Experiences Threat Modeling at Microsoft

Adam Shostack
adam.shostack@microsoft.com

Microsoft

Abstract. Describes a decade of experience threat modeling products and services at Microsoft. Describes the current threat modeling methodology used in the Security Development Lifecycle. The methodology is a practical approach, usable by non-experts, centered on data flow diagrams and a threat enumeration technique of 'STRIDE per element.' The paper covers some lessons learned which are likely applicable to other security analysis techniques. The paper closes with some possible questions for academic research.

1 Introduction

Microsoft has had documented threat modeling methodologies since 1999. These methods have been effective at finding security flaws in product designs, and have been incorporated into the Security Development Lifecycle, a set of processes applied to all Microsoft products with significant security or privacy risks[1] Microsoft continues to invest in updating our tools, methodologies and processes with lessons learned. This paper aims to share information about the history of our SDL threat modeling methods, lessons we've learned along the way (which we think may be of interest outside of Microsoft), describe our current approaches, and share some problems which we hope are of interest to academic researchers.

The author is the owner of SDL threat modeling, including processes, tools and training. As this is a single author paper, the singular pronoun is used for my own opinions, or an authorial voice, while the plural is used to express views which can reasonably be attributed to the organization. For example, in the next section, I do not believe that Microsoft has a formal or agreed opinion on the linguistic question of descriptivism versus prescriptivism.

1.1 What I mean by threat modeling

The term threat modeling has many uses. I now take an intentionally descriptivist approach to "what is threat modeling," and note how the term is used, rather than trying to force a single definition onto it. Some of the more common distinctions include: The term "threat" is used to mean both 'threat-agent,' that is, the person attacking a system, and as a risk, that is, what might go wrong. Threat modeling can refer to a requirements elicitation technique ("what's your

[1] Significant risk is explained in detail in the "Introduction to SDL Process," [8].

threat model?") or a design analysis technique ("can I see your threat model analysis?"). Additionally, threat modeling can be asset-centric, attacker-centric or software-centric. Asset-centric threat modeling often involves some level of risk assessment, approximation or ranking. Attacker-centric sometimes involves risk-ranking or attempts to estimate resources, capabilities or motivations. (Such estimates are challenging for creators of broadly deployed packaged software. Broad deployment often indicates a variety of threat landscapes for different deployments.) Each threat modeling approach has strengths and weaknesses which I focus on as needed to explain other details or lessons learned.

Threat modeling is also used to refer, variously, to analysis of software, organizational networks or systems, or, as in [10] even industrial sectors. Modeling of protocols is often done with a variety of formal methods, sometimes called network threat models. The term network threat modeling is also used to refer to the analysis of a deployed network.

Finally, threat modeling can be done by security experts and shared with engineers, done collaboratively with both engineers and experts, or done by engineers without experts available. Experts might question the value or wisdom of such an approach. There are at least two reasons it may make sense to perform threat modeling without experts. First, experts might not be available because they are in short supply for financial or other reasons. Second, having the people who will build a system – not all of whom are security experts – involved gives them a sense of ownership and an understanding of the security model. Threat modeling encompasses a wide variety of activities in the elicitation of security requirements and analysis of security designs. There is no single "best" or "correct" way of performing threat modeling, but rather, a complex and multidimensional set of tradeoffs which might be made in order to meet a some set of implicit or explicit goals.

1.2 Evaluating Methodologies

One important part of the evaluation of a methodology is the choice of what to compare it to. It is easy to critique these in relation to some idealized notion of "what should be done, absent any constraints." "What should be done"is a great question for research, and we are fully supportive of such questions. There is another important set of questions, especially for a workshop bringing together academics and practitioners. Those questions include "what is typically done" and "what are the barriers to adoption of new methods?" The overarching question in this vein is "what existing processes would be improved by the addition of this methodology, and how much will it cost?" Cost includes getting started (training, setup, software) and ongoing time commitments within a project. Software costs may also be included, but tend to be dwarfed by the human effort involved. The major goals for our threat modeling process are to improve the security of our products, to document the analysis, (both to provide a level of assurance and to re-use the analysis), and to train people to perform implicit security analysis, even when not performing explicit threat modeling tasks.

2 Some History

Threat modeling at Microsoft was first documented as a methodology in a 1999 internal Microsoft document, “Threats to our software” by Jason Garms, Praerit Garg and Michael Howard. This was not the first time anyone threat modeled at Microsoft, but rather the first time the methodology was formalized[2] or considered as an abstracted engineering activity. Additional published Microsoft versions include (at least):

- Writing Secure Code, Howard and LeBlanc, 2001
- Writing Secure Code, Howard and LeBlanc, 2nd edition 2002
- Threat Modeling, Swiderski and Snyder , 2004
- The Security Development Lifecycle, Howard and Lipner, 2006

I list these to illustrate that methodologies evolved, and they evolved and continue to evolve in response to needs we discovered as we worked. I’ll first explain the current system, then explore some of the needs which drove the evolution, and some of the issues that came up. Many of these issues may be of interest to academic researchers who hope to see their innovative systems adopted.

3 Current SDL Methodology

The current SDL threat modeling methodology is a 4 step process, designed to enable engineers with a modicum of security expertise to threat model and have reasonable confidence that they have found important threats. The goals of the process are to improve the security of designs, to document the security design activity and to teach about security as people work through the process. The method involves 4 major steps: diagramming, threat enumeration, mitigation and verification. The process is further described in Hernan, et al [4].

3.1 Diagramming

We generally use a diagramming system derived from standard Data Flow Diagrams (DFD), with the addition of “trust boundaries.” We find the DFD elements of ‘Process,’ ‘Data Store,’ ‘Data Flow’ and ‘External entity’ work reasonably well as a means of eliciting information which can be used to drive analysis. These are shown in table 1. We have added “trust boundaries,” represented by a dotted line or box, which are places where the different sides of the boundary operate at different privilege levels. The reasoning behind the original decision to use DFDs stemmed from two factors. First, it’s easy to understand, and second,

[2] I’m using “formalized” in the sense of “concerned with the form, as distinguished from the matter, of reasoning” [9] rather than as a mathematical formalism to enable theorem proving. Thus a formal methology involves a set of repeatable and documented steps, with defined inputs and outputs.

it's very data-centric. A great many software attacks involve the flow of data through the system in some way, and so DFDs are focused on 'the right thing.' With 7 or 8 years of experience and practice, the DFD approach to diagramming is strongly entrenched. Suggested replacement approaches need to demonstrate clear benefit for us to consider them.

Name	External entity	Process	Data flow	Data store
Representation	rectangular box	circle	directed arrow	parallel lines
Definition	Things outside your control	Code	How information flows between other elements	Data at rest
Examples	People, other systems, web sites	exes, assemblies, COM components	Function calls	files, databases, registry keys

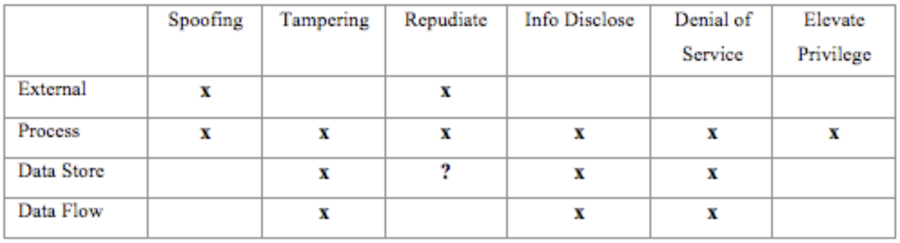
Table 1. DFD Elements

A typical DFD contains between 10 and 150 elements (excluding trust boundaries and any textual annotations on the diagrams.) The main determinants of complexity are trust boundaries and how much detail is needed to clarify what happens as things cross trust boundaries. Many systems are diagrammed "bottom-up" for two reasons. The first is that Microsoft often uses 'feature crews' of developers, testers and program managers who are very focused on their feature or features. It's a natural breakdown of work to ask each crew to threat model their own work. The second reason is that the SDL required threat models of 'all new features and the product as a whole,' rather than some other wording. I raise this to point out that particular wording in descriptions can have unexpected consequences.

3.2 Threat enumeration

Origins Many threat modeling activities at Microsoft and elsewhere, including early SDL processes, have organized around a brainstorming session, or other informal approaches to issue enumeration. Brainstorming sessions are may work for experts, but even with experts, there are issues of completeness and repeatability. We identified a need to provide more prescriptive and easier to follow advice. The method we now use derives from unpublished analysis of CVE and MSRC issues which was performed by Shawn Hernan and Michael Howard.

Current Methodology Our current methodology uses the diagrams in a technique we call "STRIDE per element" to provide guidance for non-experts, as well as repeatability. The technique is based on the observation that the software architecture threats we are concerned with are clustered. The essence of the technique is to note that for each type of element within the DFD, there are threats we tend to see, and thus look for elements as shown in table 2.



	Spoofing	Tampering	Repudiate	Info Disclose	Denial of Service	Elevate Privilege
External	x		x			
Process	x	x	x	x	x	x
Data Store		x	?	x	x	
Data Flow		x		x	x	

Table 2. STRIDE Threats Per Element

For data stores which are logs, we are concerned with repudiation issues, and attacks on the data store to delete the logs. In our current tooling, we use a set of questions to make these threats more concrete and accessible.

Analysis I make no claim of universal applicability for this enumeration. It is focused on those issues for which Microsoft issues updates. Other organizations might extend or replace it. For example, "information disclosure by external entity" sounds like a good description of a subset of privacy issues. Alternately, another organization, focused on Web 2.0 apps might replace the list of threats with ones tuned for their environment.

In our current tooling, we offer guidance about how each of these threats manifests against each type of element. It is clear that more specific guidance would be very helpful. For example, the opportunities, methods and mitigations for tampering with a data flow are very different between an HTTP GET request and a Windows LPC (Local Procedure Call) or a Unix named pipe.

3.3 Mitigation

The SDL threat modeling training and documentation discusses four approaches to mitigation, in order of preference: redesign, use 'standard' mitigations, such as ACLs, use unique mitigations (with caution) or accept risk in accordance with policies.

From a practitioner perspective, connecting a model to practical resolution steps is important on a number of levels. First, improving system security is the goal of modeling. Providing a connection between an identified problem and a way to address the problem makes that easier. Second, there is a psychological component. Telling an engineer that there is a problem without providing fix information will frustrate many engineers[3]. (More on the psychological factors in section 4.5 below.)

3.4 Validation

We provide a number of heuristics for validating threat models, including graph analysis of diagrams, checking that the final diagrams reflect the final code, that STRIDE threats per element have been enumerated, that the whole threat model has been reviewed, and that each threat is mitigated. We would like to be able to compare models to code in many languages. (See section 5.1 for more.)

3.5 Analysis of the methodology

This approach is in use across some very large software products, and is embedded in both the SDL process and other software development processes. These other processes provide much of our assurance as to the correctness and completeness of the threat models produced in relation to the software. Delegating the correctness and completeness to a broader set of processes has benefits and costs. The benefits are in accountability and integration. Individuals are required to sign-off that threat models are complete with regard to trust boundaries and accurate with regards to what is being built. The primary cost is that the lack of strong theoretical underpinnings makes it hard to connect the models to academic work which has been done. Overall, we believe that the simplicity of the approach, the integration into the development process, and new tooling (not presented here) provide for a very effective approach for identifying and addressing design issues in commercial software development.

One of the anonymous reviewers asked for evidence that 'the approaches taken are the right ones.' I make no such claim. I do claim that the approaches are *useful*, for the challenges that we have identified and which I have discussed. In light of the many meanings of threat modeling and the many goals which processes may serve, I don't believe that there is a right or wrong approach, only ones that are more or less useful.

4 Issues Encountered and Lessons Learned

4.1 Threat modeling as an aspirational tabula rasa

With threat modeling having many meanings (as mentioned in section 1.1) many people projected their beliefs about what threat modeling ought to be into a methodology, and added steps to the methods. These steps were often added without understanding the costs, benefits or issues theymight generate. An example of this is the addition of the DREAD risk assessment technique.[6] DREAD may work for some systems, but for software-centric threat modeling,

[3] Some engineers will relish the challenge. In security, experience shows that naive approaches rarely work. This combination is usually frustrating.

it seems to add numbers without defining their scales, generating a risk of making a risk assessment appear algorithmic when it's not However, Microsoft or SDL threat modeling "obviously" needs a risk management technique, and so DREAD was added.

4.2 Complexity

Many participants in threat modeling have never received formal training, but have picked up what to do informally from others. I was once surprised to find an entire security team, all of whom were conversant in threat modeling techniques and jargon, none of whom had never been to a training class. Given the very large number of security classes, tools and techniques available within Microsoft, it is reasonable to assume that most practitioners have at most, 2 hours of threat modeling training in the last few years.

The method enumerated in the Security Development Lifecycle book has 9 steps. Some others have had as many as 11. The difficulty of each step can vary widely, from "describe use scenarios" to "determine threat types." The former is probably reasonable to ask of most engineers, the latter is almost certainly out of the reach of non-experts, and likely to engender strong disagreements between experts. Methodology descriptions are full of jargon, adding complexity for small benefit.

Designers of methodologies should pay careful attention to the demands made of their users, and consider the value of each step or element.

4.3 "Connectedness"

Threat modeling activities can seem very disconnected from software development. The agile motto of "YAGNI" ('You Ain't Gonna Need It') can seem like it was coined for threat modeling. A number of approaches have helped integrate threat modeling into development process, including treating threats as bugs and mitigations/countermeasures as features. Bugs and features are understood by developers, and organizations know how to handle them. Additionally, we encourage teams to use the question "can I see your threat model" as a way to kick off security discussions. This generates peer pressure to have a good threat model. [4]

There are two additional design considerations which I wanted to call out specifically, both relating to people. The first is humans within the threat model, and the second is the human factors engineering issues related to the design of security modeling methods, process and tooling.

[4] More generally, approaches like the Microsoft SDL, with a need to address the variety of development methodologies, can seem disconnected. The point is not unique to threat modeling.

4.4 People in the security model

There are a number of considerations about people in security models. Ellison has argued [3] that all network protocols, considered fully, include both computers and people. A failure to effectively model people leads to problems such as phishing, where three levels of poor authentication combine to result in tremendous fraud. (The three levels are schemes which have proven insufficient for authenticating email, web sites and users.) While modeling users is tremendously challenging, failure to do so demonstrably leads to important security issues. How to help engineers do this effectively is an open problem, which has also been considered by Cranor [2].

4.5 People as users of a security modeling process

Engineers, designing for engineers, sometimes ignore usability concerns. There is a great deal which has been written on user interface design, human-computer interaction, and related topics, and I assume that system designers are familiar with it. I wanted to raise three points which are important to security modeling systems intended for use by a broad set of engineers. The first is to clearly state your expected user and their skillset. Advice we gave, such as "think like an attacker," was meaningless to many people [5]. I find challenging security experts to think like a professional chef helps illustrate the feeling.

The second is to design explicit development integration points for the security modeling methodology. The integration points may seem obvious to the designer, but might not be to someone learning the system for the first time. An explicit statement of how to integrate into agile and waterfall processes can help overcome this hurdle. In particular, explaining what outputs might be bugs or features can be very helpful. More generally, security modeling is done by a wide group of people with different skillsets and goals. Bring a diverse toolset to the problems, and look at the problem end to end (including the engineers and the organizations that they work for).

The third and final point is perhaps obvious but worth a little emphasis from the industrial perspective. Processes which can be used by any intelligent and skilled engineer will be more broadly used than processes which require unusual skills. Ease of use and clear documentation are important, and often receive little attention, even in work labeled "practical."

5 Open Research Problems We See

The methodologies outlined above are less grounded in mathematics than many presented at academic events. As I have emphasized, this approach has benefits

[5] Additionally, people were reluctant to admit that they didn't know how to "think like an attacker." The way in which it was often stated implied it was the most natural thing in the world. So knowing your model of the user is important, as is testing that model actually applies.

and costs. We would like to see more collaboration between the broad software industry and the modeling community. Towards this end, I have attempted to present clear explanations of the motivations and crucial factors within the decisions we have made. I am hopeful that the 'applied' side of the academic community can incorporate some of these factors in their assessment of what constitutes interesting research. In our roadmap for our processes and tooling, there are many practical engineering problems, some of which I have alluded to. There are others which we hope are interesting to the research community, and I present three of those here.

5.1 Relating models to code

There are many benefits to our lightweight model. At the same time, the model is not the code, and we would like to be able to address two scenarios: The first is that we have a body of code for which no threat model or DFD exists, for example after an acquisition. We would like to be able to rapidly create models from the code to accelerate threat modeling activity. The second scenario is that we have threat models and code, and would like to be able to compare them. Does the model contain all of the trust boundaries and entry points as the code? Are there links in the code which are important to, and not in, the model?

Some work has been done [1] on using Reflexion Models to semi-automatically derive models from code and compare them to user-created models, but much work remains to be done to completely automate the derivation of models from code. In particular, there is a tremendous amount of existing code written in C and C++. These languages are more challenging to model and analyze, but the payoff for such modeling could be very large.

5.2 Extending models with validation information

Much security advice involves "accept only what is explicitly allowed." The idea is that an engineer knows what they expect, and can deny all other input. Assuming this is locally true, much data which enters a system has been validated for some whitelist or purpose, such as "valid IP packet" "valid POP3 message." However, this data is not generically trustworthy, it's trusted for some set of purposes. Defining that set of things is easy for small problems, and seems to be hard to generalize. It would be very useful to have languages to describe such intent, and tools which could use such information to improve security model analysis, code, or both.

5.3 Threat model measurement

Today we create a great many threat models. There are some simple measurements we could take (number of elements, various completeness or verbosity measures). What is less obvious is what measurements we should take, and why. What aspects of a threat model would most accurately indicate that our goals of security analysis, assurance and training are being met? More generally, what indicators relate to which goals, and how? How much does it cost to gather a measurement? (A brief story: One of our threat modeling discussion lists received an email asking about how to approach a certain problem. A number of people answered, and there was rapid iteration over a variety of design possibilities. The team selected one. It is unlikely that any of the possibilities were ever formally documented. I think this was a reasonable choice: we got an improvement in security at low cost. The value to documenting the iteration is low.) Is there a way we could analyze models and determine the likelihood that it has iterated? What other measurement approaches could aid developers or decision makers?

6 Acknowledgements

I would like to thank the anonymous reviewers, Shawn Hernan, Steven Lipner J.D. Meier and Glenn Pittaway for helpful comments on drafts, and colleagues, too numerous to list here, for illuminating conversations about threat modeling. I am also thankful to Bruce Schneier for blogging about the paper, and to several of his readers for drawing attention to ambiguities which are perhaps now less ambiguous. Curtis Koenig pointed out an important error in table 1, where the representations were swapped.

7 Conclusion

This paper has briefly described some history of threat modeling as practiced at Microsoft. It shares details of the current process and some lessons learned. It has also presented a selection of perspectives research problems that we hope are of interest to the academic research community.

References

1. Marwan Abi-Antoun, Daniel Wang and Peter Torr, Checking Threat Modeling Data Flow Diagrams for Implementation Conformance and Security, ASE'07, 2007. Atlanta, Georgia, USA
2. Lorrie Faith Cranor, A Framework for Reasoning About the Human in the Loop, Symposium on Usable Privacy and Security, 2008. Pittsburgh, Pennsylvania, USA
3. Carl Ellison, Ceremony Analysis, Microsoft ThinkWeek paper, January 24, 2005. A version of this paper appears as Cryptography ePrint 2007/399, October 2007. http://eprint.iacr.org/2007/399
4. Shawn Hernan, Scott Lambert,Tomasz Ostwald and Adam Shostack, Uncover Security Design Flaws Using The STRIDE Approach, MSDN magazine, November 2006
5. Michael Howard and David LeBlanc, Writing Secure Code, Microsoft Press, 2001
6. Michael Howard and David LeBlanc, Writing Secure Code, 2nd edition Microsoft Press, 2002
7. Michael Howard and Steven Lipner, The Security Development Lifecycle, Microsoft Press, 2006

8. Microsoft Corporation, "SDL Process Introduction," 2008, `http://msdn.microsoft.com/en-us/library/cc307406.aspx`
9. Oxford English Dictionary Online, visited July 14, 2008
10. Craig Rubens, Cleantech Terror Alert: Hacking the Grid, Earth2Tech, June 26, 2008, `http://earth2tech.com/2008/06/26/cleantech-terror-alert-hacking-the-grid/`
11. Adam Shostack, Reinvigorate your Threat Modeling Process, MSDN Magazine, July 2008
12. Frank Swiderski and Window Snyder, "Threat Modeling," Microsoft Press, 2004