Schneier on Security

Attack Trees

B. Schneier

Dr. Dobb's Journal, December 1999.

Modeling security threats

By Bruce Schneier

Few people truly understand computer security, as illustrated by computer-security company marketing literature that touts "hacker proof software," "triple-DES security," and the like. In truth, unbreakable security is broken all the time, often in ways its designers never imagined. Seemingly strong cryptography gets broken, too. Attacks thought to be beyond the ability of mortal men become commonplace. And as newspapers report security bug after security bug, it becomes increasingly clear that the term "security" doesn't have meaning unless also you know things like "Secure from whom?" or "Secure for how long?"

Clearly, what we need is a way to model threats against computer systems. If we can understand all the different ways in which a system can be attacked, we can likely design countermeasures to thwart those attacks. And if we can understand who the attackers are -- not to mention their abilities, motivations, and goals -- maybe we can install the proper countermeasures to deal with the real threats.

Enter Attack Trees

Attack trees provide a formal, methodical way of describing the security of systems, based on varying attacks. Basically, you represent attacks against a system in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes.

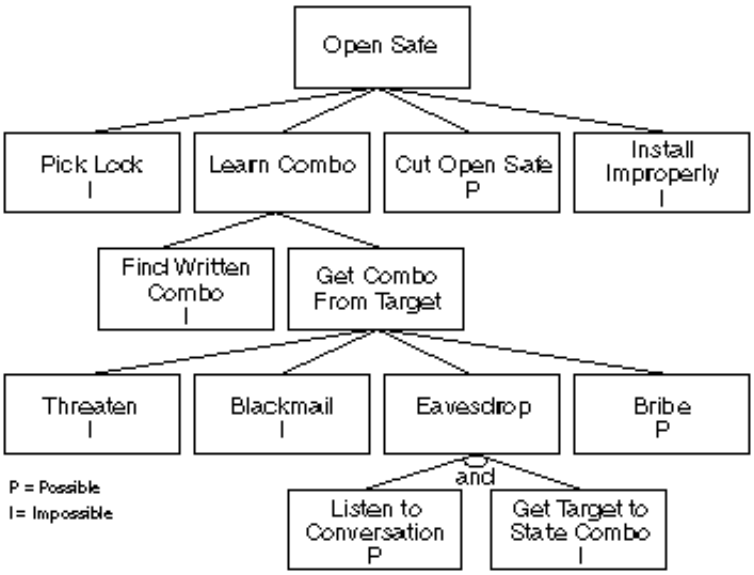

Figure 1: Attack Nodes

Figure 1, for instance, is a simple attack tree against a physical safe. The goal is opening the safe. To open the safe, attackers can pick the lock, learn the combination, cut open the safe, or install the safe improperly so that they can easily open it later. To learn the combination, they either have to find the combination written down or get the combination from the safe owner. And so on. Each node becomes a subgoal, and children of that node are ways to achieve that subgoal. (Of course, this is just a sample attack tree, and an incomplete one at that. How many other attacks can you think of that would achieve the goal?)

Note that there are AND nodes and OR nodes (in the figures, everything that isn't an AND node is an OR node). OR nodes are alternatives -- the four ways to open a safe, for example. AND nodes represent different steps toward achieving the same goal. To eavesdrop on someone saying the safe combination, attackers have to eavesdrop on the conversation AND get safe owners to say the combination. Attackers can't achieve the goal unless both subgoals are satisfied.

That's the basic attack tree. Once you have it completed, you can assign values -- *I* (impossible) and *P* (possible) in Figure 1 -- to the various leaf nodes, then make calculations about the nodes. (Again, this is only an illustrative example; do not take the values as an indication of how secure my safe really is.) Once you assign these values -- presumably this assignment will be the result of painstaking research on the safe itself -- you can calculate the security of the goal. The value of an OR node is possible if any of its children are possible, and impossible if all of its children are impossible. The value of an AND node is possible only if all children are possible, and impossible otherwise; see Figure 2.

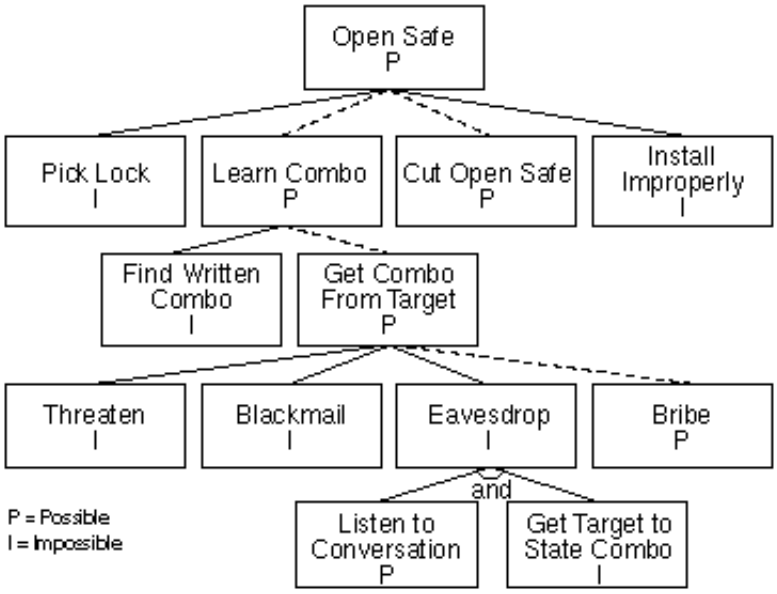

Figure 2: Possible Attacks

The dotted lines in Figure 2 show all possible attacks -- a hierarchy of possible nodes, from a leaf to the goal. In this sample system, there are two possible attacks: Cutting open the safe, or learning the combination by bribing the owner of the safe. With this knowledge, you know exactly how to defend this system against attack.

Assigning "possible" and "impossible" to the nodes is just one way to look at the tree. Any Boolean value can be assigned to the leaf nodes and then propagated up the tree structure in the same manner: easy versus difficult, expensive versus inexpensive, intrusive versus nonintrusive, legal versus illegal, special equipment required versus no special equipment. Figure 3 shows the same tree with another Boolean node value.

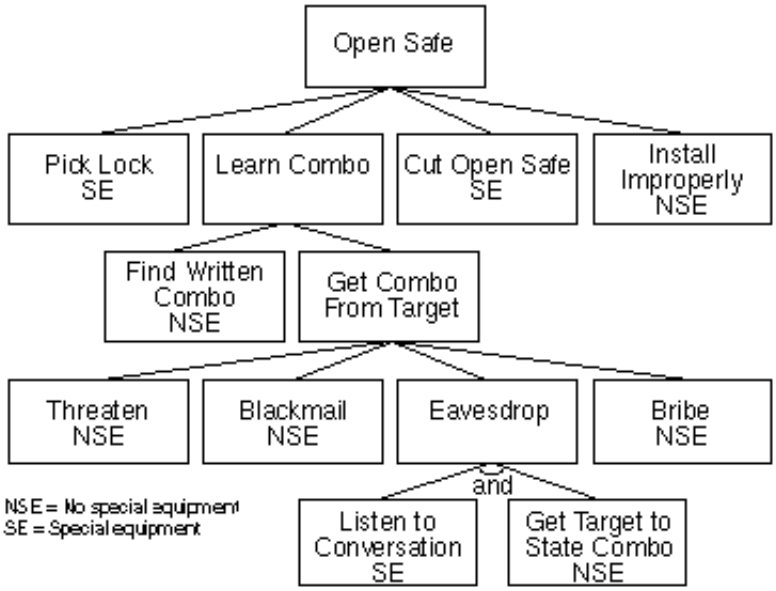

Figure 3: Special Equipment Required

Assigning "expensive" and "not expensive" to nodes is useful, but it would be better to show exactly how expensive. It is also possible to assign continuous values to nodes. Figure 4 shows the tree with different costs assigned to the leaf nodes. Like Boolean node values, these can propagate up the tree as well. OR nodes have the value of their cheapest child; AND nodes have the value of the sum of their children. In Figure 4, the costs have propagated up the tree, and the cheapest attack has been highlighted.

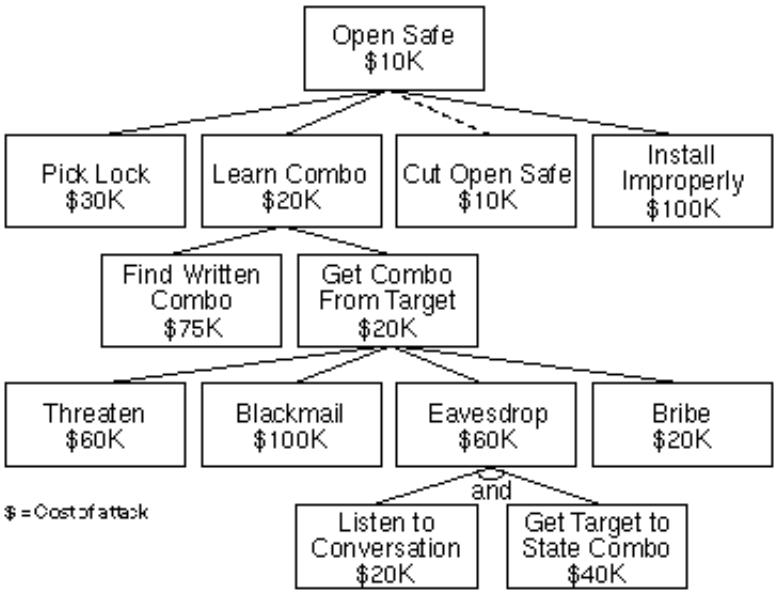

Figure 4: Cost of Attack

Again, this tree can be used to determine where a system is vulnerable. Figure 5 shows all attacks that cost less than $100,000. If you are only concerned with attacks that are less expensive (maybe the contents of the safe are only worth $100,000), then you should only concern yourself with those attacks.

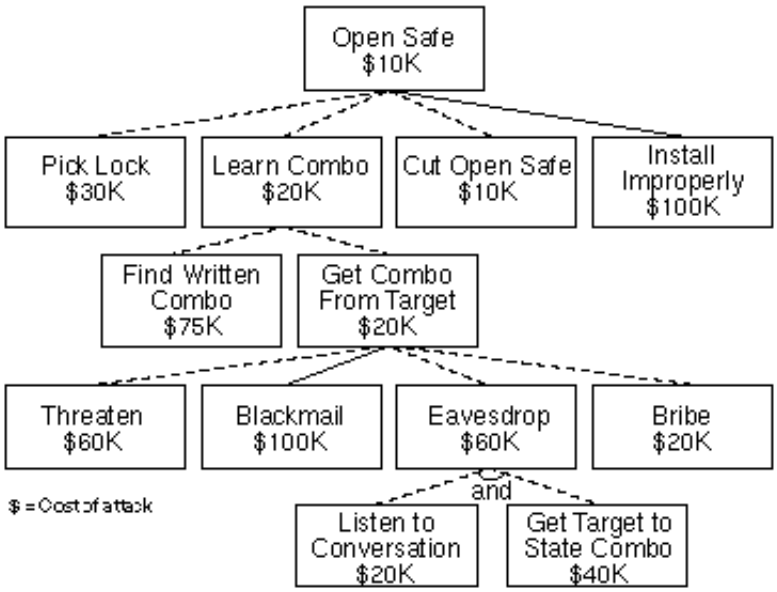

Figure 5: All Attacks Less than $100,000

There are many other possible continuous node values, including probability of success of a given attack, likelihood that an attacker will try a given attack, and so on.

Nodes and Their Values

In any real attack tree, nodes will have many different values corresponding to many different variables, both Boolean and continuous. Different node values can be combined to learn even more about a system's vulnerabilities. Figure 6, for instance, determines the cheapest attack requiring no special equipment. You can also find the cheapest low-risk attack, most likely nonintrusive attack, best low-skill attack, cheapest attack with the highest probability of success, most likely legal attack, and so on. Every time you query the attack tree about a certain characteristic of attack, you learn more about the system's security.

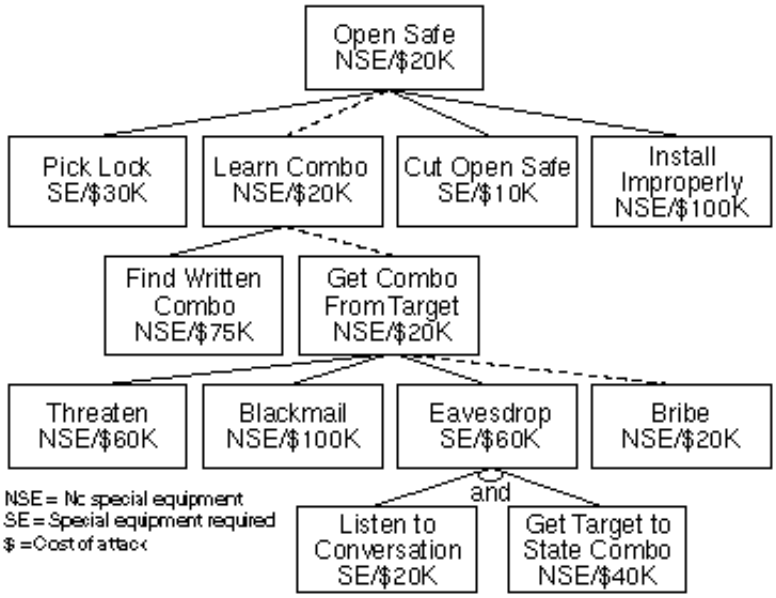

Figure 6: Cheapest Attack Requiring No Special Equipment

To make this work, you must marry attack trees with knowledge about attackers. Different attackers have different levels of skill, access, risk aversion, money, and so on. If you're worried about organized crime, you have to worry about expensive attacks and attackers who are willing to go to jail. If you are worried about terrorists, you also have to worry about attackers who are willing to die to achieve their goal. If you're worried about bored graduate students studying the security of your system, you usually don't have to worry about illegal attacks such as bribery and blackmail. The characteristics of your attacker determine which parts of the attack tree you have to worry about.

Attack trees also let you play "what if" games with potential countermeasures. In Figure 6, for example, the goal has a cost of $20,000. This is because the cheapest attack requiring no special equipment is bribing the person who knows the combination. What if you implemented a countermeasure -- paying that person more so that he is less susceptible to bribes? If you assume that the cost to bribe him is now $80,000 (again, this is an example; in the real world you'd be expected to research exactly how a countermeasure affects the node value), then the cost increases to $60,000 (presumably to hire the thugs to do the threatening).

A PGP Example

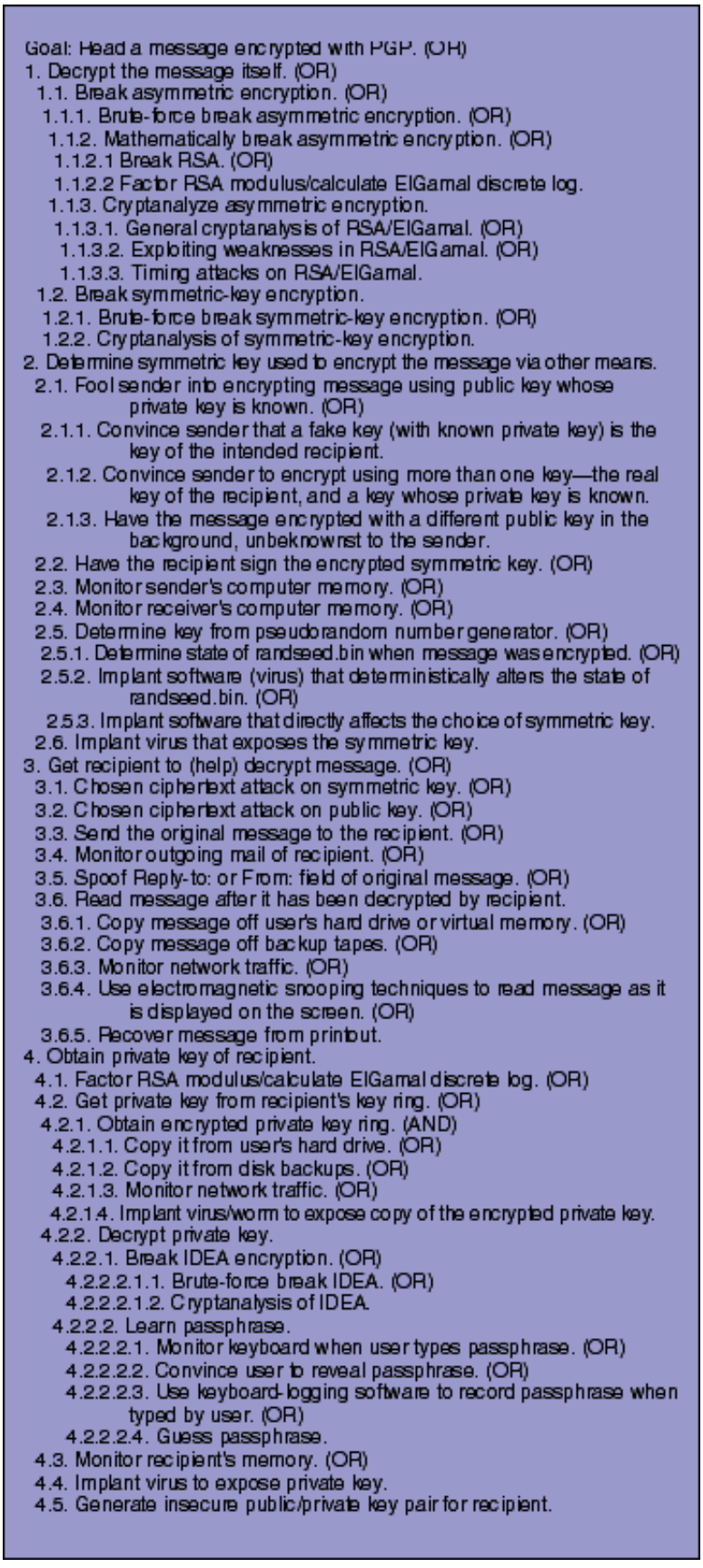

Figure 7: Attack Tree Against PGP

Figure 7 is an attack tree for the popular PGP e-mail security program. Since PGP is a complex program, this is a complex tree, and it's easier to write it in outline form than graphically. PGP has several security features, so this is only one of several attack trees for PGP. This particular attack tree has "read a message encrypted with PGP" as its goal. Other goals might be: "forge someone else's signature on a message," "change the signature on a message," "undetectibly modify a PGP-signed or PGP-encrypted message," and so on.

What immediately becomes apparent from the attack tree is that breaking the RSA or IDEA encryption algorithms are not the most profitable attacks against PGP. There are many ways to read someone's PGP-encrypted messages without breaking the cryptography. You can capture their screen when they decrypt and read the messages (using a Trojan horse like Back Orifice, a TEMPEST receiver, or a secret camera), grab their private key after they enter a passphrase (Back Orifice again, or a dedicated computer virus), recover their passphrase (a keyboard sniffer, TEMPEST receiver, or Back Orifice), or simply try to brute force their passphrase (I can assure you that it will have much less entropy than the 128-bit IDEA keys that it generates). In the scheme of things, the choice of algorithm and the key length is probably the least important thing that affects PGP's overall security. PGP not only has to be secure, but it has to be used in an environment that leverages that security without creating any new insecurities.

Creating Attack Trees

How do you create an attack tree like this? First, you identify the possible attack goals. Each goal forms a separate tree, although they might share subtrees and nodes. Then, try to think of all attacks against each goal. Add them to the tree. Repeat this process down the tree until you are done. Give the tree to someone else, and have him think about the process and add any nodes he thinks of. Repeat as necessary, possibly over the course of several months. Of course there's always the chance that you forgot about an attack, but you'll get better with time. Like any security analysis, creating attack trees requires a certain mindset and takes practice.

Once you have the attack tree, and have researched all the node values (these values will change over time, both as attacks become easier and as you get more exact information on the values), you can use the attack tree to make security decisions. You can look at the values of the root node to see if the system's goal is vulnerable to attack. You can determine if the system is vulnerable to a particular kind of attack; password guessing, for instance. You can use the attack tree to list the security assumptions of a system; for example, the security of PGP might assume that no one could successfully bribe the programmers. You can determine the impact of a system modification or a new vulnerability discovery: Recalculate the nodes based on the new information and see how the goal node is affected. And you can compare and rank attacks -- which is cheaper, which is more likely to succeed, and the like.

One of the surprising things that comes out of this kind of analysis is that the areas people think of as vulnerable usually aren't. With PGP, for example, people generally worry about key length. Should they use 1024-bit RSA or 2048-bit RSA? Looking at the attack tree, though, shows that the key length of RSA doesn't really matter. There are all sorts of other attacks -- installing a keyboard sniffer, modifying the program on the victim's hard drive -- that are much easier than breaking the RSA public key. Increasing the key length from 1024 bits to 2048 bits is like putting an enormous stake into the ground and hoping the enemy runs right into it, as opposed to building a lower palisade around the target. Attack trees give you perspective on the whole system.

One of the things that really makes attack trees valuable is that they capture knowledge in a reusable form. Once you've completed the PGP attack tree, you can use it in any situation that uses PGP. The attack tree against PGP becomes part of a larger attack tree. For example, Figure 8 shows an attack tree whose goal is to read a specific message that has been sent from one Windows 98 computer to another. If you look at the root nodes of the tree, the entire attack trees for PGP and for opening a safe fit into this attack tree.

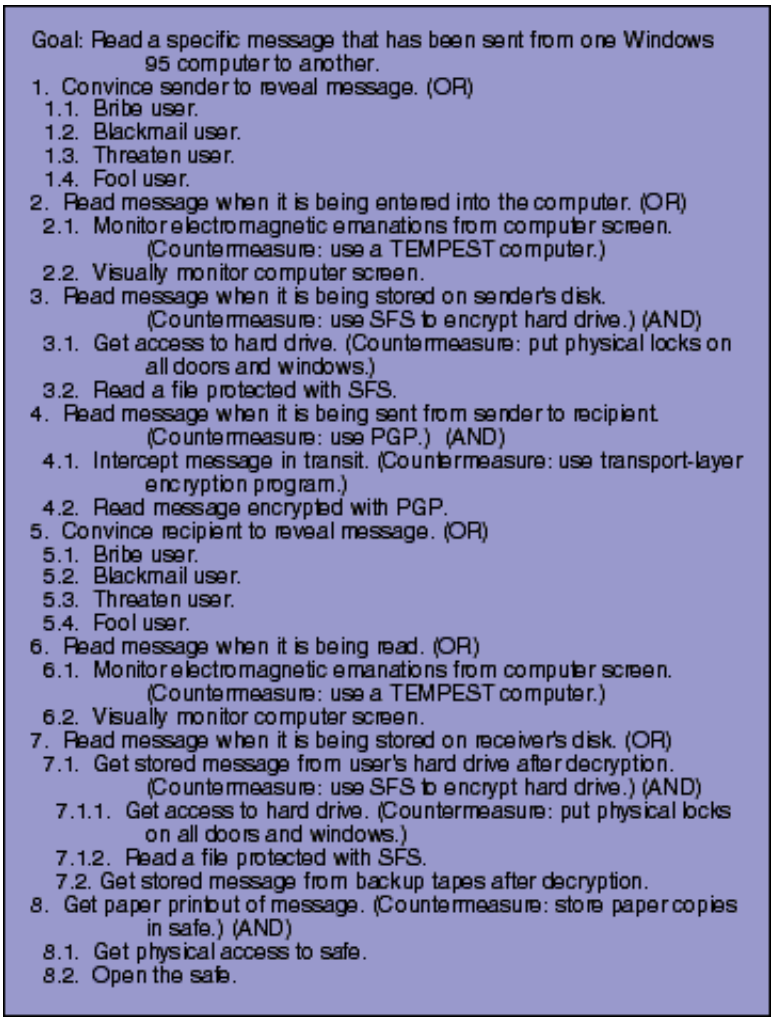

Figure 8: Attack Tree Against a General Computer System

This scalability means that you don't have to be an expert in everything. If you're using PGP in a system, you don't have to know the details of the PGP attack tree; all you need to know are the values of the root node. If you're a computer-security expert, you don't have to know the details about how difficult a particular model of safe is to crack; you just need to know the values of the root node. Once you build up a library of attack trees against particular computer programs, door and window locks, network security protocols, or whatever, you can reuse them whenever you need to. For a national security agency concerned about compartmentalizing attack expertise, this kind of system is very useful.

Conclusion

Attack trees provide a formal methodology for analyzing the security of systems and subsystems. They provide a way to think about security, to capture and reuse expertise about security, and to respond to changes in security. Security is not a product -- it's a process. Attack trees form the basis of understanding that process.

Categories: Miscellaneous Papers

← Security in the Real World: How to Evaluate Security Technology

Yarrow-160 →