Software Security lesson 4 - threat modeling

Jukka-Pekka Kinnunen

April 16, 2014

Abstract

This is a set of University of Helsinki course "Software Security" lecture notes given by Antti Vähä-Sipilä April 2nd 2014.
1 Vocabulary

Concepts of threat and risk are used almost interchangeably even though they do have overlapping meanings. However threat actors, or attackers are always people.

Threat modeling has several levels, at least business level, technical level, and attacker profiling of which the technical level can also be called Architectural Threat Modeling.

2 Threat Modeling

2.1 Business Level

On the business level the business impacts are the base of decision making whether the question is about the value of the software, reputation of the company or reputation of customer. The decision maker is normally the person in charge of either the product (production manager) or service (service manager).

On the business level the analysis is normally unstructured discussion process of risk management in which all findings are probable risks. Ignoring the risk is stupid unless done intentionally - in which case the risk is observed but accepted.

2.2 Technical Level

2.3 Architectural Threat Modeling

Architectural threat modeling is also known as ARA, Architectural Risk Analysis and it has most value in software (security) projects.

The modeling is best done by drawing, preferably by hand on large paper and in teams so as many as possible may have their say. Computer aided tools do exist but they may hinder the thought process when brainstorming and are best used afterward to finalize the drafts.

The models are not pictures for pictures sake but a tool that extends human capacity of handling abstract data and extends your memory. When drawing a picture any omission of detail becomes more obvious.

---

1 I do think there should be no difference between software project and software security project, as the security should be built in the software.
Attack trees may be used so that for every reason (node of a tree) corresponds one or several leaves (consequences or things that happen) which in turn are used as nodes growing more leaves. However this may lead to recursive trees or that the trees degenerate to graphs which does not easily lend to mitigation. Lectured regarded attack trees as "something academics write but nobody uses".

Data flow diagrams (DFD) are tools for data flow analysis. Basically any physical machine is a box. Any virtual machine is a box. Any process, service, dynamic or static library is a box. Between the boxes flows data and whenever it crosses a box it crosses security boundary or trust boundary or at least a question whether the boundary exists must be asked.

Data flows over protocols so the TCP levels could be spelled out but normally should only if it matters.

Message Sequence Chart (MSC) are useful to depict I/O data round-trips.

Lecturer expressed as his opinion that the UML Use Case Diagrams are least useful from Threat Modeling point of view, User Stories however are useful. User Stories could be appended with Abuse Stories, textual descriptions how an Abuse Case goes.

Component Diagrams are useful only to some degree as the classes are not necessarily useful but the inputs and outputs are crucial.

Draw only those diagrams that matter.

As an example lecturer used Web Server diagram (of which I may have a picture soonish). The questions "is it a process?", "does it have load balancer?", "are there several servers behind LB?" were examples of questions that should be asked when drawing a threat model. Does it run Apache? Which protocol? Is there PHP? If there is, where is the configuration? Is there a database? Which protocol does it use? Where are the SSHD keys?

When drawing picture for threat model lecturer Antti Vähä-Sipilä proposed using just boxes to depict about anything from physical hosts to virtual hosts to single processes to even threads in a process as any of these is also a security boundary, a central concept in threat modeling. Basically anything in the same security boundary belongs in the same box, and if there is separation of trust there should be a boundary also.

Security boundary is a structural limit where the trust ends.

As such it is technical reason for separation.

Each and every place where data flow crosses trust boundary is - or should be - considered thoroughly as these are the attack surfaces through which an attacker may try to pry information by injecting fuzzed data and interpreting output data.
Storage may be depicted as magnetic drum regardless of the actual type of storage. According to Vähä-Sipilä storage could and maybe should be considered as a data flow in time.

2.3.1 Attack surface enumeration

After the picture is drawn all the attack surfaces are enumerated. Each attack surface has test requirements, for example "test invalid password inputs".

Any place which is an endpoint of protocol (or protocol layer) needs to be tested.

At this point the person doing threat modeling should be careful when assuming anything and rather defining test for each assumption. Beware of assumptions!

When enumerating attack surfaces one should not forget Administrator as attack surface. With Administrator or root privileges all boxes (security boundaries) inside a certain host are transparent.

What limits the enumeration is how much risk you are willing to accept. Blocking and checking everything is not possible or cost-effective so there is a balance between costs and acceptable risk.

2.3.2 Inputs Matter

From the architectural threat modeling point of view the crucial thing is the data that ends up in the process confusing the inputs.

As the previous lessons and exercises proved, the inputs matter, because that is the way to obtain "weird machine" or to make a system to behave outside of its specified functionality. What interests the attacker most, is the unspecified functionality, which may be induced for example fuzzing the inputs.

As the XSS exercise demonstrated the software is hard to protect with negative requirements as they lead to negative test cases and the test coverage of something non-existent is next to impossible to achieve.

"List all those files You do NOT have" is a real-life request in a real life situation where some thousand files went missing after a data flow - or rather a data tsunami - filled quota but the server daemon did not notice and churned on.

Basically there are three layers of input validation.

First of all, the input may be filtered allowing only the subset of data matching a pre-set pattern to enter. For example a social security number has only limited number of characters and only a limited subset of all characters.
Secondly, the input from uncertain source which we cannot fully trust, shall be marked as tainted, and considered unsafe until it is untainted. For example in Perl, the most used way to untaint tainted data is to match input with a regular expression (and hope the regex is tight enough to filter malicious input) and use only the match reference - not the original input - in assignment.

And finally the output should be sanitized so that even though the input data were fuzzed, the output would only contain protocol-wise correct data. For example, if an output should contain only ISO-8859-1 characters, emitting UTF-8 would be an error.

### 2.4 Software projects tools

There are some software projects tools advertised as security tools. Usability depends on the project. Not all tools are equal and a hammer makes a poor screwdriver.

Basically a threat analysis should concentrate on what could go wrong, what are the requirements and how they can be tested.

Some of the tools are simply too slow to be useful for example in a project where Continuous Integration (CI) is used and a build system runs test set for each build. For example if statical code analysis tool takes several hours for doing its job it does not lend itself well to a project with hourly build.

### 2.5 Methodology

Crucial security terms are CIA (Confidentiality, Integrity, Availability).

STRIDE is an acronym from Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of privilege. It is also a methodology for developers to mitigate problems with each of these parts.

Spoofing: "How do we know the server is right?" or "How do we know the client is right?"

Tampering: "How can we be sure the data is correct?"

Repudiation: Usually in form of non-repudiation, undeniability of responsibility. "Do you have logs?" "Are the logs protected?" On privacy viewpoint there are some issues with gathering repudiation data.

Information disclosure: "Can data be spied on?"

---

2This gets a bit hairy when we realize that Western European ISO-8859-1 characters are in fact also UTF-8 characters as ISO-8859-1 is a subset of UTF-8
Denial-Of-Service: "What can we do?" "What are the (counter)measures?"

Elevation of privilege: "May the data be interpreted as code?" For example SQL injection, rootkits.

2.6 Attack analysis

Attacker analysis or attacker portrait is just the old adage "know your enemy". The potential attackers may be enumerated and the level of risk evaluated against the protection costs. For example the protection against random "script kid" costs less and has both greater probability and greater probability to succeed than protection against any military intelligence operation.

2.7 Threat modeling in agile project

Threat modeling may be used in an agile software project where product backlog is used as two-way communication tool between product owner and project team. Usually the team codes from the top of the list (the items or features with most business value). Threat modeling results are pushed back to the list so that they eventually get coded. Threat modeling may also create feature acceptance criteria.