COMS30038 — SECURITY BEHAVIOURS

THREAT MODELLING: KILLCHAINS & ATTACK TREES

Matthew Edwards[*]

---

1 THE CYBER-KILLCHAIN

The concept of a 'kill chain' comes from the military, where it is a model for identifying the steps involved in carrying out an attack. For example, the United States Air Force operates a six-step killchain for dealing with time-sensitive threats which is known by the charming acronym of F2T2EA [1]. The steps in their model are, in order: *Find*ing the target, *Fix*ing the target's location, continuing to *Track* the target, *Target*ing it with an attack method, before then moving to *Engage* in action against the target, followed by an *Assess*ment of whether the desired results[1] were achieved. The F2T2EA model is to some degree normative – it describes things that members of the USAF *should* do when carrying out an attack on time-sensitive targets. It may not be completely faithful as a *description* of the realities of engagement in combat, but it stands as a checklist to guide Air Force personnel's behaviour.

In the context of computer security, the killchain has been adopted as a useful method for understanding attacks from the perspective of the adversary – that is, to plan an attack as if we were the cybercriminals [2]. Just as with F2T2EA, the cyber-killchain describes a sequence of actions to be taken by an attacker, with each phase leading to the next. The difference is that the cyber-killchain is used primarily to identify possible interventions that would "break the chain" and prevent an attack from being carried out.

In their paper introducing the cyber-killchain, Hutchins et al. [2] suggested six categories of countermeasure that might be deployed at each stage of the chain – these categories themselves also being drawn from the United States Department of Defence's doctrine on information operations. This model suggests that for every step in the cyber-killchain, defenders of systems should be alert for opportunities to *detect*, *deny*, *disrupt*, *degrade*, *deceive*, or *destroy*[2] an attack against their organisation.

[*]matthew.john.edwards@bristol.ac.uk

[1] e.g., total destruction.

[2] This option is usually omitted. It comes from the original DoD doctrine, but Hutchins et al. didn't find an application of it for their cyber-killchain, and nor have most other authors.

There are seven stages to the cyber-killchain model[3]:

Reconnaissance is the phase in which attackers identify possible targets. This might involve, for example, scanning a network for servers with known vulnerabilities, or researching a company's employees online, looking for details that could be useful in carrying out an attack.

Weaponisation is the phase where the attacker creates the 'attack payload' they are going to use. Typically this refers to a software exploit, but it could equally apply to writing a pseudoscript or false profile for use in a social engineering attack.

Delivery is the phase in which the attacker transmits the payload to the victim. This might mean launching a server, placing an advert, or sending an email.

Exploitation is when the payload exploits the target's vulnerability, and the work done in the weaponisation phase pays off by obtaining the intended access or information.

Installation typically refers to the installation of a new longer-term "backdoor" access method for the attacker to make use of; giving them the ability to make use of the machine. This often involves some form of *privilege escalation*.

Command and Control is when the attacker connects a targeted machine to their wider infrastructure – typically this is seen in the construction of a botnet.

Actions on Objectives is the term for the phase in which the attacker benefits—usually monetarily—from the control established over the target system. This might involve theft of sensitive information, extortion (e.g., through ransomware) or use of the machine as an asset to mine cryptocurrencies or deliver spam.

For each stage, and for each method criminals are known to deploy in that stage, security professionals should be looking for ways to implement a countermeasure that interrupts the killchain. For example, you might *deny* attackers access to sensitive information by securely encrypting it. This prevents at least that particular 'action on objective'. Generally speaking, the earlier in the chain a countermeasure comes, the more harm it may be able to prevent. An attacker prevented from accessing your sensitive information at the end of the chain still has access to your machine, and might redirect their efforts towards, e.g., mining cryptocurrencies. If instead an antivirus product were to *detect* an exploit attached to an inbound email, the chain could be broken

[3]There are actually a number of competing cyber-killchain models, with some positing as many as 18 different phases to cyber-attacks [3]. The one we'll use in this course is the one proposed by Hutchins et al., which is still widely recognised.

much earlier, before the attacker has managed to gain any access. Naturally, it would be even better if they were unable to even identify any possible targets in your organisation.

Of the categories of countermeasure, *detect* and *deny* are the most straightforward and common. In the former case, you have a system that can identify that the appropriate phase of an attack is happening, and alert someone to take action. In the latter, you have some countermeasure in place that completely prevents this phase of the attack from being carried out, like a patch for a known software exploit. Of the remaining three, *disrupt* refers to countermeasures like an automated network intrusion prevention system, which might block malicious activity a certain amount of the time, but cannot be considered to completely *deny* that form of attack from happening—its defences are probabilistic, or based on certain known patterns, and they could be evaded by a skilled attacker. To *degrade* an attack usually means deploying countermeasures that slow down or otherwise inhibit your attacker. For example, you could prevent large amounts of data being quickly exfiltrated from your systems by throttling the network, or give new processes such limited resources that they are of no use to an attacker for mining cryptocurrencies. Finally, a cunning defender may *deceive* their adversaries by feeding them false information or letting them into a 'honeypot' system[4]. Deception can be leveraged to waste the time of attackers, to gather evidence for prosecution, or simply to learn what the attacker *would have done* if they had obtained access to your real system.

The aim of using the killchain is to identify opportunities to prevent an attack from being carried out. It naturally lends itself to considerations of *layered defence*, in which countermeasures can be deployed both early and late in the killchain, with the later countermeasures being there to handle threats that slip through disruptive countermeasures or arrive by unpredicted means. However, there are limitations to the model. For one, it is very situational, focusing on the series of events leading up to an attack, but without giving much consideration to what comes before[5] or after[6]. The model also bakes-in certain assumptions about the methods and objectives of the attacker, such as that the attack is primarily technical in nature, or aims at continued control of a target machine. These assumptions, while in many cases justified, can make for an awkward fit when applying the model to understand other forms of cyber-attack, in which motives and methods may be rather more diverse.

[4] A system you set up to look like a vulnerable target containing something valuable, but which actually contains no real value for the attacker, and is closely monitored to record their activities.

[5] e.g., how did this threat actor become who they are?

[6] e.g., what can we do to stop them trying this again?

2 ATTACK TREES

A somewhat more flexible model for understanding attacks is the *attack tree*. Unlike the cyber-killchain, attack-tree modelling does not describe fixed stages that attacks go through. In fact, attack-tree modelling is intentionally highly assumption-free, to the extent that it does not even assume that attackers have any particular goal in mind, or that you are defending any particular sort of asset. This flexibility makes the attack tree a powerful model, equally suited to identifying vulnerabilities in software like PGP [4], protecting entire SCADA systems [5] or even identifying human attack vectors from within organisations [6].

Attack trees represent attacks and countermeasures using a tree structure, as the name suggests. The root node of an attack tree is the goal of the attacker[7]. The child nodes of any node in the tree are means by which the parent node may be achieved. As you traverse down the tree, each node becomes a sub-goal that it is necessary for the attacker to achieve in order to reach their final goal, the root node. The leaf nodes can be considered the necessary requirements for obtaining the final result.

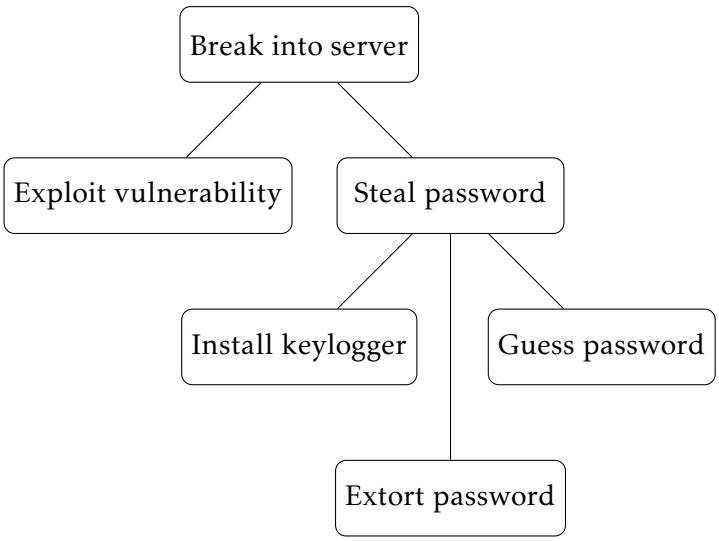

Figure 1: A simple attack tree example

Consider the example given in Figure 1. The ultimate goal of the attacker is to break into a server[8]. We start by considering how they might do this, given what we know about this fictional server and its authentication mechanisms. There are (at least) two possible paths for the attackers to achieve their goal: they could exploit a software vulnerability and bypass the authentication mechanisms, or they could get a password in order to log in as an authenticated

[7] In most cases there will be multiple root nodes, representing different goals that attackers may have, and correspondingly multiple trees.

[8] This of course is unlikely to be be real final goal of an attacker – this would be a sub-tree in a more complex tree, but it serves as a toy example.

user. Focusing on the second option, we now start to consider how the attacker might steal a password. One option could involve getting a keylogger onto the personal device of an authenticated user. The attacker could also blackmail an authenticated user into handing over their credentials. Another method might be to simply guess the password. Any of these methods would fulfill the 'steal password' subgoal, which in turn would allow the attacker to achieve the root goal. The child nodes are here being considered to have an 'or' relationship – any of them would serve to achieve the parent node.

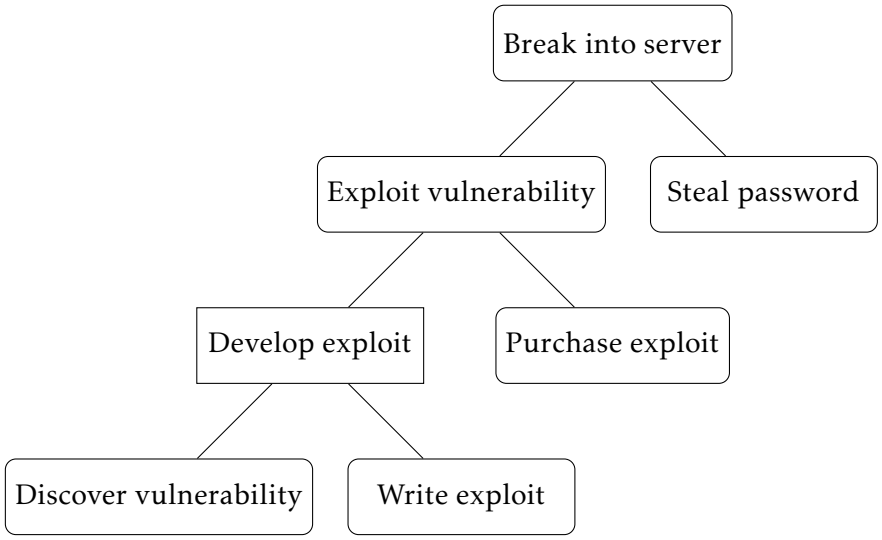

Figure 2: A simple attack tree with an AND node.

Now consider the view of the tree in Figure 2, which expands a different branch of the same tree. Here the attacker is considering the option of bypassing authentication by exploiting a software vulnerability. There are (at least) two options they might consider: purchasing a known exploit for this target, or developing their own. Either would work, so this is an 'or' relationship. When developing their own exploit, however, there are a few components that all need to come together. That is, this is an 'and' relationship, where *all* the child nodes must be satisfied to achieve the parent node. Here, to develop an exploit, the attacker needs to both discover a vulnerability in the software and write an exploit for that vulnerability[9].

The 'and' and 'or' relationships (and the underlying concept of expanding sub-goals in a tree structure) are all the conceptual tools needed to start constructing attack trees. You might notice that some of the leaf nodes in the tree depicted above could be expanded upon, or there are approaches which are missing – see the optional exercises for more about adding to this. However, the main purpose of attack-tree modelling is not to serve as a planning aid for your cyber-attack[10], but to help diagnose vulnerabilities and prevent attacks.

[9]Take Systems & Software Security in your 4th year to learn about how to do this.
[10]Though it can, and probably has been used for this purpose.

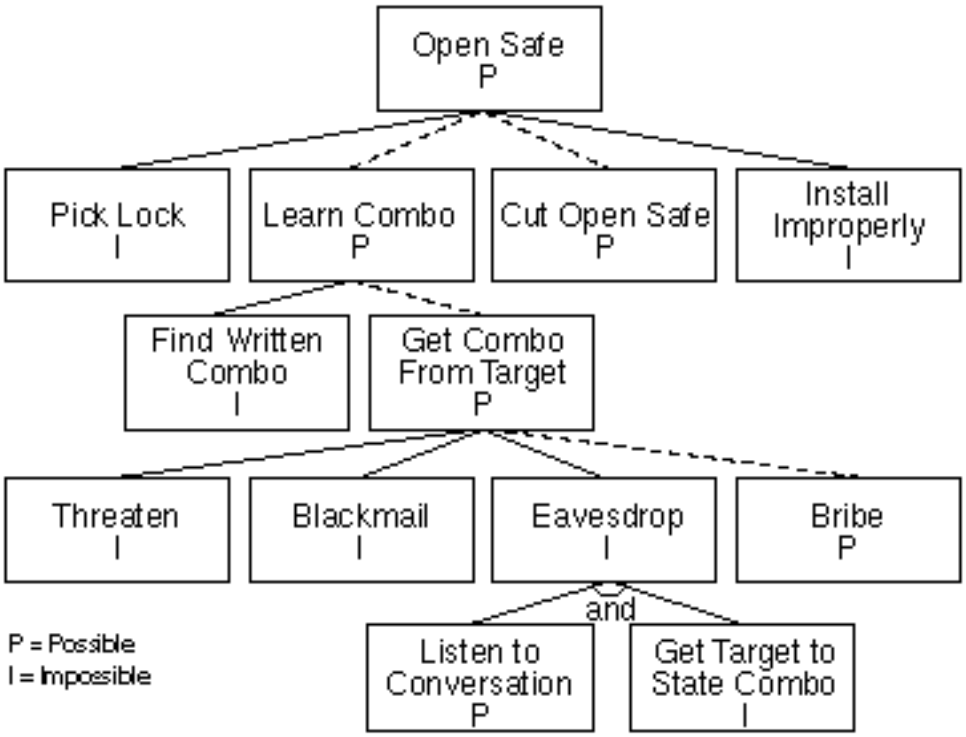

Figure 3: An example of an annotated attack tree, from Bruce Schneier [4]

The way to make use of attack trees defensively is through their annotations. Figure 3 gives an example of an attack tree with a simple binary annotation scheme. For each leaf node[11], the annotator decides whether the event described by the node is possible or impossible in the current version of the system. Then, the rest of the tree can be automatically labelled following some simple rules: if a node is an 'or' node, then it is labelled 'possible' if any of its child nodes are 'possible', otherwise it is 'impossible'. Conversely, an 'and' node is 'impossible' if any of its child nodes are 'impossible', otherwise it is 'possible'. Using this logic, a simple piece of code can quickly tell you whether the root goal is currently 'possible', even for large attack trees containing thousands of nodes. Here, it is possible to open the safe, and we can see two of the routes that permit it – the leaf node 'cut open safe' is marked possible, and the leaf node 'bribe' is marked possible, which percolates upward to make 'learn combo' possible. This tells the security engineer which problems they need to focus on in order to protect the safe. Note that, even though a node 'listen to conversation' was marked 'possible', it doesn't lead to an attack, because of a required component of that branch being 'impossible'. In this example this would be straightforward to spot, but in a large tree for a complex system there can be less obvious relationships between leaf nodes, giving automated analysis real value.

[11] A node with no children.

Any kind of binary annotation could be employed using the rules given above: easy/difficult, expensive/cheap, legal/illegal. However, binary annotations are not always appropriate. Those of you with some statistical training might have been frowning a bit at the notion of 'impossible' on the previous graph – shouldn't we instead be talking about assigning probabilities to each of the events? The good news is that you can define any annotation scheme you like, so long as it has rules for 'or' and 'and' relationships. So you could define an 'or' relationship which takes the maximum of all child nodes' probabilities, and an 'and' relationship that takes their product[12], and your attack tree could then tell you the probability of the attacker obtaining the root node. Many such relationships can be defined for annotations. For example, Figure 4 shows the tree from before annotated with attack costs[13], with the 'and' relationship being a sum and the 'or' relationship being the minimum of the options.

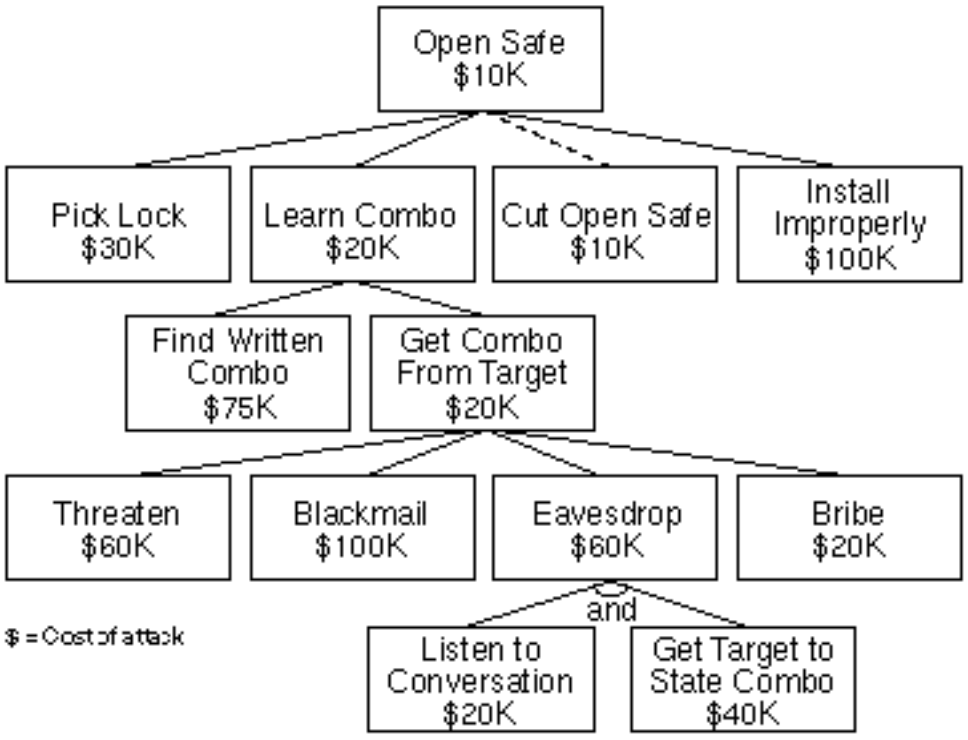

Figure 4: Another annotated attack tree, with costs [4]

[12]This assumes the child node events are independent, which is probably inappropriate.

[13]The cost to the attacker can be a useful way to think about defending your system. Their investment in attacks will be limited by both their available resources and, assuming they are rational, the value of what you are protecting. Investing in preventing very expensive attacks might not be necessary or worthwhile, and it could be sensible to invest in preventing cheaper attacks first.

The fundamental requirement for attack-tree modelling is an understanding of your attacker: what they want, what kind of access to your system they already have, what sort of skills they have, and what sort of risks they are willing to take. By considering these features of the attacker, in combination with knowledge of your system, you can map out paths to the attacker's goal, analyse them, and insert defences. The attack tree model doesn't explain how you arrive at this understanding of your attacker—just like it doesn't explain how you learn about your system—it merely helps you translate this understanding into appropriate countermeasures.

3 OPTIONAL EXERCISES

These are suggestions for students who are particularly interested in threat modelling and some of the topics discussed in this lecture. They are not assessed in any way. You don't need to complete the below to do well on the course.

3.1 *Further Reading*

A natural extension of treating attacks as trees is to treat them as graphs, converging on multiple goals through an interconnected structure rather than separate trees, enabling a fuller probabilistic risk assessment. I encourage the interested student to read the paper by Phillips & Swiler, "A graph-based system for network-vulnerability analysis" [7]. As well as the attack tree papers referenced below, there is also a related literature on the automatic generation of attack trees for particular domains, for which Vigo et al. [8] is a good introduction. However, in industrial settings attack trees can often be considered too unwieldy, and there is a preference for threat modelling approaches that build on data-flow diagrams of a system with simple threat annotations (a key example would be Microsoft's STRIDE [9]). There are a variety of such systems, for which Sass's 'Threat Modelling Field Guide' [10] serves as the best general overview I've yet seen.

REFERENCES

References

[1] J. F. Jackson, "Targeting, Air Force doctrine document 2-1.9," Department of the Air Force, Washington DC, Tech. Rep., 2006.

[2] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.

[3] P. Pols, "The unified kill chain: Designing a unified kill chain for analyzing, comparing and defending against cyber attacks," CSA Thesis, Cyber Security Academy, 2017.

[4] B. Schneier, "Attack trees," *Dr. Dobb's Journal*, vol. 24, no. 12, pp. 21–29, 1999.

[5] E. J. Byres, M. Franz, and D. Miller, "The use of attack trees in assessing vulnerabilities in SCADA systems," in *Proceedings of the International Infrastructure Survivability Workshop*. Citeseer, 2004, pp. 3–10.

[6] I. Ray and N. Poolsapassit, "Using attack trees to identify malicious attacks from authorized insiders," in *European Symposium on Research in Computer Security*. Springer, 2005, pp. 231–246.

[7] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proceedings of the 1998 workshop on New security paradigms*, 1998, pp. 71–79.

[8] R. Vigo, F. Nielson, and H. R. Nielson, "Automated generation of attack trees," in *2014 IEEE 27th Computer Security Foundations Symposium*. IEEE, 2014, pp. 337–350.

[9] Microsoft, "The STRIDE threat model," accessed 2022-10-01. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)

[10] M. Sass, "The enchiridion of impetus exemplar," *ShellSharks*, 2022, accessed 2022-10-01. [Online]. Available: https://shellsharks.com/threat-modeling