EECE 457 Notes — General

Security problems and security policies

A security policy is a set of rules listing the desired properties we want a security system to enforce, for example “Only Human Resources employees may be access HR files.” A security system is a collection of technology, policies, and practices that should guarantee the that a security policy is satisfied.

That term “security policy” can cause some confusion, because we also use the word “policy” in other aspects. The word “policy” by itself may refer to public policy, matters of law that pertain to security, such as copyright law or export control law; and a security system will feature both technologies and policies to solve a problem.

Security versus Secrecy

Much of our focus is information security (and computer security, and network security,) and this implies a strong focus on achieving secrecy. Secrecy is one major type of security goal, that amounts to controlling who can possess or access information. There are of course other security goals: if we want to reduce shoplifting at a retail store, we aren’t trying to prevent people from knowing things, we’re trying to prevent people from taking things.

Why do we spend so much focusing on secrecy?

1. Because in computing applications, many security problems *are* secrecy problems.

2. Because other security problems can often be reduced to secrecy problems. I may prevent you from stealing items by placing them in a safe, whose access depends on knowing a secret combination; or I may prevent you from spending my money by using a secret password to access my bank/Amazon/PayPal account.

   This is a potentially dangerous abstraction, however, because gaining that secret combination or password is not the only way to beat the system. Nevertheless, many security systems rely on secrecy guarantees as “load bearing” elements.

3. Secrecy is quantifiable and fungible. I can choose a secret key and mathematically express the expected amount of time required for an attacker to guess it. Furthermore, secret parameters can be easily generated, replaced, stored in bit strings and transferred from place to place. The same can not be said for a perimeter wall around a military base.

4. We have plenty of tools, such as cryptographic primitives and protocols, to solve secrecy problems, and these tools allow us to achieve pretty amazing and sophisticated secrecy arrangements.

5. Secrecy is a matter of information, which can be treated mathematically and processed by algorithm. This gives us a comfortable mathematical certainty: for example we can sometimes provide mathematical proofs of secrecy or security properties, and we can rely on computer programs to exactly enforce rules in a predictable way. We do not have such

certainty when a problem requires human beings to perform procedures, or when we must rely on imperfect physical security systems such as door locks.

But again, this is a dangerous abstraction. Information is stored and processed by physical systems, and human beings are always involved. We never really have "comfortable mathematical certainty," unless we make unwarranted assumptions.

To summarize, then: much of this course will focus on secrecy engineering, but we must always be wary about the difference between secrecy and security in general. We must be careful when attempting to solve a security problem by reducing it to a secrecy problem, because we may then be ignoring additional risks and attacks. We must also be wary of human factors and physical factors in a security system, and not view problems as an abstract matter of computer algorithms processing bit strings.

Harm, threat and vulnerability analysis.

In this class, we distinguish between three "variables" in a security problem, that are vaguely analogous to the flow, effort and impedance variables we encounter in courses like circuits or dynamics. This is only a vague analogy, however, and there are predictable physical laws that relate these things.

Harm is the outcome a security policy and security system is attempting to prevent. Harm can sometimes be expressed in objective units, such as dollars lost to shoplifting losses. Often a security problem involves multiple types of harm, some that may not be immediately obvious. For example, we may want to prevent unauthorized access to a computer, but an attacker may also deny others access through a denial of service attack.

A **threat** is a potential for harm to manifest, vaguely similar to voltage as a potential for current to manifest. A threat is due to an adversary having both the general capability and incentive to cause harm—we will call this an "attack"—and the overall threat level is a matter of the number, capability, and incentive of adversaries.

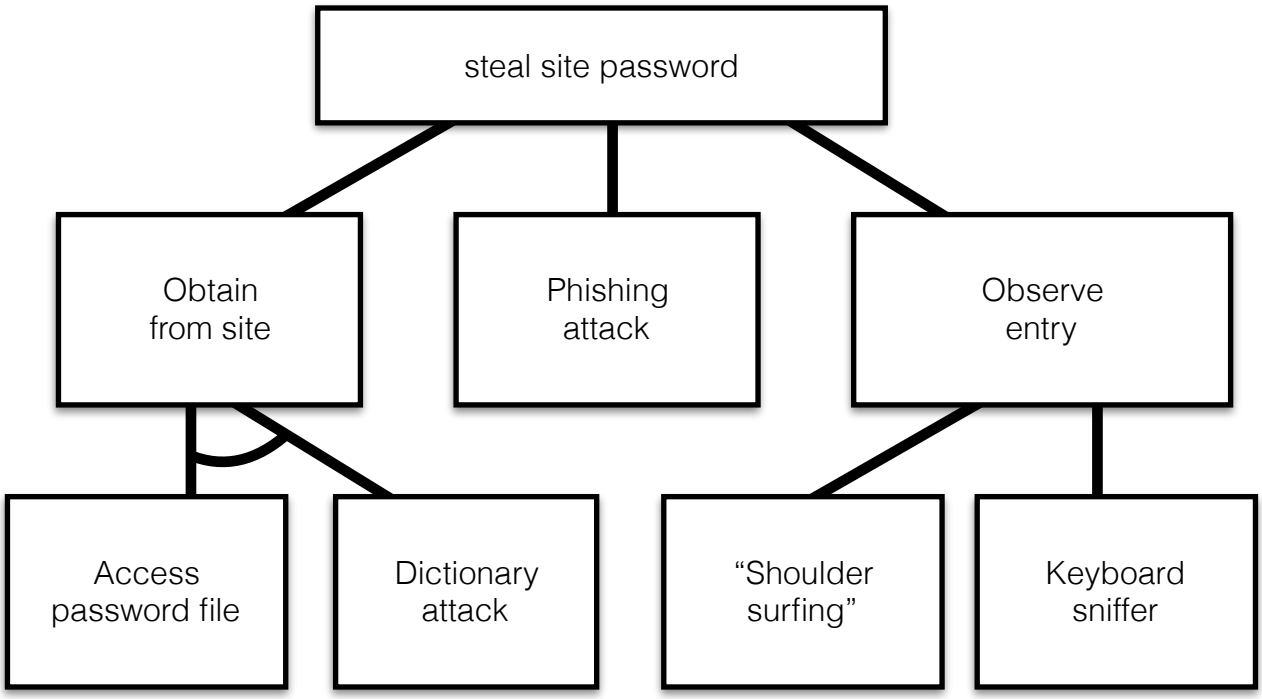
A **vulnerability** is an opportunity for an attack to manifest, either due to a flaw in a security system or the lack of a security system. Again, this is vaguely analogous to a lack of impedance in a circuit. Note the subtle difference between a vulnerability and an attacker's capability. Your vulnerability might be having a machine on the Internet; my capability might be the ability to program a computer or run attack software.

When addressing a security problem, it is valuable to maintain a document summarizing all three of these variables, separately. When you are asked to analyze a security problem you will provide a harm assessment, a threat model, and a vulnerability analysis. Each of these are really just bulleted lists. A harm assessment is a list of relevant sources of harm, essentially a list of "what can go wrong," along with some analysis of the severity or cost of each bad thing. A threat model is a list of adversaries, each category of adversary including a rough estimate of their incentive, capability, and number.

A vulnerability analysis is a list of attacks that can be performed, owing to bad security or a lack of security. This can often be expressed using a threat tree. A threat tree is a tree whose root is a type of harm (that should be listed in your harm assessment,) and whose nodes describe how

it can be achieved. A bad outcome may be achieved by very different attacks, and a tree helps to express this so that we do not become single-minded in our attempt to prevent just one attack.

A threat tree is an AND-OR tree. Each node is a possible attack or action, and the children represent the steps needed for that action to be achieved. It may be that only one child step is needed, or that all child steps are needed; we mark this by drawing an arc connecting child nodes if they must all be performed (an AND relationship,) and drawing them without annotation if only one of them must be performed (an OR relationship.) An example is below.



Practically, diagrams of this type can be difficult to alter or maintain, so the same information can be written as a bulleted list with nested bullets, as long as there is a clear convention showing which bullets are ANDs and which are ORs.

Sometimes threat trees are augmented with nodes listing countermeasures to attacks. We will use threat trees to document the security problem under analysis, separate from any proposed solution, and so we will restrict our trees to listing attacks and vulnerabilities.

It is important to have a document detailing the harm, threat and vulnerability aspects of a security problem, and to extend it as we uncover new threats and vulnerabilities, because it provides critical perspective for addressing a security problem. An understanding of harm sources and threats help us to rank problems by their relative severity. If a client wants you to solve a very specific technical problem involving a single, specific threat, an overall harm-threat-vulnerability analysis is still critical. The client may not be aware of the extent of the problem being addressed.

Technology versus policy solutions

A security system will often include both technological solution and policy solutions. A technological element, such as a door lock or an encryption algorithm, achieves a security goal in a direct manner. A policy solution addresses a security problem by reducing the overall threat.

For example, the recording and motion picture industries have long been concerned about music and movie piracy, and they have long desired some technological means to prevent music or movies from being ripped, copied, and transmitted over the Internet. It is not likely that some technology will arise to meet this demand: information is inherently copiable and transmittable. However, in recent years the industry has offered an unprecedented amount of content legitimately over the Internet, selling music and movies online, and allowing easy access through multiple services. This policy reduces the threat of piracy, because there is little incentive for a user to pirate music if it is easy to obtain the music through normal means.

In our language of Harm, Threat and Vulnerability analysis, a technology solution will patch a vulnerability or prevent a threat from manifesting as harm; whereas a policy solution will reduce the threat level by reducing an adversary's incentive, or reducing their number. Just as we can reduce an electric current either by lowering voltage or introducing an impedance, we can reduce harm either through threat reduction policies or threat prevention technology. A system will often incorporate both.

It is important to understand that clients are not always welcoming of policy recommendations. A client may wish to continue a policy that causes the threat in the first place. For example, a movie studio may suffer from widespread piracy of a movie that is not yet released on home video. This threat can be reduced by releasing the movie on home video, but clearly the studio wishes to control its release date. What the client really wants is a technology to stop the piracy, not a recommendation that it do business differently. A client may also perceive a policy recommendation as a concession to the adversary, or even blaming the client instead of targeting the attacker. It is nevertheless often necessary to recommend a policy solution, because some problems can not be solved by a technological magic bullet.

Vertical integration

A successful security system must address multiple levels of abstraction, and guarantee "vertical" security from theory through to implementation. Suppose, for example, that we must devise a security system to allow anonymous financial transactions over the Internet. Our solution may include:

- Communications and cryptographic protocols that achieve the desired security policies;

  This may be the most cerebral aspect of our solution, the part that can be published in a paper and studied by other security experts for logical flaws.

- Cryptographic algorithms or primitives that implement the desired protocols;

  Protocols use primitives, and sometimes a primitive is a poor choice. This isn't just a matter of

an encryption algorithm or hash algorithm possibly being broken or weak; some algorithms are not supposed to be used in certain ways, or must be used in certain ways in order to achieve the desired security level.

- Secure software implementation of the aforementioned algorithms;

  Even if we have the right protocols and algorithms, what if the system is poorly coded? What if it is buggy, or if its implementation introduces an unexpected means of breaking the system?

- Proper software installation of the aforementioned software;

  Even if we have secure software, it may not run securely on a computer system. A computer may be infected or untrustworthy, and we must be able to demonstrate an appropriate installation in which the software achieves its goals.

- Training and enforcement in correct use of the system by end users;

  Ultimately it all goes to crap when the user writes a password on a post-it note, or chooses a birthday as a password. It is necessary for your solution to account for the human element, and explain how people will use the system securely, and know how to do so.

- Continual maintenance and administration of the system.

  If anything goes wrong, how is the system updated? There must be a plan for your solution to work over its expected lifetime.

We can't just solve a security problem by dashing off a brilliant protocol. The problem we face is that vulnerabilities can arise, and be exploited, in this entire stack from mathematical formula down to end user behavior. Security is often an AND relationship: everything must work from the program down to the foundations.

When vulnerabilities exist at multiple levels, our threat tree will show a broad range of different attacks to achieve the same goal, that target different layers from software to hardware to users.

Defense in depth

The phrase "defense in depth" is a common buzzword, and it essentially refers to redundant security measures. This introduces an OR relationship: that our security policy is enforced as long as at least one measure is effective.

If "defense in depth" is so great, why not just have it all the time? Because sometimes multiple security measures are not possible, they can be more costly, and if poorly implemented they may require additional hassle and work for users.

If we have defense in depth, it should force AND relationships in the threat tree, requiring an attacker to perform multiple attacks in order to achieve an end goal.