Some Thoughts on Threat Modelling

Peter Gutmann

University of Auckland

Traditional Threat Modelling

Think up threats until you get bored, then declare victory

Often leads to circular reasoning

- "Our threat model is whatever our application defends against"

Example: DNSSEC

- The requirements document (RFC 4033) wasn't written until a decade after the protocol was created

Traditional Threat Modelling (ctd)

SSL was somewhat similar

- The Internet Threat Model:
  "I'm OK, you're OK, and eavesdropping on credit card transactions is the threat"
- The actual threat model:
  "I think I may be OK, I don't know about about you, and any Internet-based eavesdropping is so vanishingly small as to be virtually nonexistent"

Result: Multibillion(?) dollar global phishing industry

Traditional Threat Modelling (ctd)

Use a standard methodology (ITSEC, Common Criteria, PCI-DSS, ...)

- Work your way down a checklist making sure that you have a response to each item
- Standardised defences that can only deal with standardised attacks

Traditional Threat Modelling (ctd)

Only works if the attacker is using the same checklist as the defender, and carefully follows their instructions

> [Attackers would] be intrigued and go to Safe2Login.com to learn about the product, notice that it records all usage and issues immediate alerts on all misuse, and move on to a target with a lower risk of hacker identity discovery
>
> — Safe2Login documentation

This is not an effective defence strategy

Traditional Threat Modelling (ctd)

Risk mitigation

- Document the risk
- Get sign-off on it

This is an even less effective defence strategy

Notable Threat Modelling Failures I

In the 19th century with the introduction of HE shells, fortress designers went underground

Noted Belgian designer Henri Brialmont fortified Belgium with a string of underground fortresses

- Threat model assumption: You can't move artillery of more than 210mm calibre over land
- No road or bridge, built for horse-drawn traffic, could take the load

Notable Threat Modelling Failures I (ctd)

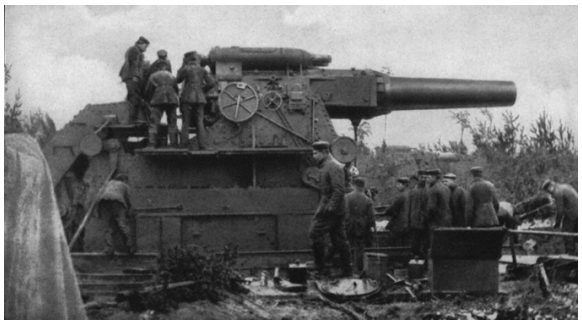
This explicit model sent an unfortunate message to potential attackers

Your shells must be at least this large to enter

Notable Threat Modelling Failures I (ctd)

German military went to Krupp, the armourer of Europe

- Krupp had a 420mm gun capable of attacking the forts
- Weighed 150 tons
- Required a custom-built concrete emplacement ("bettungsgerät")



Living proof of Brialmont's threat model

Notable Threat Modelling Failures I (ctd)

Could they make a lighter version?

- Weeeellll, maybe, if you're prepared to accept a range reduction from 10,000 to 9,500 metres
- Created a ~~portable~~ relocatable version of 40 tons
- Could be broken down into five components for transport
- Each piece weighed less than a standard 210mm gun

These were the famous (and rather unjustly named) Big Berthas

Notable Threat Modelling Failures I (ctd)

The defenders were totally unable to cope with this

- Shell weight increases with the cube of the calibre
- Depending on the shell type, these were *ten times* the maximum size that the forts could handle

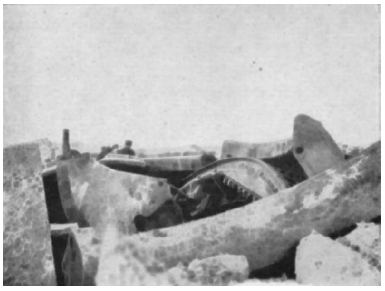
When the first shell struck the surrounding forts assumed that a lucky short had detonated the magazine in the other fort

- There was no other way to explain the size of the explosion

Notable Threat Modelling Failures I (ctd)

Attackers knocked out the forts protecting one sector of the town of Liège

- Wheeled the guns into the centre of town
- Rotated them around like clock hands taking out each fort



- (Another security lesson: Don't bother putting an iron padlock on a plywood door)

Notable Threat Modelling Failures II

Developers of the Xbox took great pains to create a locked-down environment

- Widespread piracy of games scares away developers

Some portions of the hardware were heavily protected, others weren't

- Threat model assumption: You can't read data off the high-speed system buses
- Memory bus = 6.4 GB/s, HyperTransport bus = 400 MB/s
- At the time, this was a quite valid assumption

"Your sampling hardware must be at least this fast to recover unprotected content"

Notable Threat Modelling Failures II (ctd)

MIT student Andrew "bunnie" Huang noticed that HT used the same signalling as LVDS

- Could use an off-the-shelf LVDS transceiver to pull data off the bus

This still left 400 MB/s of data to decode

- Standard approach is to throw an FPGA at it
- But… no regular FPGA could handle this data rate
- Specifically, no regular FPGA run as the manufacturer intended could do this

Notable Threat Modelling Failures II (ctd)

What if you …

- Used an oscilloscope to characterise the performance of individual FPGA elements to locate the fastest ones

… and …

- Hand-optimised that data paths through the FPGA switching fabric rather than using a VHDL compiler

… and …

- Clocked the data onto four phases of a quarter-speed clock
- Transforms the 8-bit stream into a 32-bit stream at ¼ the speed

… and …

- Overclocked the FPGA

Yeah, that should about do it

Notable Threat Modelling Failures II (ctd)

Next-generation Xbox360 included extensive threat modelling and interlocking defence mechanisms to ensure that a break of one element didn't give the whole game away

A Brief Introduction to DFD-based T.M.

Information flow diagrams make a great tool for threat modelling

- Model flows of data into/through/out of code modules
- Trust boundaries represent points at risk

Data flow diagrams (DFDs) are a standard tool for modelling information flow

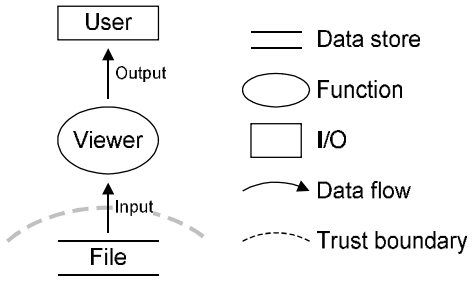
- Been around since the 1970s
- Supported by numerous drawing/modelling packages

Originally intended as a software design tool

- About as useful as flowcharts and other similar tools
- Repurposed for threat modelling by Microsoft

A Brief Introduction to DFD-based T.M. (ctd)

Example: Image viewer



- Image data crosses a trust boundary
- This is an attack vector

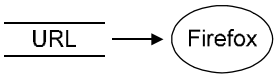
(OK, it's a greatly-simplified example to show how DFDs work)

Notable Threat Modelling Failures, Part III

How do you allow non web-enabled applications to direct users to web pages?

- Example: Link to the developer's home page from the app's About box
- Example: Web-based online help

Register a URI handler for the browser

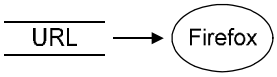


- `ShellExecute( "http://…" );`

Pretty simple, right?

Notable Threat Modelling Failures, Part III (ct

But wait, why is the URL coming from a fixed data source?

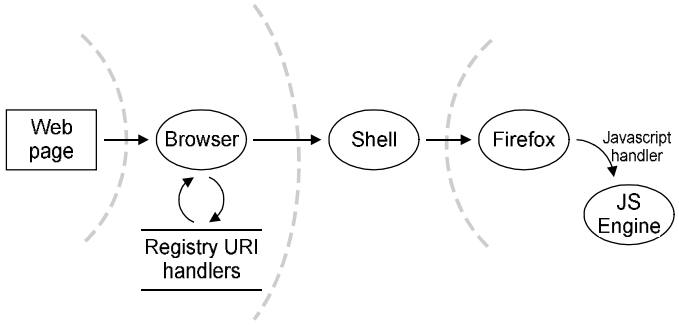


Is this really the full DFD?

- Data doesn't spontaneously appear on disk
- At the very least there's a trust boundary between the URL source and the browser

Notable Threat Modelling Failures, Part III (ct

Let's do this properly…

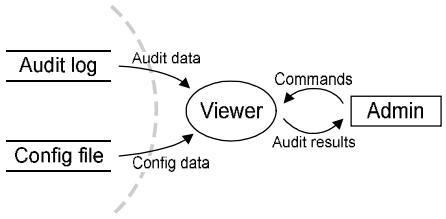


- Even without drawing this full DFD, simply identifying the presence of the trust boundary between the URL source and Firefox in the original DFD would have helped

Thoughts on DFD-based Threat Modelling

DFDs are a useful tool for documenting the assumptions that you're making in your threat model

Example: Audit log viewer



"Audit events are stored in a secure manner"

- How do you draw a DFD for "in a secure manner"?
- Secure relative to what?
- Who wrote this policy anyway? An accountant?

Thoughts on DFD-based Threat Modelling (ct

Assumptions

- Audit log is only writeable by the audit user
- OS and filesystem protection mechanisms are operating as intended
- Audit user account (and equivalent high-privilege accounts) haven't been compromised
- Security of the OS kernel hasn't been compromised, e.g. through a rootkit

Now you've got something that you can actually work with

- (Use of a well-known example means that the threats are pretty obvious, it's not so obvious for lesser-known cases)

Thoughts on DFD-based Threat Modelling (ct

Example for filesystem protection mechanisms

- What if someone mounts the drive under another OS that doesn't respect the access control mechanisms?
- What if it's a USB key using FAT?

Use cryptographic protection mechanisms to detect tampering

Thoughts on DFD-based Threat Modelling (ct

Example for subverted OS

- Cryptographically tie the information to other systems under the assumption that the attacker can't compromise all of them

In general though what this threat model shows is that you can't trust any audit data coming from a compromised machine

- Asking the drunk whether he's drunk

Thoughts on DFD-based Threat Modelling (ct

This is well known for the standard case of audit data, but less obvious for other cases

Embedded device controlled via HTTP over SSL

- Single OS/application image
- FAT filesystem
- Download the default config file from the manufacturer's web site and overwrite the device configuration via pathname tricks
- Fixed by creating an internal reference monitor a la VFS for a BSD jail-like effect

Thoughts on DFD-based Threat Modelling (ct

DFDs can help identify impossible tasks

- The null hypothesis test, instead of spending forever trying to solve it, show that it's unsolvable
- This was used in e.g. Windows Vista to get a network access tool pulled from the final release when threat modelling showed that there was no way to secure it

Thoughts on DFD-based Threat Modelling (ct

Modelling componentised applications, e.g. LAMP stack

- Web front-end to an online store
- Online store talks to a database back-end
- Database controlled via the online store application

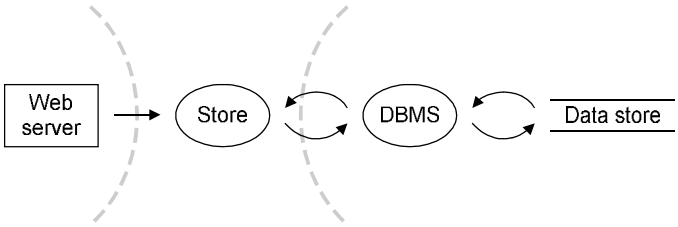
Thoughts on DFD-based Threat Modelling (ct

All are carefully written to be secure

- Web server is implemented in a standards-compliant manner and carefully passes requests on to the online store application
- Online store treats request data as opaque blobs to avoid data-formatting attacks
- Database knows that commands fed to it are safe because it's controlled by the online store app, which would hardly attack its own data store

Thoughts on DFD-based Threat Modelling (ct

Wait a minute...



Each individual component is perfectly secure

- The composition of the secure components isn't

Again, the use of a well-known example means that the threats in this case are obvious

- What about less well-known cases, like `firefoxuri:`

Thoughts on DFD-based Threat Modelling (ct

Identifying vulnerable interfaces

- ☑ addURL()
- ☑ executeSQL()
- ?? setAttribute()

Trying to create a DFD for this shows up the problem with the polymorphic interface

- Can find obviously vulnerable interfaces with `grep -i sql`, but the less obvious ones require a more rigorous approach
- If you can't decide whether you have a trust boundary at a given location then you have a problem

Thoughts on DFD-based Threat Modelling (ct

Where are your trust boundaries?

- Application calls `libx` which calls `liby` which calls `libz`
- Do you have to threat-model all of `/usr/lib` or `/Windows/System32`?

DFDs help identify hidden trust boundaries

- We call the LDAP library, which we assume is safe, but …
- … what about LDAP injection?
- What about protocol-I've-never-heard-of injection?
  - How many `firefoxuri`'s are there hiding on people's machines?
  - Have a look at the Windows registry's catalogue of file type and URI handlers some time

Other Threat Modelling Techniques

Checklist-based (many, many)

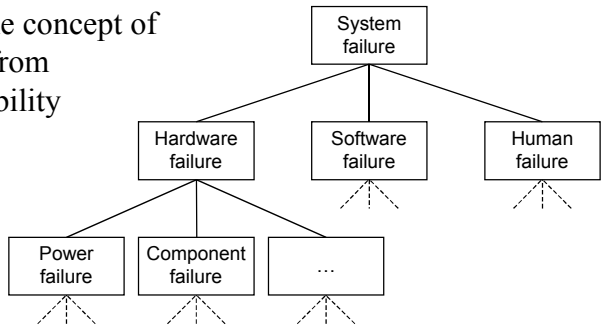
- See earlier slides

PCI-DSS gets its fair share of flak, but it's better than the nothing-at-all that preceded it

Other Threat Modelling Techniques (ctd)

Attack trees

- Based on the concept of fault-trees from high-availability systems



Works for non-malicious faults

- Can calculate the probability of each failure type and address any likely-to-occur ones

Other Threat Modelling Techniques (ctd)

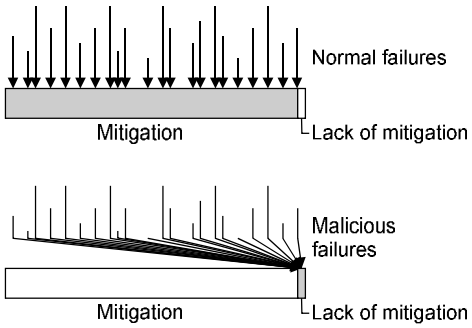
Doesn't work for non-malicious faults

- Would need to enumerate every possible attack type
- Including ones that no-one's thought of yet

Even if you could enumerate every possible attack, how would you match them to your application?

Other Threat Modelling Techniques (ctd)

Attacker can choose the attack that they want to use



- Probability of failure can be 100% for every attack, not just a manageable subset

Other Threat Modelling Techniques (ctd)

If anyone knows of anything else that works that isn't either a checklist or boil-the-ocean, come and see me afterwards

Threat Modelling Summary

Pen-testing: There-exists

- Is there *a* weakness in this?

Threat modelling: For-all

- Are there any remaining weaknesses in this?

Identify uncommon/unknown attack vectors

- Best practice: Store your private key on a removable USB token and keep it safe
- They all have FAT filesystems
- Any unprivileged user/process on the system can potentially read your data

Document unstated underlying assumptions