, or the system as a whole, is well defined. By sending in data designed to cause a buffer overflow, it is possible to write into areas known to hold [executable code](https://en.wikipedia.org/wiki/Execution_(computing)) and replace it with [malicious code](https://en.wikipedia.org/wiki/Malicious_code), or to selectively overwrite data pertaining to the program's state, therefore causing behavior that was not intended by the original programmer. Buffers are widespread in [operating system](https://en.wikipedia.org/wiki/Operating_system) (OS) code, so it is possible to make attacks that perform [privilege escalation](https://en.wikipedia.org/wiki/Privilege_escalation) and gain unlimited access to the computer's resources. The famed [Morris worm](https://en.wikipedia.org/wiki/Morris_worm) in 1988 used this as one of its attack techniques.
* [Wiki](https://en.wikipedia.org/wiki/Buffer_overflow#Example)
* Use a fat pointer, they know the size of the given data structures?

### Buffer overflow protection

* [wiki](https://en.wikipedia.org/wiki/Buffer_overflow_protection)

## SQL injection

* S
* Ss
* S

## SQL injection

* S
* Ss
* S

## SQL injection

* S
* Ss
* S