CHAPTER

# 4

# Attack Trees

As Bruce Schneier wrote in his introduction to the subject, "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" (Schneier, 1999).

In this chapter you'll learn about the attack tree building block as an alternative to STRIDE. You can use attack trees as a way to find threats, as a way to organize threats found with other building blocks, or both. You'll start with how to use an attack tree that's provided to you, and from there learn various ways you can create trees. You'll also examine several example and real attack trees and see how they fit into finding threats. The chapter closes with some additional perspective on attack trees.

## Working with Attack Trees

Attack trees work well as a building block for threat enumeration in the four-step framework. They have been presented as a full approach to threat modeling (Salter, 1998), but the threat modeling community has learned a lot since then.

There are three ways you can use attack trees to enumerate threats: You can use an attack tree someone else created to help you find threats. You can create

a tree to help you think through threats for a project you're working on. Or you can create trees with the intent that others will use them. Creating new trees for general use is challenging, even for security experts.

## Using Attack Trees to Find Threats

If you have an attack tree that is relevant to the system you're building, you can use it to find threats. Once you've modeled your system with a DFD or other diagram, you use an attack tree to analyze it. The attack elicitation task is to iterate over each node in the tree and consider if that issue (or a variant of that issue) impacts your system. You might choose to track either the threats that apply or each interaction. If your system or trees are complex, or if process documentation is important, each interaction may be helpful, but otherwise that tracking may be distracting or tedious. You can use the attack trees in this chapter or in Appendix B "Threat Trees" for this purpose.

If there's no tree that applies to your system, you can either create one, or use a different threat enumeration building block.

## Creating New Attack Trees

If there are no attack trees that you can use for your system, you can create a project-specific tree. A project-specific tree is a way to organize your thinking about threats. You may end up with one or more trees, but this section assumes you're putting everything in one tree. The same approach enables you to create trees for a single project or trees for general use.

The basic steps to create an attack tree are as follows:

1. Decide on a representation.
2. Create a root node.
3. Create subnodes.
4. Consider completeness.
5. Prune the tree.
6. Check the presentation.

### *Decide on a Representation*

There are AND trees, where the state of a node depends on all of the nodes below it being true, and OR trees, where a node is true if any of its subnodes are true. You need to decide, will your tree be an AND or an OR tree? (Most will be OR trees.) Your tree can be created or presented graphically or as an outline. See the section "Representing a Tree" later in this chapter for more on the various forms of representation.

### *Create a Root Node*

To create an attack tree, start with a root node. The root node can be the component that prompts the analysis, or an adversary's goal. Some attack trees use the problematic state (rather than the goal) as the root. Which you should use is a matter of preference. If the root node is a component, the subnodes should be labeled with what can go wrong for the node. If the root node is an attacker goal, consider ways to achieve that goal. Each alternative way to achieve the goal should be drawn in as a subnode.

The guidance in "Toward a Secure System Engineering Methodology" (Salter, 1999) is helpful to security experts; however, it doesn't shed much light on how to actually generate the trees, comparative advice about what a root node should be (in other words, whether it's a goal or a system component and, most important, when one is better than the other), or how to evaluate trees in a structured fashion that would be suitable for those who are not security experts. To be prescriptive:

- Create a root node with an attacker goal or high-impact action.
- Use OR trees.
- Draw them into a grid that the eye can track linearly.

### *Create Subnodes*

You can create subnodes by brainstorming, or you can look for a structured way to find more nodes. The relation between your nodes can be AND or OR, and you'll have to make a choice and communicate it to those who are using your tree. Some possible structures for first-level subnodes include:

- Attacking a system:
  - physical access
  - subvert software
  - subvert a person
- Attacking a system via:
  - People
  - Process
  - Technology
- Attacking a product during:
  - Design
  - Production
  - Distribution
  - Usage
  - Discard

You can use these as a starting point, and make them more specific to your system. Iterate on the trees, adding subnodes as appropriate.

**NOTE** **Here the term subnode is used to include leaf (end) nodes and nodes with children, because as you create something you may not always know whether it is a leaf or whether it has more branches.**

### *Consider Completeness*

For this step, you want to determine whether your set of attack trees is complete enough. For example, if you are using components, you might need to add additional trees for additional components. You can also look at each node and ask "is there another way that could happen?" If you're using attacker motivations, consider additional attackers or motivations. The lists of attackers in Appendix C "Attacker Lists" can be used as a basis.

An attack tree can be checked for quality by iterating over the nodes, looking for additional ways to reach the goal. It may be helpful to use STRIDE, one of the attack libraries in the next chapter, or a literature review to help you check the quality.

### *Prune the Tree*

In this step, go through each node in the tree and consider whether the action in each subnode is prevented or duplicative. (An attack that's worth putting in a tree will generally only be prevented in the context of a project.) If an attack is prevented, by some mitigation you can mark those nodes to indicate that they don't need to be analyzed. (For example, you can use the test case ID, an "I" for impossible, put a slash through the node, or shade it gray.) Marking the nodes (rather than deleting them) helps people see that the attacks were considered. You might choose to test the assumption that a given node is impossible. See the "Test Process Integration" section in Chapter 10 "Validating That Threats Are Addressed" for more details.

### *Check the Presentation*

Regardless of graphical form, you should aim to present each tree or subtree in no more than a page. If your tree is hard to see on a page, it may be helpful to break it into smaller trees. Each top level subnode can be the root of a new tree, with a "context" tree that shows the overall relations. You may also be able to adjust presentation details such as font size, within the constraints of usability.

The node labels should be of the same form, focusing on active terms. Finally, draw the tree on a grid to make it easy to track. Ideally, the equivalent level subnodes will show on a single line. That becomes more challenging as you go deeper into a tree.

## Representing a Tree

Trees can be represented in two ways: as a free-form (human-viewable) model without any technical structure, or as a structured representation with variable types and/or metadata to facilitate programmatic analysis.

### Human-Viewable Representations

Attack trees can be drawn graphically or shown in outline form. Graphical representations are a bit more work to create but have more potential to focus attention. In either case, if your nodes are not all related by the same logic (AND/OR), you'll need to decide on a way to represent the relationship and communicate that decision. If your tree is being shown graphically, you'll also want to decide if you use a distinct shape for a terminal node: The labels in a node should be carefully chosen to be rich in information, especially if you're using a graphical tree. Words such as "attack" or "via" can distract from the key information. Choose "modify file" over "attack via modifying file." Words such as "weak" are more helpful when other nodes say "no." So "weak cryptography" is a good contrast to "no cryptography."

As always, care should be taken to ensure that the graphics are actually information-rich and communicative. For instance, consider the three representations of a tree shown in Figure 4–1.

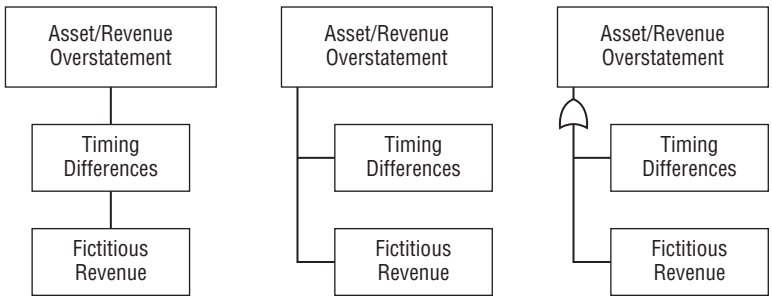


**Figure 4–1:** Three representations of a tree

The left tree shows an example of a real tree that simply uses boxes. This representation does not clearly distinguish hierarchy, making it hard to tell which nodes are at the same level of the tree. Compare that to the center tree, which uses a tree to show the equivalence of the leaf nodes. The rightmost tree adds the "OR gate" symbol from circuit design to show that any of the leaf nodes lead to the parent condition.

Additionally, tree layout should make considered use of space. In the very small tree in Figure 4–2, note the pleasant grid that helps your eye follow the layout. In contrast, consider the layout of Figure 4–3, which feels jumbled. To focus your attention on the layout, both are shown too small to read.

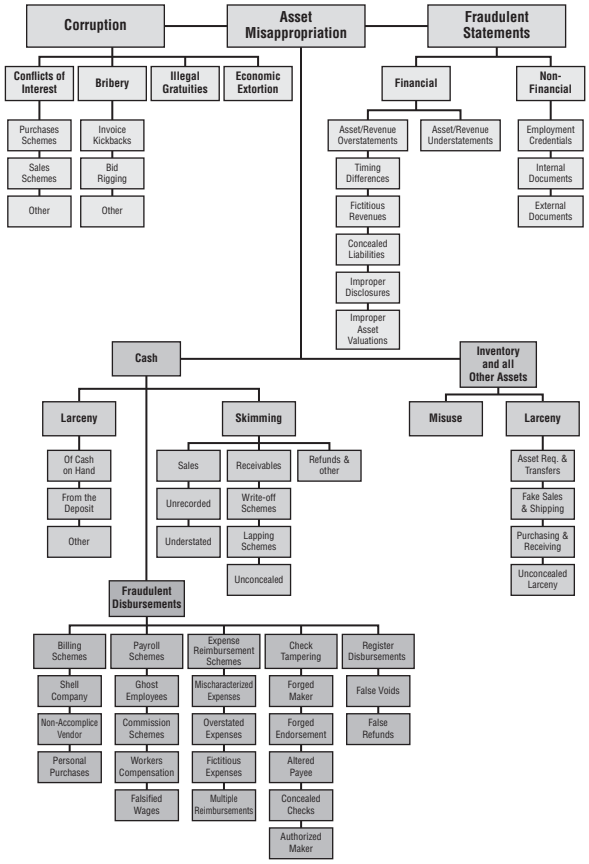


**Figure 4–2:** A tree drawn on a grid

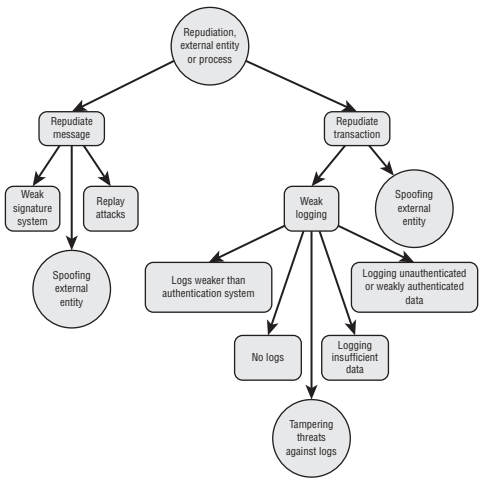


**Figure 4–3:** A tree drawn without a grid

**NOTE** **In *Writing Secure Code 2* (Microsoft Press, 2003), Michael Howard and David LeBlanc suggest the use of dotted lines for unlikely threats, solid lines for likely threats, and circles to show mitigations, although including mitigations may make the trees too complex.**

Outline representations are easier to create than graphical representations, but they tend to be less attention-grabbing. Ideally, an outline tree is shown on a single page, not crossing pages. The question of how to effectively represent AND/OR is not simple. Some representations leave them out, others include an indicator either before or after a line. The next three samples are modeled after the trees in "Election Operations Assessment Threat Trees" later in this chapter. As you look at them, ask yourself precisely what is needed to achieve the goal in node 1, "Attack voting equipment."

1. Attack voting equipment
   - 1.1 Gather knowledge
     - 1.1.1 From insider
     - 1.1.2 From components
   - 1.2 Gain insider access
     - 1.2.1 At voting system vendor
     - 1.2.2 By illegal insider entry

The preceding excerpt isn't clear. Should the outline be read as a need to do each of these steps, or one or the other to achieve the goal of attacking voting equipment? Contrast that with the next tree, which is somewhat better:

1. Attack voting equipment
   - 1.1 Gather knowledge (and)
     - 1.1.1 From insider (or)
     - 1.1.2 From components
   - 1.2 Gain insider access (and)
     - 1.2.1 At voting system vendor (or)
     - 1.2.2 By illegal insider entry

This representation is useful at the end nodes: It is clearly 1.1.1 or 1.1.2. But what does the "and" on line 1.1 refer to? 1.1.1 or 1.1.2? The representation is not clear. Another possible form is shown next:

1. Attack voting equipment
   O 1.1 Gather knowledge
      T 1.1.1 From insider
      O 1.1.2 From components
   O 1.2 Gain insider access
      T 1.2.1 At voting system vendor
      T 1.2.2 By illegal insider entry

This is intended to be read as "AND Node: 1: Attack voting equipment, involves 1.1, gather knowledge either from insider or from components AND 1.2, gain insider access . . ." This can be confusing if read as the children of that node are to be ORed, rather than being ORed with its sibling nodes. This is much clearer in the graphical presentation. Also note that the steps are intended to be sequential. You must gather knowledge, then gain insider access, then attack the components to pull off the attack.

As you can see from the preceding examples, the question of how to use an outline representation of a tree is less simple than you might expect. If you are using someone else's tree, be sure you understand their intent. If you are creating a tree, be sure you are clear on your intent, and clear in your communication of your intent.

### Structured Representations

Graphical and outline presentation of trees are useful for humans, but a tree is also a data structure, and a structured representation of a tree makes it possible to apply logic to the tree and in turn, the system you're modeling. Several software packages enable you to create and manage complex trees. One such package allows the modeler to add costs to each node, and then assess what attacks an attacker with a given budget can execute. As your trees become more complex, such software is more likely to be worthwhile. See Chapter 11 "Threat Modeling Tools" for a list of tree management software.

## Example Attack Tree

The following simple example of an attack tree (and a useful component for other attack tree activity) models how an attacker might get into a building. The entire tree is an OR tree; any of the methods listed will achieve the goal. (This tree is derived from "An Attack Tree for the Border Gateway Protocol" [Convery, 2004].)

**Goal: Access to the building**

1. Go through a door
   a. When it's unlocked:
      i. Get lucky.
      ii. Obstruct the latch plate (the "Watergate Classic").
      iii. Distract the person who locks the door at night.
   b. Drill the lock.
   c. Pick the lock.
   d. Use the key.
      i. Find a key.
      ii. Steal a key.
      iii. Photograph and reproduce the key.
      iv. Social engineer a key from someone.
         1. Borrow the key.
         2. Convince someone to post a photo of their key ring.
   e. Social engineer your way in.
      i. Act like you're authorized and follow someone in.
      ii. Make friends with an authorized person.
      iii. Carry a box, a cup of coffee in each hand, etc.
2. Go through a window.
   a. Break a window.
   b. Lift the window.
3. Go through a wall.
   a. Use a sledgehammer or axe.
   b. Use a truck to go through the wall.
4. Gain access via other means.
   a. Use a fire escape.
   b. Use roof access from a helicopter (preferably black) or adjacent building.
   c. Enter another part of the building, using another tenant's access.

## Real Attack Trees

A variety of real attack trees have been published. These trees may be helpful to you either directly, because they model systems like the one you're modeling, or as examples of how to build an attack tree. The three attack trees in this section show how insiders commit financial fraud, how to attack elections, and threats against SSL.

Each of these trees has the nice property of being available now, either as an extended example, as a model for you to build from, or (if you're working around fraud, elections, or SSL), to use directly in analyzing a system which matters to you.

The fraud tree is designed for you to use. In contrast, the election trees were developed to help the team think through their threats and organize the possibilities.

### Fraud Attack Tree

An attack tree from the Association of Certified Fraud Examiners is shown with their gracious permission in Figure 4–4, and it has a number of good qualities. First, it's derived from actual experience in finding and exposing fraud. Second, it has a structure put together by subject matter experts, so it's not a random collection of threats. Finally, it has an associated set of mitigations, which are discussed at great length in Joseph Wells' *Corporate Fraud Handbook* (Wiley, 2011).

### Election Operations Assessment Threat Trees

The largest publicly accessible set of threat trees was created for the Elections Assistance Commission by a team centered at the University of Southern Alabama. There are six high-level trees. They are useful both as an example and for you to use directly, and there are some process lessons you can learn.

**NOTE** **This model covers a wider scope of attacks than typical for software threat models, but is scoped like many operational threat models.**

1. Attack voting equipment.
2. Perform an insider attack.
3. Subvert the voting process.
4. Experience technical failure.
5. Attack audit.
6. Disrupt operations.

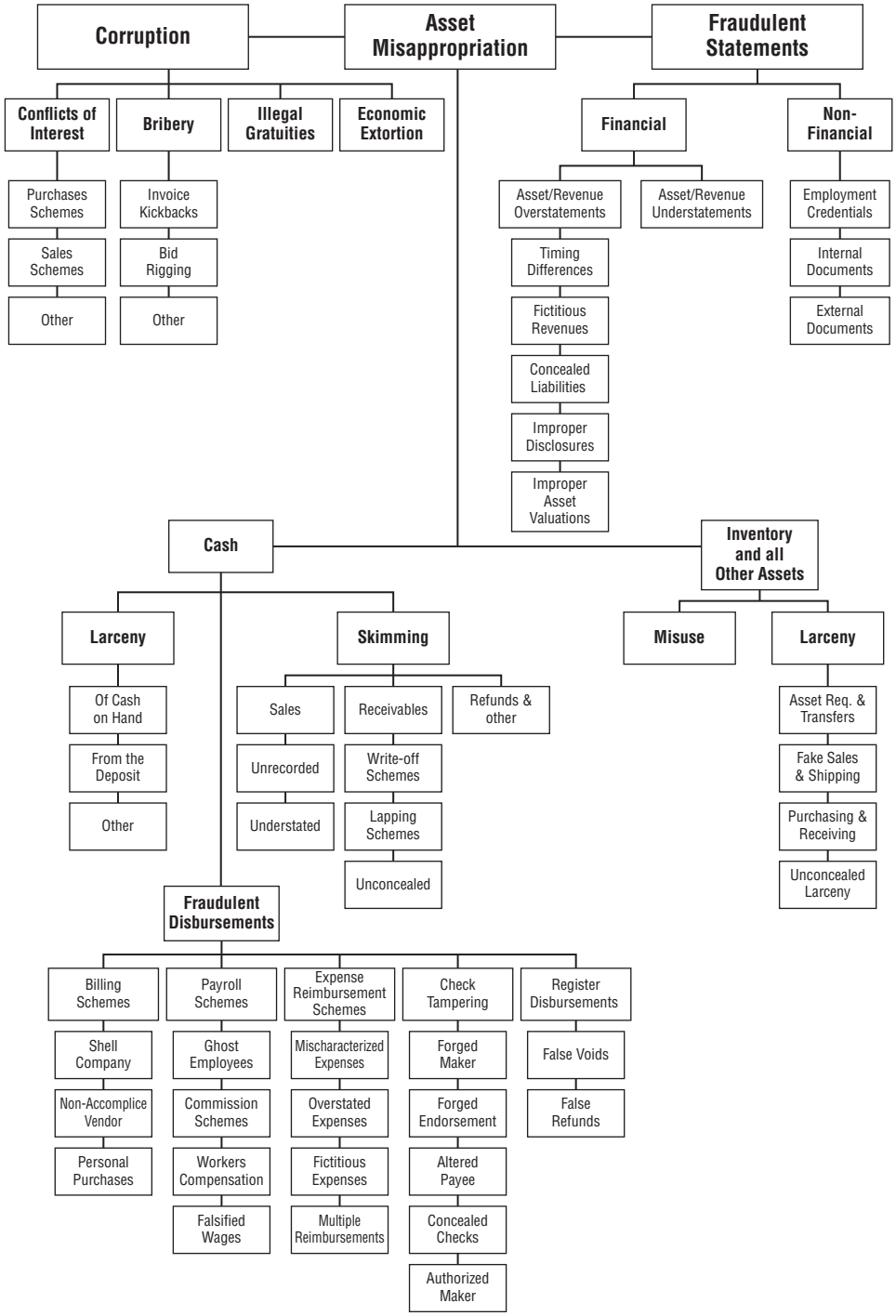


**Figure 4–4:** The ACFE fraud tree

If your system is vulnerable to threats such as equipment attack, insider attack, process subversion or disruption, these attack trees may work well to help you find threats against those systems.

The team created these trees to organize their thinking around what might go wrong. They described their process as having found a very large set of issues via literature review, brainstorming, and original research. They then broke the threats into high-level sets, and had individuals organize them into trees. An attempt to sort the sets into a tree in a facilitated group process did not work (Yanisac, 2012). The organization of trees may require a single person or a very close-knit team; you should be cautious about trying for consensus trees.

### Mind Maps

Application security specialist Ivan Ristic (Ristić, 2009) conducted an interesting experiment using a mind map for what he calls an SSL threat model, as shown in Figure 4–5.

This is an interesting way to present a tree. There are very few mind-map trees out there. This tree, like the election trees, shows a set of editorial decisions and those who use mind maps may find the following perspective on this mind map helpful:

- The distinction between "Protocols/Implementation bugs" and "End points/Client side/secure implementation" is unclear.
- There's "End points/Client side/secure implementation" but no "server side" counterpart to that.
- Under "End points/server side/server config" there's a large subtree. Compare that to Client side where there's no subtree at all.
- Some items have an asterisk (*) but it's unclear what that means. After discussion with Ivan, it turns out that those "may not apply to everyone."
- There's an entire set of traffic analytic threats that allow you to see where on a secure site someone is. These issues are made worse by AJAX, but more important here, how should they fit into this mind map? Perhaps under "Protocols/specifications/scope limits"?
- It's hard to find elements of the map, as it draws the eye in various directions, some of which don't align with the direction of reading.

## Perspective on Attack Trees

Attack trees can be a useful way to convey information about threats. They can be helpful even to security experts as a way to quickly consider possible attack types. However, despite their surface appeal, it is very hard to create attack trees.

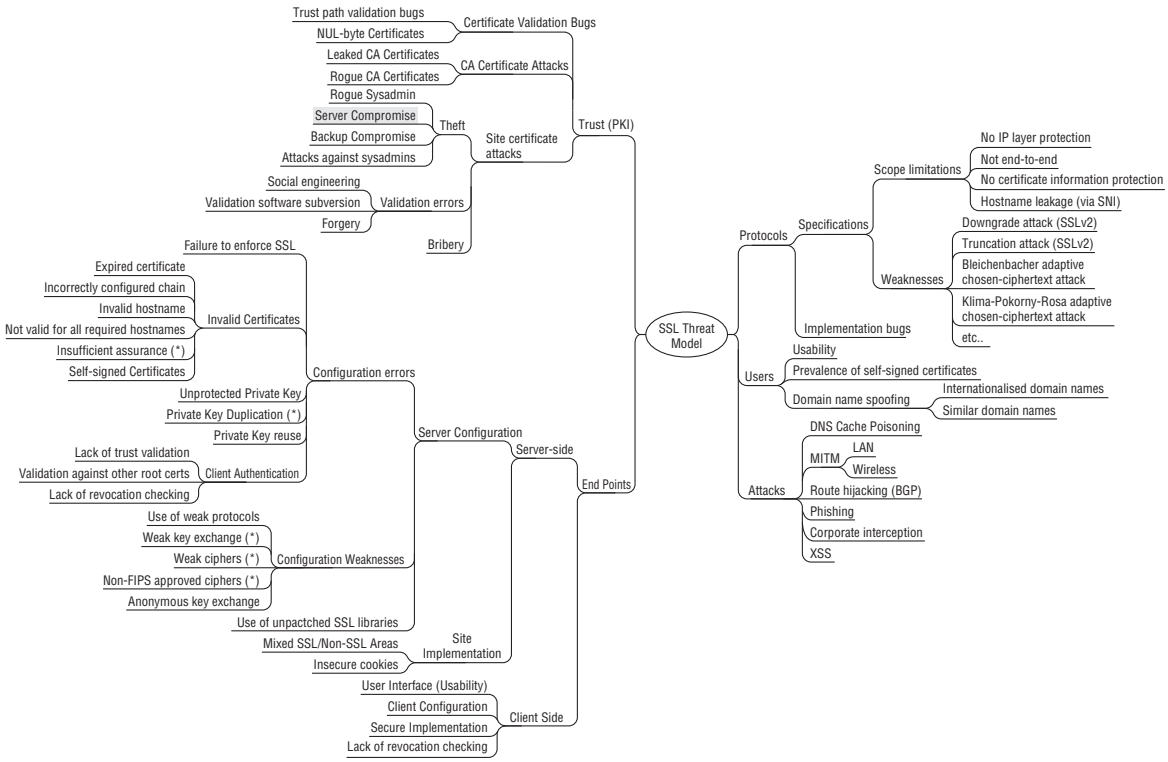


**Figure 4–5:** Ristic's SSL mind map

I hope that we'll see experimentation and perhaps progress in the quality of advice. There are also a set of issues that can make trees hard to use, including completeness, scoping, and meaning:

- **Completeness:** Without the right set of root nodes, you could miss entire attack groupings. For example, if your threat model for a safe doesn't include "pour liquid nitrogen on the metal, then hit with a hammer," then your safe is unlikely to resist this attack. Drawing a tree encourages specific questions, such as "how could I open the safe without the combination?" It may or may not bring you to the specific threat. Because there's no way of knowing how many nodes a branch should have, you may never reach that point. A close variant of the this is how do you know that you're done? (Schneier's attack tree article alludes to these problems.)
- **Scoping:** It may be unreasonable to consider what happens when the computer's main memory is rapidly cooled and removed from the motherboard. If you write commercial software for word processing, this may seem like an operating system issue. If you create commercial operating systems, it may seem like a hardware issue. The nature of attack trees means many of the issues discovered will fall under the category of "there's no way for us to fix that."
- **Meaning:** There is no consistency around AND/OR, or around sequence, which means that understanding a new tree takes longer.

## Summary

Attack trees fit well into the four-step framework for threat modeling. They can be a useful tool for finding threats, or a way to organize thinking about threats (either for your project or more broadly).

To create a new attack tree to help you organize thinking, you need to decide on a representation, and then select a root node. With that root node, you can brainstorm, use STRIDE, or use a literature review to find threats to add to nodes. As you iterate over the nodes, consider if the tree is complete or overly-full, aiming to ensure the right threats are in the tree. When you're happy with the content of the tree, you should check the presentation so others can use it. Attack trees can be represented as graphical trees, as outlines, or in software.

You saw a sample tree for breaking into a building, and real trees for fraud, elections, and SSL. Each can be used as presented, or as an example for you to consider how to construct trees of your own.