Introduction to Software Security

Chapter 1.2: Basic Concepts and Terminology

Loren Kohnfelder
loren.kohnfelder@gmail.com

Elisa Heymann
elisa@cs.wisc.edu

Barton P. Miller
bart@cs.wisc.edu

DRAFT — Revision 1.1, January 2019.

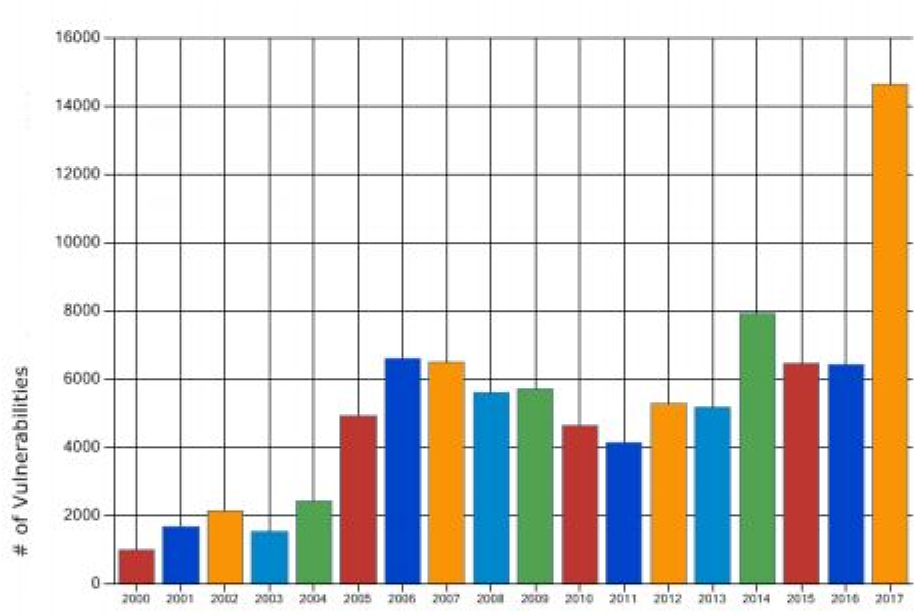
Objectives

- Establish a common vocabulary for security since "language shapes thought".
- Understand the pillars of security: Confidentiality, Integrity, and Availability (C-I-A).
- Understand the kind of threats that your system faces.
- Understand how attacks to software security can come through vulnerabilities in the code.

An unsolved problem

As you begin studying software security, you can rest assured that there is no danger of the problem being completely eliminated or your expertise made moot. Security vulnerabilities are, if anything, on the increase. And years after some of the earliest most fundamental flaws were first discovered, they continue to recur and cause serious harm with no end in sight.

The number of incidents registered in the National Vulnerability Database (NVD) maintained by the US National Institute of Standards and Technology (NIST) provides a very rough measure. The figure below clearly demonstrates that security problems are ongoing with no evidence of becoming a solved problem anytime soon. Common Vulnerabilities and Exposures (CVE) numbers are assigned to issues reported to the NVD (for example, the well-known Heartbleed bug in OpenSSL that compromised the security of computers and related equipment worldwide is identified as CVE-2014-0160). There is no well established statistic that captures the overall trend in software security, and CVE counts are not entirely accurate as such; nonetheless, the NVD may be the best publicly available data.



CVE (Common Vulnerability and Exposures) counts from 2000 to 2017
Source: NVD (National Vulnerability Database) from NIST (nvd.nist.gov).

The C-I-A Pillars

The three pillars of information security are Confidentiality, Integrity, and Availability (C-I-A), expressing the ideals we seek to protect and that attackers seek to undermine. These are the fundamental expectations for any information system: good security means that data will not be disclosed inappropriately (confidentiality), data and code will not be tampered or destroyed (integrity), and proper access will be promptly given at all times (availability).

These principles are so basic that it is worth defining and giving examples of each of the three pillars.

Confidentiality

Confidentiality refers to protecting against the disclosure of information to unauthorized parties. Potentially, even slight information leakages that provide any clues about the protected data represent a compromise of this principle. For example, revealing just the first letter or the number of letters in a name, combined with other information or guesses represents a partial disclosure.

Examples of attacks on confidentiality include:

- Reading stored data without authorization.
- Performing statistical analysis on restricted data.
- Reading messages (communication) without authorization.
- Detecting if communication traffic is present or not.

Integrity

Integrity refers to protecting information from being created, modified, or destroyed by unauthorized parties. Potentially, even the plausible threat that data may have been tampered with (and no means of knowing whether it was or not) could be considered an exploit of Integrity.

Examples of attacks on integrity include:

- Modifying, destroying, or creating new data files.
- Changing program code with the goal of causing errors, failures, crashes, or unexpected behavior.
- Modifying, deleting, replicating, change of order, or creating extra messages.

Availability

Availability refers to ensuring that authorized parties are able to access the system and its resources when needed. Successful compromise of availability is often called a Denial-of-Service (DoS) attack. While destruction of a system or its data certainly does make it unavailable, this is normally considered loss of integrity; the term "availability" is generally concerned specifically with temporary impediments to access.

Examples of attacks on availability include:

- Swamping an internet-connected service with requests, impeding its ability to serve clients.
- Removing applications, components, or files, rendering the system completely or partial inoperable.
- Crashing a server, preventing it from serving clients.
- Overloading some part of a system, including the hardware or software service.

Attack Paths

Understanding how to achieve security inherently begins with a look at how it potentially could be compromised. Consider by analogy an example of a bank vault containing a lot of money that must be protected from bad guys.

There are many attack paths available to a bank robber: enter through the front door during business hours, break into the employee-only rear entrance, dig a tunnel, and so forth. The sum of all possible starting points is called the **attack surface**, so that where building defenses begins. The money is an **asset** — the object of protection — which in software systems is often data, code, equipment or other digital resource.

In this bank robbery example, suppose the vault was given the easily guessed combination of 1-2-3 (that's the specific vulnerability), and so the robber poses as a customer, walks up to the vault and in a few seconds gains access while nobody is looking (insufficient access control, surveillance, and guarding are additional contributing weaknesses). So the **attack path** now consists of the sequence of front door entry during business hours, walking up to the vault, and trying to guess the combination. The sum of these steps completes the exploit.

Once in the vault, in terms of the C-I-A fundamentals, the robber has many options:

- Count the money to learn the bank's funds. (confidentiality)
- Replace with money with fake bills. (integrity)
- Hide the money somewhere inside the bank. (availability)

Obviously, real bank robbers are focused on taking money for themselves rather than the C-I-A model, which is intended for information systems, but the same principles can be applied.

Security Terminology

The field of information security has developed specific terminology for the underlying fundamental principles. As with software in general, we reuse existing English words adding specific meanings so it is easy to pick up the vocabulary, but it is worth taking the time to explicitly define the specialized meanings clearly.

Unfortunately, the specialized terminology and meanings of these words is not uniformly applied so you may hear some variations and there is no authoritative right and wrong. Some of the common alternative terms will be noted here parenthetically when the meaning is similar. Generally speaking, these differences in language usually are harmless and easy to adapt to, so it is not something to try to fix. In this book, we use what we believe to be the most common and logical words and meanings from our diverse experiences, and make every effort to use these terms consistently throughout the text.

> *"Software bugs are errors, mistakes, or oversights in programs that result in unexpected and typically undesirable behavior." — The Art of Software Security Assessment*

"All software has bugs." The origin of this statement seems lost in time, but after many decades of software development, it remains truer than ever and hardly controversial. Given a set of bugs (also called defects or **weaknesses**) in a piece of software, some subset of those bugs will inevitably be useful to an **attacker** (bad guy, malefactor) to cause harm: these bugs are thus termed **vulnerabilities**.

> *"A vulnerability is a defect or weakness in system security procedures, design, implementation, or internal controls that can be exercised and result in a security breach or violation of security policy." - Gary McGraw, Software Security*

Reducing vulnerabilities is the central goal of this book. Eliminating a vulnerability, or somehow making it less likely to be found or exploited is called **mitigation.** Since the complete elimination of all bugs in a complex piece of software usually is not practical, we try to build in multiple layers of defense so that a vulnerability is more difficult to find or has a less impact.

The various kinds of hypothetical harm an attacker might achieve on a system are called **threats**. These threats are inherent — if the system holds a secret, there will always be a threat of it being revealed — and cannot be eliminated. Threats are identified with respect to protected resources called **assets**, things of value. In a computer system, these things are usually data, though they might also include the computing resource itself or a physical device controlled by the computer. The reason that it is important to recognize threats is that they can be an extremely useful lens for determining where important vulnerabilities may exist.

> *"The process of attacking a vulnerability in a program is called exploiting." — The Art of Software Security Assessment*

When a real attacker uses a vulnerability to actually cause harm, that attack is called an **exploit**. Getting past protective defenses is called an **intrusion**.

Threats

Secure software is achieved by anticipating the many attack scenarios and protecting against hypothetical attacks that might violate confidentiality, integrity, or availability. Here is a technical definition of an information security threat:

"A potential cause of an incident that may result in harm of systems and organization." — ISO 27005

Put simply, threats are the answers to the question: what could go wrong? It is important to remember that threats are hypothetical and cannot be eliminated unless you are needlessly doing something risky and can dispensed with it entirely. If your system holds an important secret, then the threat of disclosure or destruction of that secret is inevitably a threat to be dealt with.

Security improvements begin with evaluating potential threats to your system. Since these threats will be diverse and many, you need to prioritize them to focus your efforts appropriately. From this strategic holistic perspective, you begin the process of identifying places where threats can be reduced, finding and then fixing the actual vulnerabilities. It is critical to add here that the vulnerabilities can be anywhere along the attack path of a possible exploit, so the location of the vulnerability can be distant and seemingly unrelated to the actual asset the attack may compromise.

STRIDE

STRIDE is a threat classification model developed at Microsoft as part of the company's Trustworthy Computing initiative (one of the co-authors of this book assembled the letters of the acronym for the original paper), now integrated into the company's Software Development Lifecycle (SDL) Threat Modeling tool. The model is a useful way to categorize common threats: when filing a security vulnerability report, noting the STRIDE category is a quick and easy way to characterize the issue for others to quickly see.

The six STRIDE threat categories are as follows, each to be explained in detail with examples following:

- **S**poofing of an identity
- **T**ampering
- **R**epudiation
- **I**nformation disclosure
- **D**enial of service
- **E**levation of privilege

STRIDE is also handy as a kind of checklist of types of threat to consider, though it is important to bear in mind that it may not necessarily cover every possible threat to every system. For example, when looking at a system component block diagram or reviewing the security of a protocol, it helps to write out the six letters to make it easy to remember the corresponding threat categories, and then apply each to the analysis at hand for separate detailed consideration. For example, S: Is there an identity spoofing threat here? T: What data might be subject to tampering and what would the implications of that? And so forth for R, I, D, and E.

Spoofing

Spoofing of an identity is when a person or program successfully impersonates another, thereby concealing the attacker's true identity and often gaining unauthorized access in the process.

- In our vault example, an attacker impersonates Rich Customer and convinces the bank to allow them to have access to Rich Customer's money.
- E-mail address spoofing.
- Man-in-the-middle such as impersonating a service.
- Web page referrer spoofing.

Tampering

Tampering includes unauthorized modification of data or code that alters the system behavior.

- In our vault example, evil attacker manipulates the security cameras so the attacker can access the safe without being seen.
- Installing backdoor access (a secret, unauthorized access path).
- Disabling security monitoring.
- Subverting authorization.
- Bypassing valid license checks.
- Altering control flow.
- Injecting malicious code into a program.

When tampering occurs, it can be much more pernicious if it goes unnoticed for a long period of time. In a commonly used devious form of attack, code is left behind (often termed an **implant**) that may be hidden or intentionally left inactive for a period of time, making it difficult to detect. Persistent implants can survive reboots and system updates and can be very difficult to eradicate.

Repudiation

Repudiation is when a user plausibly denies performing a specific action — a fancy word for, "It wasn't me!" Non-repudiation is the good security property of having strong evidence so, in the case of a dispute, it can be reasonably proven to a third party exactly who did what (and, ideally, when).

- In our vault example, evil Attacker denies that they took Rich Customer's money from the vault. They claim that they were at the movies at the time of the crime.
- A user signing a document and then claiming that the action was performed by someone who stole their credentials.

- A user claiming that they never sent an email that they actually sent.

Information disclosure

Information disclosure is releasing information to an actor that is not explicitly authorized to have access to that information.

- Evil attacker quietly snaps a photo of Rich Customer's transaction receipt, and then posts the receipt to Twitter, revealing their account balance.
- An attacker extracting data files from your system (exfiltration).
- Error message revealing too much information, for example, exposing the full path of the program.
- Debug error messages left in the deployed software.
- Using a weak encryption method.

Denial of Service

Denial of service is making a resource, such as a host, server, network or application, unavailable for legitimate users. This category matches loss of availability as described within the C-I-A principles.

- In our vault example, preventing Rich Customer from taking their money from the vault, by putting a fake "bank closed" sign on the door.
- Saturating a service with a large number of requests.
- Taking control of a large number of computers and using them to bombard a server in what is called a distributed denial of service (DDoS, pronounced "dee-doss") attack.
- Exploiting a buffer overflow and making the system crash.

Elevation of privilege

Elevation of privilege is getting access to more resources or functionality than a user is normally allowed by giving them more privileges than intended by the developer or system administrator.

- In our vault example, a bank employee steals the bank president's ID card so that they can have access to all the bank records.
- Vertical escalation: gaining root or administrator access.
- Horizontal escalation: accessing information associated with a different user.

Security Risk

The threats to large information systems are great. Such threats arise based on the extent to which such a system processes sensitive data, has a large attack surface, and has a high level of implementation complexity. Given the difficulty of securing such a system, it is essential to take a strategic approach focusing our protection efforts on the parts of the system with highest risk. Since time and resources available for security work are always limited, the work inevitably runs up against practice constraints — this is why all software is said to have vulnerabilities. Ideally, security effort continues working in priority order until further effort enters the realm of diminishing returns.

Risk management is an established practice that arose in the financial sector (insurance, investment, and gaming) to identify sources of risk and attempts to quantify them for the purpose of predicting potential losses and implementing cost effective mitigations. Put simply, risk management strategy accepts that risk is unavoidable and attempts to “put it in a box” in the sense of putting a cap on the worst case, spreading the loss over time, investing in preventative efforts, and monitoring actual losses so that management can accurately understand their actual risk stance.

Financial risk management allows quantitative assessments of risk by converting impacts into dollars, however this is more challenging in the information security space for several reasons.

- Financial assets are based on money, which is easily measured in precise amounts.
- Money is fungible (one dollar is as valuable as another) but information is not.
- Money lost can be be paid back but information once lost can be irretrievable.
- Insured risks have been accurately measured by actuaries and are well understood, but software security risks are newer and change rapidly.
- Financial institutions are carefully regulated, but software is mostly unregulated and companies often conceal details of security incidents so little public data is available.
- The financial impact of security incidents is difficult to measure, and there are additional risks such as losses to reputation or customer confidence that are difficult to quantify.

As a result, for our purposes, software risk assessment is more subjective and less quantitative, but the underlying principles remain useful to inform those decisions. Since security risk components cannot be assigned accurate numerical values one common approach is to use “T-shirt sizes” to assign probabilities and gauge impacts as Small, Medium, Large, X-Large.

Risk = Impact × Likelihood

The impact x likelihood formula is the foundational way to quantify risk (though modern financial enterprises use more complex formulas). To understand this formula, consider a couple of examples from the financial sector. Auto insurance rates are calculated based on historical data of collision statistics that detail the frequency that these collisions occur and the distribution of settlement costs. Interest rates are calculated in a similar way: a bank may make small loans without much analysis, knowing that a certain percentage of those loans will not be repaid, but the loss is relatively limited. For large loans, the bank will study the borrower, ask for collateral, and generally work harder to either lower the risk of non-payment or have some other recourse since so much money is at stake. In both cases, the lower impact events may be more likely, and that balances out against making the higher impact risks less likely to occur.

Now, let’s consider risk assessment for software security. Impact can be assessed against the three C-I-A pillars:

- Confidentiality: how sensitive is the data and how much of it might be disclosed?
- Integrity: how valuable is the data, how much could be tampered with, and how likely would that be detected?
- Availability: how long and to what degree would the system unavailable?

There is a wide range of how quantitative and how accurately these can be measured. For example, if the private key to a BitCoin wallet is compromised, then that is worth the present dollar value of the cryptocurrency it protects. On the other hand, if a private family photo is publicly revealed, it could be upsetting, lead to bad publicity, but no amount of money or effort can undo the damage done (so this is entirely subjective).

Evaluating the likelihood of exploit is usually more complicated to determine. It is impossible to fully anticipate attackers: what their motivations might be, how much inside information they may have, and what is their level of expertise and capability. In addition, there may be a large amount of luck involved in whether or not a vulnerability is discovered and then exploited maximally. With open source software, you can assume that the attacker has the same knowledge that you do. However, it is risky to assume that you have much of an advantage if your software is based on proprietary source code. This amounts to **Security by Obscurity**, which is a dangerous strategy.

The one exception to the difficulty of assessing likelihood is when detection is virtually certain. CVE-2018-7602 is a good example of this: a critical remote code execution vulnerability in the popular website system Drupal. Since, by design, it is trivial to scan a website and determine that it runs on Drupal and learn what version is installed, all vulnerable sites were well known. Once the fix was deployed publicly, attackers easily learned about the vulnerability in detail and how to exploit it. Simply by scanning the installed version they could automate attacks on any unpatched systems. So the likelihood of such an attack was high — and, in fact, was quite common. Obviously, such high-impact high-likelihood risks are of the utmost priority.

The multiplicative risk formula is a useful lens through which to consider the value of various defenses. For example, well designed password-based login systems should restrict the number and frequency of login attempts as a defense against brute-force password guessing. Since the risk of a lucky guess (and weak passwords make this all the easier) cannot be eliminated, making it harder to guess does significantly reduce the likelihood. It is a little more inconvenient for a good user who has mistyped their own password to wait a little longer between attempts, but the risk reduction is arguably worth it.

Reducing high risks on the impact side with specific mitigations is another valuable approach, focusing on the other multiplier in the equation. Any enterprise that maintains a large customer database needs to defend against the prospect of a major disclosure of all that sensitive data (as has happened famously to companies like Target, Equifax, and many more). A standard database configuration will include an administrative function of dumping the full database given the proper authorization, representing a huge risk. Possible mitigations might include only allowing download of an encrypted copy of the data (requiring a key to which only administrators would have access), or restricting the rate of data export such that it would take a long time to get all the data (yet fast enough to be useful for routine backup purposes). In these ways, the impact of a compromise may be reduced, lowering the overall risk.

A counterpoint about the limits of applying the standard risk formula to software security is worth mentioning. Quantifying both impact and likelihood in a meaningful way can be quite difficult. By the commutative principle of multiplication, this risk formula implies that in theory the case where the likelihood is extremely low and the impact is very high is equal to the case where the likelihood is high and the impact is very low. Clearly these are very different risks, since an example of the former might be the world's chances of nuclear destruction and the latter an obvious bug in an obscure software application. The point here is that software security risk assessment is ultimately subjective and should not be reduced to a mindless calculation that could miss seeing the forest for the trees.

Cost of Insufficient Security

In theory, it is hard to imagine anyone seriously arguing that software security is frivolous and not important, yet somehow in practice diligent security professionals find themselves forever needing to justify the need for this important work. So it is worth elaborating why security is important, which amounts to pointing out the significant costs of insufficiently security. Part of the problem is the psychological effect. In practice, good software security is difficult to appreciate because it is so hard to notice; good security results in the absence of bad things happening. Yet, when the inevitable exploit does happen, then and only then is the spotlight on security in a bad light.

Software security is critical to protect our digital systems especially as these become more ubiquitously part of so many growing aspects of modern life, business, government, and culture at a time when everything is increasingly interconnected on the Internet (where attacks are always possible from literally anywhere at any time).

Attacks are expensive and may directly or indirectly affect assets including: money, reputation, data, communications, services, infrastructure, intellectual property. Examples of significant losses due to poor software security regularly appear in the mainstream media, and the potential for future damage is even larger and growing.

The explosive growth of digital systems, exemplified but by no means limited to giant platforms such as Google, Amazon, Apple, and Microsoft, is only possible because of the great value digital information has transforming the economy and most aspects of modern life. The value of the data they process greatly exceeds to cost of the physical infrastructure (such as data centers and networks) supporting them, and that high value represents high risk both in terms of massive complexity as well as attractiveness to malicious attackers. Experience shows time after time that if anything we need to practice more software security to higher standards in order to keep up with growing threats.

Summary

- Software security remains an unsolved problem; even old well-known vulnerabilities persist.
- Security terminology expresses subtle meanings and it is important to use terms correctly.
- The pillars of information security are confidentiality, integrity, and availability
- Attackers exploit vulnerabilities in the code to successfully exploit them.
- STRIDE is a general system of information security threat classification.
- Every system (and all assets) faces many threats.
- Software security is important because the cost of insufficient security can be devastating.

Exercises

1. List your own creative examples of the three C-I-A pillars of information security.

2. Study the design of a well known open source technology or component and enumerate some of its attack paths. Suggested subjects: OpenSSH, Sendmail, OpenVPN, VirtualBox.
3. From memory enumerate the STRIDE categories and list some applicable threats for a real or theoretical software component. Suggested subjects: a game console, a home assistant (like Alexa or Siri), a smart connected kitchen appliance, or something more creative.
4. Pick a few issues from an open bug tracking system (or make up some hypothetical bugs yourself) and prioritize them for security risk. Suggestions: Chrome bugs, Firefox bugs, Linux kernel bugs.
5. Choose a well publicized software security incident (feel free to invent undisclosed details) and enumerate all of the resultant costs that resulted fixing and recovering from the exploit.
6. Write a short essay on why computer security problems persist or even grow, despite much study and great efforts fixing them.