ORSHIN

D2.2

Report about security requirements

Project number:	101070008
Project acronym:	**ORSHIN**
Project title:	Open-source ReSilient Hardware and software for Internet of thiNgs
Project Start Date:	1st October, 2022
Duration:	36 months
Programme:	HORIZON-CL3-2021-CS-01
Deliverable Type:	Report
Reference Number:	CL3-2021-CS-01 / D2.2 / 1.0
Workpackage:	WP2
Due Date:	Jun 2023 – M09
Actual Submission Date:	28th June 2023
Responsible Organisation:	ECM
Editor:	Daniele Antonioli
Dissemination Level:	PU
Revision:	1.0
Abstract:	To address Task 2.2, we present the AttackDefense Framework (ADF), a framework to set abstract security and privacy requirements into concrete policies on the ORSHIN Trusted Life Cycle (TLC) and related devices. We show the ADF design based on a flexible, updatable, and reusable data structure, we call the AD object. We implement the ADF with state-of-the-art open-source software and we will open-source our implementation. We successfully evaluated the ADF in seven complementary case studies demonstrating, among others, its capabilities to drive research for ORSHIN's WP2, WP3, WP4, and WP5.
Keywords:	ADF, Threat modeling, Requirements, TLC, OSH

Funded by the European Union under grant agreement no. 101070008. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Editor

Daniele Antonioli (ECM)

Contributors (ordered according to beneficiary numbers)

Stefano Cristalli (SEC)
Arianna Gringiani (SEC)
Tommaso Sacchetti (ECM)
Olivier Thomas (TXP)
Clarisse Ginet (TXP)
Jesse De Meulemeester (KUL)
Márton Bognár (KUL)
Benedikt Gierlichs (KUL)
Linde Nouwen (KUL)
Volodymyr Bezsmertnyi (NXP)

Reviewers

Maria Chiara Molteni (SEC)
Frank Piessens (KUL)

Disclaimer

Executive Summary

We present the design, implementation, and evaluation of the **AttackDefense Framework (ADF)**, a framework we developed to address the goals we set in Task 2.2. Specifically, ADF can be used to map abstract security and privacy requirements into concrete policies and mechanisms for the ORSHIN Trusted Life Cycle (TLC) and its related open-source hardware (OSH) devices. Moreover, we empirically show that ADF is useful in driving the research of ORSHIN's technical work packages, i.e., WP3 (Models for formal verification), WP4 (Effective security audits), and WP5 (Secure auth and comms).

ADF enhances and extends current best practices around threat modeling and its three steps (i.e., system modeling, threat identification and ranking, and defense strategy). For example, it can be used to threat model attacks and defenses, security and privacy, hardware and firmware, processes and products, and high-level and low-level threats. None of the state-of-the-art threat modeling solutions address all these requirements at the same time.

The ADF's building block is the **AD object**, a novel, flexible, and extendible data format to store information about a threat. Every AD object is unique and is easy to (re)use, update, and read by machines and people. Multiple AD objects (ADs) can be combined and arranged to threat model ORSHIN's TLC and devices. For instance, we can create a collection of ADs representing high- and low-level attacks and defenses in a specific scenario (e.g., IoT firmware threats). Then, we can filter the ADs based on attack surfaces and vectors, arrange them in a tree to visualize their dependencies, create chains to represent complex exploits and map them to known threat taxonomies.

We implemented our design with the `ADF` toolkit. Our toolkit is based on free and open-source libraries (e.g., pandas and graphviz) and programming languages (e.g., Python and YAML). It includes extendible collections of ADs covering domains relevant to ORSHIN (e.g., hardware, software, firmware, and protocol attacks and defenses). It has parsing functions to automatically parse ADs from popular serialization formats (e.g., YAML, TOML, JSON and XML). Moreover, `ADF` contains routines to enforce the AD objects' syntax and semantics automatically and also threat modeling automation functions to process the object (e.g., create sets, trees, chains, and maps). We will open source `ADF` with a permissive license.

We show how to use ADF in an ORSHIN-related scenario where we are asked to develop a new secure and privacy-preserving cryptowallet using the ORSHIN TLC. We set seven abstract requirements (AR) on the cryptowallet, including secure and authenticated communication, resilience against side channel, fault injection, speculative execution, chip-level attacks, and a trustable life cycle. These ARs cover ORSHIN's WP2, WP3, WP4, and WP5. Then, we explain, based on our recent experience with ADF, how to create ADs according to the threat domain and use them to address the ARs. Our explanation provides high-level recommendations that we collected from the experiments described in the next paragraph.

We empirically evaluated ADF with seven case studies related to the cryptowallet scenario discussed before, and we report remarkable results. We asked several ORSHIN members to focus on a class of threats in their expertise domain, create a catalog of ADs and use the catalog for threat modeling. Our evaluation generated novel insights, 169 high-quality ADs developed by domain experts, covered a broad set of high- and low-level threats related to WP2 (TLC), WP3 (hardware, software, and firmware), WP4 (firmware, hardware), and WP5 (protocol, firmware, and software), Moreover, our evaluation provided valuable feedback to improve ADF further.

Some of ORSHIN's industrial members are already benefiting from ADF's novel and useful features. Texplained (TXP) created a repository of attacks and defenses related to invasive physical attacks on chips, which has never been done before. Security Patterns (SEC) translated a state-of-the-art standard for IoT secure development (i.e., ISA/IEC 62443-4-1) into ADs and incorporated them into their threat modeling process. NXP employed the ADF to model their pre-silicon security auditing on an existing RISC-V secure core (i.e., CV32E40S).

Table of Content

List of Figures

List of Tables

Listings

List of Abbreviations

Abbr.	Meaning
AD	AttackDefense
ADF	AttackDefense Framework
APT	Advanced Persistent Threat
ARX	Abstract Requirement X
ATree	Attack trees
BIAS	Bluetooth Impersonation AttackS
BC	Bluetooth Classic
BLE	Bluetooth Low Energy
BM	Bluetooth Mesh
BoF	Buffer overflow
CAPEC	Common Attack Pattern Enumerations and Classifications
CIA	Confidentiality Integrity Availability
CISA	Cybersecurity and Infrastructure Security Agency
COSIC	Computer Security and Industrial Cryptography
CKC	Cyber Kill Chain
CPU	Central Processing Unit
CTKD	Cross-Transport Key Derivation
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
D2.X	Deliverable 2.X
DFD	Data Flow Diagram
DistriNet	Distributed Systems and Computer Networks
DREAD	Damage, Reproducibility, Exploitability, Affected users, and Discoverability
DoS	Denial of Service
EC	European Commission
ECC	Elliptic Curve Cryptography
ECM	EURECOM
ENISA	European Union Agency for Cybersecurity
EoP	Elevation of Privilege
FIB	Focused Ion Beam
EOP	Elevation of Privilege
FI	Fault Injection
FIDO	Fast IDentity Online
GATT	Generic ATTribute
GPT-4	Generative Pre-trained Transformer 4
IAPP	International Association of Privacy Professionals

Abbr.	Meaning
IC	Integrated Circuit
ID	Information Disclosure
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
ISA	International Society for Automation
IoC	Indicators of Compromise
JSON	JavaScript Object Notation
KASAN	Kernel Address Sanitizer
KASLR	Kernel Address Space Layout Randomization
KMSAN	Kernel Memory Sanitizer
KNOB	Key Negotiation of Bluetooth
KUL	KU Leuven
LINDDUN	Linkability, Identifiability, Non repudiation, Detectability, ID, Unawareness, Non compliance
LLM	Large Language Model
LTK	Long Term Key
MAL	Meta Attack Language
MIST	Malware Information Sharing Platform
MIT	Massachusetts Institute of Technology
MitM	Man in the Middle
NFC	Near Field Communication
NINO	No Input No Output
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
NYC3	New York City Cyber Command
Npm	Node package manager
ORSHIN	Open-source ReSilient Hardware and software for Internet of thiNgs
OSH	Open Source Hardware
OSS	Open Source Software
OTM	Open Threat Model
OWASP	Open Worldwide Application Security Project
OpenCTI	Open Cyber Threat Intelligence
PDF	Portable Document Format
PDU	Protocol Data Unit
PHY	PHYsical layer
PIE	Position Independent Executable
PMD	Predictability, Manageability, Dissassociability
PaLM	Pathway Language Model
PyPI	Python Package Index
RA	Risk Assessment
RAM	Random Access Memory
RCE	Remote Code Execution
RFCOMM	Radio frequency communication
RISC-V	Risk Five
ROM	Read Only Memory
RSA	Rivest Shamir Adleman
RTMP	Rapid Threat Modeling Prototyping

Abbr.	Meaning
RTOS	Real Time Operating System
RX	Requirement X
SC	Side Channel
SCO	Secure Connections Only
SD	Sequence Diagram
SDLC	Secure Development Life Cycle
SE	Secure Element
SEC	Security Patterns
SoC	System on Chip
SMP	Security Manager Protocol
SSP	Secure Simple Pairing
SPA	Simple Power Analysis
STIX	Structured Threat Information Expression
STRIDE	Spoofing, Tampering, Repudiation, ID, DoS, EoP
S&P	Security&Privacy
SoK	Systematisation of Knowledge
T2.X	Task 2.X
TARA	Threat Agent Risk Assessment
TI	Threat Identification
TLC	Trusted Life Cycle
TLS	Transport Layer Security
TM	Threat Modeling
TMT	Threat Modeling Tool
TOML	Tom's Obvious, Minimal Language
TRAM	Threat Report ATT&CK Mapper
TXP	Texplained
UIT	Unlinkability, Intervenability, Transparency
US	United States
USB	Universal Serial Bus
VFI	Voltage Fault Injection
WP	Work Package
XML	Extensible Markup Language
YAML	Yet Another Markup Language

Introduction

ORSHIN's Deliverable 2.2 (D2.2) relates to Task 2.2 (T2.2) which focuses on *two* goals: (i) developing a framework to map abstract security and privacy requirements to concrete policies for the ORSHIN trusted life cycle (TLC) and its device, (ii) use the framework to drive research on ORSHIN's technical works packages, i.e., WP3 (Models for formal verification), WP4 (Effective security audits), and WP5 (Secure auth and comms).

To tackle Task 2.2 we start from *Threat Modeling (TM)* which allows to systematically list, prioritize, and address digital threats [106, 116, 122]. Case studies demonstrated that TM provides tangible security benefits compared to other more common, but potentially less effective, practices, such as compliance with security standards [48, 83, 118]. TM is usually employed to analyze the security and privacy of software systems, but we want to push its boundary to ORSHIN-related domains, including the TLC from Task 2.1, hardware, hardware-software interface, and firmware.

In Chapter 1 we review state-of-the-art TM methodologies, tools, and catalogs. As shown on the left side of Figure 1, TM has *four* main steps: (1) system and attacker modeling, (2) threat identification (3) scoring of threat risk/severity and (4) elaboration of a defense plan. We introduce the STRIDE, LINDDUN, and ATree threat identification techniques focusing respectively on software security, privacy, and attacker goals. Moreover, we introduce pytm, a Python tool to automate threat modeling in specific use cases (e.g., enterprise network security), and CAPEC, CWE, and CVE which are three popular and useful TM catalogs.

In Chapter 2 we present the design of the **AttackDefense Framework (ADF)**, a TM framework we developed to achieve Task 2.2's two main goals. We present seven requirements (i.e., R1, ..., R7) for ADF, some of which are novel. For example, no current TM framework models security and privacy attacks and defenses on a process (e.g., ORSHIN TLC), and a device (e.g., open-source cryptowallet). To address our requirements we introduce the **AD object** a data structure to represent attacks and defenses in a compact, versatile, and useful way (e.g., file with YAML objects). An AD object has a unique name, six primary fields, and is extendible with optional fields.

Moreover, in Chapter 2, we show how to *classify* AD objects (ADs) and collect them in *flat* and *hierarchical* ways. We define four AD categories to concisely classify an AD, e.g., Security AD vs. Privacy AD or Hardware AD vs. Software AD. We show how to collect ADs in sets (e.g., by attack surface), map them to standard threat taxonomies (e.g., CIA, and STRIDE), construct trees of ADs (e.g., by attack surface and sub-surface), ADs chains (e.g., multi-step exploit chain) and ADs wordclouds. Finally, we explain how the ADF complements STRIDE, LINDDUN, and ATree which alone are not capable of satisfying our design requirements.

As shown in Figure 1, the ADF is *not* replacing the four TM phases, but is *enhancing* them by providing extra information and automation. For instance, ADF provides new and valuable threat

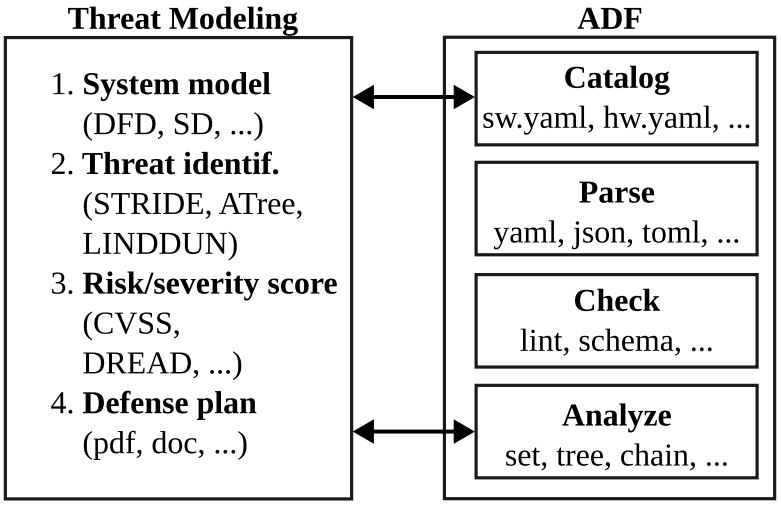

Figure 1: High-level overview of the AttackDefense Framework (ADF). Catalog collects the ADs (e.g., Software, Hardware, Product, Process ADs). Parser semi-automatically generates ADs from different file formats (e.g., YAML, JSON, TOML, and XML). Checker automatically enforces the syntax and semantics of the ADs. Analyze automatically processes the ADs (e.g., sets, maps, trees, and chains). ADF is useful in all Threat Modeling phases (i.e., System model, Threat identification, Risk scoring, and Defense plan)

catalogs about threats related to hardware, firmware, and processes not covered by current TM methodologies. Or, extra machine- and human-friendly functionalities, including threats' filtering, mapping, and chaining, that are missing from open source TM tools.

In Chapter 3 we present `ADF`, a toolkit implementing the ADF. The toolkit has four modules. (1) Catalog contains the ADs that we develop in our case studies, (2) Parse is capable of extracting ADs from YAML, JSON, TOML, and XML files and can be easily extended to parse other file types, (3) Check automatically validates the syntax, semantics, and content of the ADs using a combination of checkers such as yamllint and Python schema, (4) Analyze provides useful functions to automatically process ADS to generate, among others, ADs' sets, maps, trees, wordclouds, and chains. We will open-source our toolkit with a permissive license to let other individuals take advantage of ADF and provide feedback.

In Chapter 4 we explain how to use the ADF assuming an ORSHIN-related scenario where we are tasked to develop a secure and privacy-preserving cryptowallet. The cryptowallet includes security-critical hardware (e.g., microcontroller and secure element), software (e.g., Linux) and communication (e.g., BLE, and FIDO) components and is developed using the ORSHIN TLC. We set seven orthogonal and relevant abstract requirements (i.e., AR1, . . . , AR7) covering the TLC and the cryptowallet's hardware, software, firmware, and communication capabilities. Based on our current experience with the ADF, we discuss on a high-level how to create and use ADs catalogs to address these requirements.

In Chapter 5 we describe the results of seven case studies we run to test the ADF in domains related to the Chapter 4. Our evaluation covers WP2 (TLC), WP3 (side channel, fault injection, and speculative execution attacks), WP4 (presilicon security auditing and invasive physical attacks), and WP5 (protocol and implementation-level attacks on communication technologies). We involved experts in these domains from our consortium (i.e., ECM, KUL, NXP, SEC, and TXP). We developed seven ADs catalogs, for a total of 169 ADs. Our case studies not only empirically confirm that ADF successfully addresses Task 2.2. But, it also generated useful feedback to fur-

ther improve ADF. For example, we are already experimenting with better ways to create and use ADs with different levels of abstraction.

The Related Work Chapter presents relevant works covering TM methodologies including vendor-specific ones, gamification, real-world examples, automation tools, and domain-specific extensions. Furthermore, we cover related work from threat intelligence and process security domains.

We wrap up the deliverable with the Conclusion Chapter. We summarize ADF and its data model based on the AD object. We explain why and how ADF addresses Task 2.2's two main goals. We show the synergies of the contributions between Task 2.1 and 2.2., especially the TLC and the ADF. We report the added value generated by the ADF for industrial and academic members of the ORSHIN consortium. We conclude with our plans for future work on the ADF.

Chapter 1

Background

In this Chapter, we provide the relevant background about threat modeling (TM). We describe the four key questions related to TM and how they map to different methodologies (e.g., system models, threat identification, and scoring). We describe the STRIDE, LINDDUN, and ATree (attack trees) threat identification and the CVSS and DREAD scoring techniques. We present Pytm, a Python framework to automate and aid TM. Finally, we introduce CAPEC, CWE, and CVE, the most famous threat catalogs typically used to aid TM.

1.1 Threat Modeling

In its simplest top-down view, threat modeling (TM) must answer *four* questions [132]:

1. Q1: What are we working on?
2. Q2: What can go wrong?
3. Q3: What are we going to do about it?
4. Q4: Did we do a good enough job?

and we can answer these questions in four steps as shown in the left part of Figure 1.

1.1.1 Q1: What are we working on?

First, we build a *system model* including the system's components, interconnections, and security boundaries. Software systems are typically modeled with a *data flow diagram (DFD)*, while protocols with *sequence diagrams (SD)*. As shown on the left side of Figure 1.1, a DFD represents with solid lines the system's components, with dotted red lines the trust boundaries, and with numbered arrows how the data flows across components and boundaries. Instead, as depicted on the right side of Figure 1.1, an SD represents horizontally the parties involved in a protocol and vertically from top to bottom the messages that they exchange.

1.1.2 Q2: What can go wrong?

Then, we need to *identify threats* (i.e., exploitable vulnerabilities) on our system model. We define our attack surface which is the set of components that we want to protect (e.g., the web server). Then, we consider different attacker models targeting our surface (e.g., remote attacks on a web

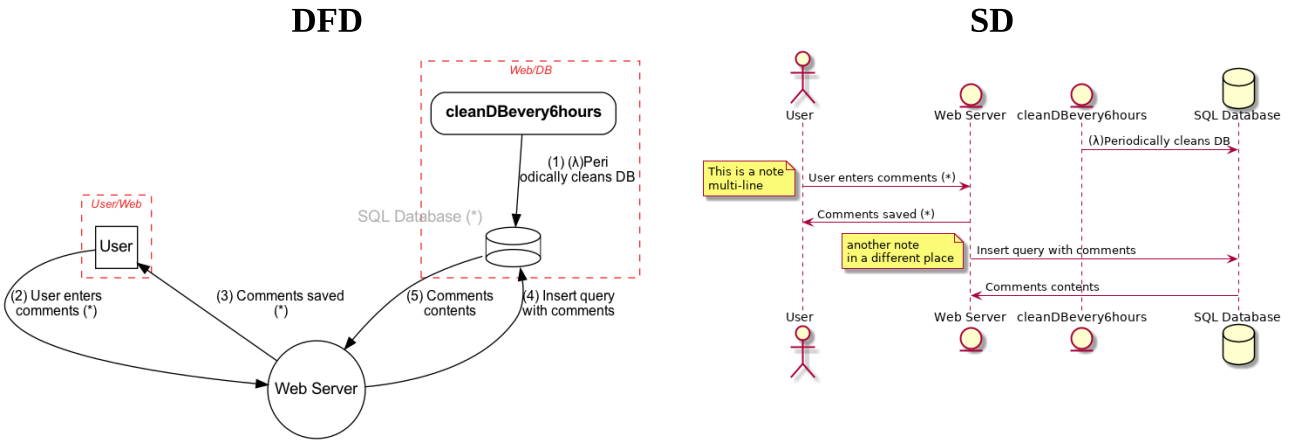

Figure 1.1: DFD and SD system models for TM. DFD on the left, and SD on the right. Both are modeling a cloud deployment with a user, a web application, and a database server. The models are generated using pytm [121]

application). Each adversary model has certain goals (e.g., leaking sensitive data from the web application) and attack vectors (e.g., SQL injection on the database server).

Threat identification (TI) is a laborious and mostly manual process. The TI methodologies differentiate according to the threat domains (e.g., security, privacy) and their relation (e.g., a flat list, and hierarchical trees). Now we introduce *STRIDE* [69] (software security), *LINDDUN* [22] (system privacy), and *Attack Trees*, shortened to *ATree* [108] (attacker goals) which are the most popular TI methodologies. For a discussion of other relevant TI (and TM) methodologies refer to [113, 112].

STRIDE STRIDE was developed by Kohnfelder et al. in 1999 and adopted by Microsoft in 2002 [116] as part of its Secure Development Life Cycle (SDLC) [68] and Threat Modeling Tool (TMT) [70]. STRIDE focuses on identifying threats violating software security, with an emphasis on networked systems (e.g., web and cloud applications). Specifically, it covers Spoofing (i.e., lack of authentication), Tampering, Repudiation, Information disclosure (e.g., data breaches), Denial of service, and Elevation of privilege threats. A STRIDE user is tasked to take a DFD (or other system models) and for each element in the attack surface list all possible threats in each STRIDE category. The threat listing can be semi-automated using an attack library. Microsoft has documented its STRIDE threat modeling approach since 1999 and provided some useful lessons learned such as the lack of threat modeling training, complexity in real-world scenarios, and the importance of the people factor [114].

LINDDUN LINDDUN is a privacy-focused TI methodology developed by Deng et al in 2010. LINDDUN complements STRIDE as it uses the same reference system model (i.e., a DFD) but produces a list of privacy threats other than software security ones. Specifically, LINDDUN targets seven threat classes: Linkability, Identifiability, Non-repudiation, Detection, Data disclosure, Unawareness, and Non-Compliance. Note that Data (Information) disclosure and Non repudiation overlap with STRIDE, hence the two might produce overlapping threats. The LINDDUN developers also provide a reference LINDDUN threat catalog extracted from empirical experiments [134] and LINDDUN GO, a lightweight LINDDUN version, especially useful for newcomers [135].

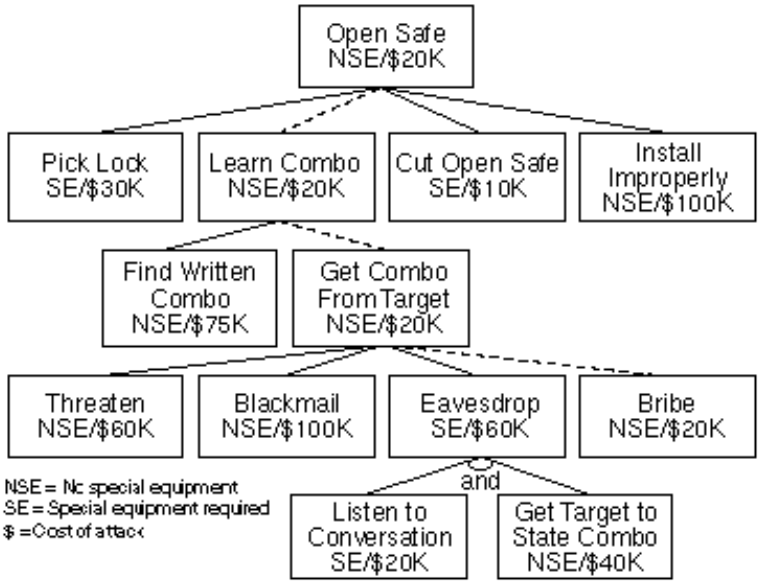

Figure 1.2: ATree open safe example. Taken from [108]

ATree ATree is a TI methodology proposed by Schneier in 1999. Each attack tree models a specific attacker's goal that is represented as the tree's root. The sub-trees are the attacker's sub-goals and can be logically linked (e.g., sub-goal1 AND/OR sub-goal2) and annotated (e.g., sub-goal feasibility, requirements, and monetary cost). For instance in Figure 1.2, we show an annotated attack tree modeling how to open a safe (root). Each sub-goal (sub-tree) is annotated with its cost and traversing the tree provides the full cost of the attack. Note that two leaves are in an AND relation (e.g., to eavesdrop the attacker has to listen to the conversation AND get the safe combination). The most effective attack strategy traverses the tree using the dotted lines, it does not require special equipment (NSE) and is worth 80K USD.

ATree differs from STRIDE and LINDDUN in several important ways. They focus on goals other than threat classes, they produce a hierarchical representation of threats rather than a (flat) list, and they are more scalable, as a single tree can address multiple threats. However, they are more difficult to create and maintain because of their hierarchical structure. For example, by missing just one sub-goal (sub-tree) a tree might become useless. Various attempts have been made to semi-automate ATree's generation. For example, the Meta Attack Language (MAL) can be used to design domain-specific languages to semi-automatically generate and analyze large attack trees [57].

1.1.3 Q3: What are we going to do about it?

The third step (left part of Figure 1 is *risk/severity scoring*. As for TI, there are many ways to compute these scores that are based on the attacker's goal, impact, and cost. There are two popular scoring schemes called CVSS and DREAD.

CVSS NIST's CVSS (common vulnerability scoring system) [89] is the most used severity scoring scheme. It is employed by the US National Vulnerability Database (NVD) to score the severity of all currently known vulnerabilities. There are two versions of CVSS: v2.0 and v3.x. Both versions use three metric groups: Base, Temporal, and Environmental. The Base metric provides a score from zero to ten and can be adjusted by scoring the Temporal and Environmental met-

rics. The Temporal metric model factors that are affecting the severity of a threat over time, while the Environmental enables to re-weight the severity according to the target. The Base scores are provided by the NVD, e.g., Low, Medium, High for v2.0. The Temporal and Environmental scores can be computed using the NVD calculators web pages [90, 91]. Recently, CVSS v4.0 was released but its adoption is not widespread [92].

DREAD Microsoft's DREAD is a software threats scoring system focusing on Damage (e.g., loss of data), Reproducibility (e.g., deterministic threat), Exploitability (e.g., attacker model strength), Affected users (e.g., scale), and Discoverability (e.g., ease of threat identification) [71]. The DREAD final score is a number from one to ten obtained by computing the arithmetic mean of five sub-scores from one to ten. Despite being discontinued by Microsoft, DREAD is still used by TM practitioners [122] and researchers [14].

1.1.4 Q4: Did we do a good enough job?

The fourth phase of TM starts with the creation of a *defense plan*. This task overlaps with Q3 but we report it in Q4 to separate it from risk scoring. A defense plan indicates a list of countermeasures or fixes based on the threats produced in the previous phase. Typically, the defense plan is a PDF document containing a summary of the first three phases and a section explaining which threats are fixed, mitigated, and accepted as risks.

The fourth phase continues with the delivery, management, and refinement of the defense plan. The defense plan's mitigations should be shipped to production and the TM system should be monitored for incidents (e.g., via secure logging functions). The defense plan should be periodically revisited and if needed updated according to multiple feedback channels, including the security logs, bug bounties, international cybersecurity advisors, and user feedback.

1.2 pytm: A Pythonic framework for threat modeling

pytm is a Python-based framework developed by OWASP. Unlike traditional graph-based approaches, pytm introduces a novel concept called "threat modeling as a code", which allows for the integration of threat modeling within the development process and aims to enhance the usability of the overall threat modeling process.

The framework comes with some pre-defined and extendible classes that are used to construct a system model. In particular, the user writes a script instantiating Python objects for each system component and specifies how the components are connected. Once the system model is completed, the tool generates a visual system diagram (e.g., DFD or SQ) and automatically identify potential threats using a threat library.

pytm's threat library incorporates approximately 100 entries sourced from CVE, CWE, and CAPEC databases, and provides detailed information about each potential threat. Notable attributes within the threat library include the *condition* attribute for threat identification, the *likelihood* and *severity* attributes for simple evaluation of the threats, and the *mitigation* attribute, which suggests countermeasures to address the identified threats. A selection of pytm threats is covered in Table 1.1.

The ability to add controls to mitigate threats within the model empowers users to apply countermeasures, generate an updated model, and subsequently update the threat model by

no longer considering the threats mitigated by the applied countermeasures. This approach aligns with the continuous threat modeling paradigm, where the process is conducted continuously rather than as a one-time assessment.

Table 1.1: Selection of pytm threats

ID	Threat description
INP01	Buffer Overflow via Environment Variables
INP02	Overflow Buffers
INP03	Server Side Include (SSI) Injection
CR01	Session Sidejacking
INP04	HTTP Request Splitting
CR02	Cross Site Tracing
INP05	Command Line Execution through SQL Injection
INP06	SQL Injection through SOAP Parameter Tampering
SC01	JSON Hijacking (aka JavaScript Hijacking)
LB01	API Manipulation
AA01	Authentication Abuse/ByPass
DS01	Excavation
DE01	Interception
DE02	Double Encoding
API01	Exploit Test APIs
AC01	Privilege Abuse
INP07	Buffer Manipulation
AC02	Shared Data Manipulation
DO01	Flooding
HA01	Path Traversal
AC03	Subverting Environment Variable Values
DO02	Excessive Allocation
DS02	Try All Common Switches
INP08	Format String Injection
INP09	LDAP Injection
INP10	Parameter Injection
INP11	Relative Path Traversal
INP12	Client-side Injection-induced Buffer Overflow
AC04	XML Schema Poisoning
DO03	XML Ping of the Death
AC05	Content Spoofing
INP13	Command Delimiters
INP14	Input Data Manipulation
DE03	Sniffing Attacks
CR03	Dictionary-based Password Attack
API02	Exploit Script-Based APIs
HA02	White Box Reverse Engineering
DS03	Footprinting
AC06	Using Malicious Files
HA03	Web Application Fingerprinting
SC02	XSS Targeting Non-Script Elements
AC07	Exploiting Incorrectly Configured Access Control Security Levels

ID	Threat description
INP15	IMAP/SMTP Command Injection
HA04	Reverse Engineering
SC03	Embedding Scripts within Scripts
INP16	PHP Remote File Inclusion
AA02	Principal Spoof
CR04	Session Credential Falsification through Forging
DO04	XML Entity Expansion
DS04	XSS Targeting Error Pages
SC04	XSS Using Alternate Syntax
CR05	Encryption Brute Forcing
AC08	Manipulate Registry Information
DS05	Lifting Sensitive Data Embedded in Cache
SC05	Removing Important Client Functionality
INP17	XSS Using MIME Type Mismatch
AA03	Exploitation of Trusted Credentials
AC09	Functionality Misuse
INP18	Fuzzing and observing application log data/errors for application mapping
CR06	Communication Channel Manipulation
AC10	Exploiting Incorrectly Configured SSL
CR07	XML Routing Detour Attacks
AA04	Exploiting Trust in Client
CR08	Client-Server Protocol Manipulation
INP19	XML External Entities Blowup
INP20	iFrame Overlay
AC11	Session Credential Falsification through Manipulation
INP21	DTD Injection
INP22	XML Attribute Blowup
INP23	File Content Injection
DO05	XML Nested Payloads
AC12	Privilege Escalation
AC13	Hijacking a privileged process
AC14	Catching exception throw/signal from privileged block
INP24	Filter Failure through Buffer Overflow
INP25	Resource Injection
INP26	Code Injection
INP27	XSS Targeting HTML Attributes
INP28	XSS Targeting URI Placeholders
INP29	XSS Using Doubled Characters
INP30	XSS Using Invalid Characters
INP31	Command Injection
INP32	XML Injection
INP33	Remote Code Inclusion
INP34	SOAP Array Overflow
INP35	Leverage Alternate Encoding
DE04	Audit Log Manipulation

Figure 1.3: MITRE's CAPEC, CWE, and CVE linked threat catalogs

1.3 Threat Catalogs

TM can be aided by a threat catalog, which is a collection of known threats represented using a data format (e.g., XML). There are several catalogs of interests. Here we describe MITRE's CAPEC [78], CWE [80] and CVE [79] which are three popular and linked collections of attack techniques, weaknesses, and vulnerabilities. Their logos are depicted in Figure 1.3.

CAPEC Mitre's CAPEC (Common Attack Pattern Enumerations and Classifications) contains attack patterns extracted from real-world threats. An attack pattern describes the adversary's approach to exploit known weaknesses and the challenges that they might face. Each CAPEC entry has the following fields: unique ID, in the form `CAPEC-nnn` (where `n` is a numeric digit), description, likelihood, severity, relationship, execution flow, prerequisites, skills required, consequences, mitigations, related weaknesses (CWE), taxonomy mappings, and content history. At the time of writing (2023-06-02) there are 559 CAPEC entries. For example, CAPEC-668 [77] models our prior research on the KNOB Bluetooth attacks [9, 7], while CAPEC-667 [76] models the BIAS ones [6].

CWE Mitre's CWE (Common Weakness Enumeration) collects known software and hardware weakness types in a taxonomy. Weakness is defined as a software, firmware, hardware, or service component flaws that could enable a vulnerability given certain assumptions. Each CWE

CWE-1189	Improper Isolation of Shared Resources on System-on-a-Chip (SoC)
CWE-1191	On-Chip Debug and Test Interface With Improper Access Control
CWE-1231	Improper Prevention of Lock Bit Modification
CWE-1233	Security-Sensitive Hardware Controls with Missing Lock Bit Protection
CWE-1240	Use of a Cryptographic Primitive with a Risky Implementation
CWE-1244	Internal Asset Exposed to Unsafe Debug Access Level or State
CWE-1256	Improper Restriction of Software Interfaces to Hardware Features
CWE-1260	Improper Handling of Overlap Between Protected Memory Ranges
CWE-1272	Sensitive Information Uncleared Before Debug/Power State Transition
CWE-1274	Improper Access Control for Volatile Memory Containing Boot Code
CWE-1277	Firmware Not Updateable
CWE-1300	Improper Protection of Physical Side Channels

Figure 1.4: 2021 CWE Most Important Hardware Weaknesses from [72]

has the following fields: unique ID, in the form of `CWE-nnnn`, description, relationship, modes of introduction, consequences, demonstrative examples, observed examples (CVE), membership, notes, taxonomy mappings, related attack patterns (CAPEC), references, content history. Currently, there are 933 CWE entries. Related to ORSHIN, in 2021 MITRE released the list of most important *hardware weaknesses* [72] which includes chips' open debug ports and improper isolation of components (see Figure 1.4 for the full list with related IDs). MITRE also maintains a list of top 25 CWE related to software [73].

CVE Mitre's CVE (Common Vulnerabilities and Exposures) is a standard format to store, discover, analyze and correlate vulnerabilities. Each CVE has these entries: unique ID, in the form of `CVE-yyyy-nnnnn`, description, references, assigner, creation record, and other legacy fields. At the time of writing, there are 204528 CVE records in the NVD database (also known as CVE List). This is a staggering number that has been exponentially growing in the last three years. Related to the previous CAPEC example covering our prior Bluetooth research work, the KNOB vulnerabilities are tracked with CVE-2019-9506 [93], while the BIAS ones with CVE-2020-10135 [94].

Chapter 2

ADF Design

This chapter presents the design of the proposed ADF. We outline the *seven* key requirements in Section 2.1 that the framework aims to fulfill. To address these requirements, we introduce the AD in Section 2.2 as a fundamental building block of the framework. We explain the AD fields, types, and their use in flat and hierarchical arrangements. Furthermore, we highlight how the AD objects enhance and extend TM with STRIDE, LINDDUN, and ATree.

2.1 Requirements

Our threat modeling framework should address key requirements to satisfy the goals we set in Task 2.2, i.e., set security and privacy requirements on the TLC and its related device. We need a framework that integrates attacks and defenses, considers security and privacy trade-offs, and covers hardware and firmware threats. Moreover, it should incorporate product and process (i.e., life cycle) aspects, include fine-grained and coarse-grained threats, and offer usability for machines and humans.

R1: Attacks and Defenses Our objective is to integrate attacks (threats) and defenses (mitigations) into the threat modeling process to enhance the understanding of threats beyond the traditional focus on the attacker. The threat modeling framework can provide a more comprehensive context around threats by considering attacks and defenses. This approach enables listing fine-grained and coarse-grained threat mitigation strategies, identifying critical attacks with(out) known defenses, exploring alternative defensive strategies for a specific attack, evaluating defense-in-depth options, and determining the minimum number of defenses required to address a set of threats.

R2: Security and Privacy Addressing security and privacy (S&P) threats is of utmost importance in the threat modeling process. Considering the trade-offs between security and privacy is critical, as they are intrinsically intertwined. However, existing approaches often treat security and privacy separately, leading to overlooked trade-offs. It is essential to integrate the analysis of security and privacy aspects jointly. For instance, we could model a scenario to explore the competing goals of confidentiality and integrity (security) vs. repudiability and traceability (privacy). By considering security and privacy in tandem, the threat modeling framework can provide a comprehensive understanding of the potential trade-offs and their implications.

R3: Hardware and Firmware There is a need to broaden the scope of threat modeling to cover hardware and firmware threats, which are relevant but often neglected areas in the threat modeling process. We will cover traditional threat modeling areas like network and software security. However, fulfilling this requirement, we aim to extend the framework coverage to novel domains, including invasive and non-invasive physical attacks targeting hardware and firmware. Furthermore, we want to address threats at the intersection of hardware and software, such as side-channel attacks, fault injection attacks, and micro-architectural threats.

R4: Product and Process Our objective is to incorporate both the threat modeling of a specific product, such as a cryptowallet, and the process involved in its development and maintenance, such as ORSHIN's Trusted Life Cycle (TLC) outlined in D2.1. Presently, existing threat modeling frameworks only focus on analyzing the product itself, often overlooking the consideration of the underlying process. However, we can significantly enhance our coverage of unknown but relevant threat classes by addressing this gap. For instance, this approach enables us to defend against hardware and software supply chain attacks (e.g., SolarWinds [131] and Supermicro [109]).

R5: Fine- and Coarse-grained We aim to include fine-grained and coarse-grained threats in our threat modeling process. Fine-grained threats encompass real-world attack and defense instances, such as a disclosed buffer overflow targeting our cryptowallet. On the other hand, coarse-grained threats refer to generic attack techniques, such as the buffer overflow attack class. By integrating threats at different levels of abstraction, we can significantly enhance our threat modeling analysis. For instance, we can automatically generate hierarchies of threats, including trees, chains, and graphs. Furthermore, we can create collections of threats using various representations such as tables, sets, and maps. These flat and hierarchical representations can be constructed using a generic filter, such as a specific attack surface, vector, or attacker model.

R6: Reusable and Updatable Our framework should be reusable as much as possible to prevent the duplication of threat modeling exercises, even across different teams and organizations. Additionally, the framework must be updatable, allowing new attacks and defenses to be incorporated as they become available. We aim to consistently and incrementally add threats over time, such as when an old threat is patched or reintroduced, or new threats are identified. Similarly, we want to cover threats in different contexts, such as when a new feature is introduced in a product, which may introduce potential new threats. For instance, when we update the firmware of our cryptowallet to address the previously mentioned buffer overflow (BoF) vulnerability, we desire the ability to reuse the existing threat model and incorporate new BoF threats associated with the latest firmware version.

R7: Machine and Human friendly We seek a framework that offers seamless usability for machines and humans, minimizing friction. By employing languages and tools that machines and humans understand, we can enhance the framework's effectiveness. Humans should be able to read, write, analyze, and share attack and defense strategies without any hindrances. Moreover, the framework should accommodate users with varying expertise in threat modeling, including developers, DevSecOps professionals, and threat modeling experts. Simultaneously, the framework should enable machines to process these attacks and defenses, generating valuable outputs such as interactive and portable reports and visualizations. It should also facilitate intelligent storage of these outputs, leveraging techniques such as version control, CI/CD pipelines, standardized formats, interoperability, and machine-checkable data formats.

Listing 2.1: AttackDefense (AD) Object written in YAML

```yaml
ad_name:
  # Primary fields
  a: attack
  d:
    policy1: [mech1, mech2]
    policy2: [mech1, mech2]
    ...
  surf: [surf, subsurf, subsubsurf, ...]
  vect: [vector1, vector2, ...]
  model: [model1, model2, ...]
  tag: [tag1, tag2, ...]
  # Optional fields
  risk: [score1, score2, ...]
  year: 2023
  cve: ["123", "456", ...]
  cwe: ["123", "456", ...]
  capec: ["123", "456", ...]
  vref: ["vendor-ref1", ...]
  ...: ...
```

2.2 AttackDefense (AD) Object

To fulfill the seven requirements outlined in Section 2.1, we introduce the **AttackDefense (AD) object** as a fundamental component of the AttackDefense Framework (ADF). Hereafter, we refer to a single AD object as *AD*, while we use *ADs* for multiple objects. Each AD object contains information about an attack and its associated defenses, effectively addressing the objectives of integrating attacks and defenses (R1) and modeling various threat domains, including security, privacy, hardware, firmware, product, and process (R2, R3, R4). Furthermore, the AD object facilitates threat modeling at different levels of granularity (R5) across different temporal and spatial contexts (R6). Finally, the AD object supports serialization in multiple languages while mapping to a standardized Python dictionary format that ensures portability, interoperability, and ease of storage or processing, satisfying the requirements for usability by both machines and humans (R7).

Listing 2.1 illustrates the creation of an AD object using the YAML (Yet Another Markup Language) serialization language, which is both human and machine-readable. Although ADs can be constructed using other serialization languages such as TOML (Tom's Obvious, Minimal Language) and JSON (JavaScript Object Notation), we opt for YAML due to its enhanced readability. Each AD object is represented as a YAML object and consists of a unique name (`ad_name`), primary fields (six), and optional fields. These fields are structured as key-value pairs and can accommodate various data types, including dictionaries, lists, strings, and integers.

The AD has six primary fields:

- `a` contains a string describing an attack with an arbitrary level of abstraction (e.g., attack instance or attack class).
- `d` stores a dict of sub-dicts to model different defense strategies for an attack. In particular, each sub-dict encodes a high-level policy string (e.g., `policy1`) and a list of concrete mechanisms strings (e.g., `[mech1, mech2]`) to satisfy the policy. The sub-dicts can be ordered according to some criteria (e.g., from the most effective to the least effective defense).

- `surf` is an ordered list of strings describing the attack surface (i.e., target). The list is ordered such that each element *narrows down* the attack surface from the broadest to the most specific (later we explain such ordering with a concrete example).
- `vect` is a list of strings containing the attack vectors (i.e., techniques) related to `a`.
- `model` stores the adversary models capable of performing `a` in a list of strings.
- `tag` is a list of strings storing useful metadata, such as the AD type, security-privacy trade-offs, and other technicalities.

The AD supports optional fields to *augment* its effectiveness. An ADF user can add whatever field, next we present some examples that we employed:

- `risk` is a list of strings storing risk scores associated to `a`, including CVSS v3 and v2.
- `year` is an int storing the year when `a` was first discovered or demonstrated.
- `cve`, `cwe`, and `capec` are lists of strings storing `a`'s common vulnerabilities (CVE), weaknesses (CWE), and attack pattern (CAPEC) identifiers.
- `vref` is a list of vendor reference strings associated to `a`, including security advisories identifiers from Linux [37] and Android [46].

We classify ADs in *four* categories: (1) *Security* or *Privacy* (2) *Product* or *Process* (3) *Hardware* or *Software* or *Firmware* or *Protocol* (4) *Fine-grained* or *Coarse-grained*. As a result, a properly built collection of ADs achieves unprecedented threat coverage. For instance, they can cover high-level and low-level security and privacy attacks and defenses on products' hardware, software, firmware, and protocols, plus their life cycles.

2.3 Flat and Hierarchical ADs

A collection of ADs (i.e., a YAML file) can be (automatically) processed to produce *flat* and *hierarchical* ADs combinations. These combinations are useful, among others, to visualize, adapt, and measure TM coverage.

Flat (set, maps) By filtering ADs based on relevant AD field values, we can create *sets* (unordered lists) of ADs. This allows us to select ADs associated with specific attack surfaces (e.g., Linux) or techniques (e.g., Buffer Overflow). Additionally, we can *map* a collection of ADs to established threat taxonomies such as CIA (Confidentiality, Integrity, Availability), STRIDE, and LINDDUN. For instance, we can create three sets of ADs, each addressing a letter of the CIA acronym. Utilizing basic set operations, we can examine intersections between ADs and merge them as needed.

Hierarchical (chains, trees, wordclouds) Hierarchical collections of ADs can be created to represent ordered relationships. This is particularly relevant for modeling modern exploits that often involve a chain of attacks exploiting multiple vulnerabilities. Such scenarios can be represented using a *chain* (vector) of ADs. For example, the Pegasus Remote Code Execution (RCE) exploit on iOS from 2021 [52] can be modeled using a chain of four ADs, where the first three represent the Trident iOS vulnerabilities CVE-2016-4655, CVE-2016-4656, and CVE-2016-4657 to get root privileges (see boxed ADs in Figure 2.1), and the fourth represents the remote privileged read exploit for iMessage CVE-2019-8646 (see ellipse AD in Figure 2.1).

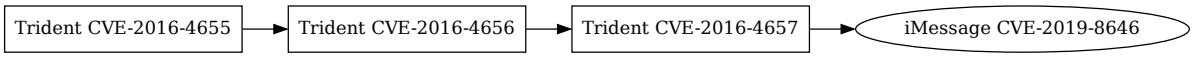

Figure 2.1: iOS Pegasus RCE chain from 2021

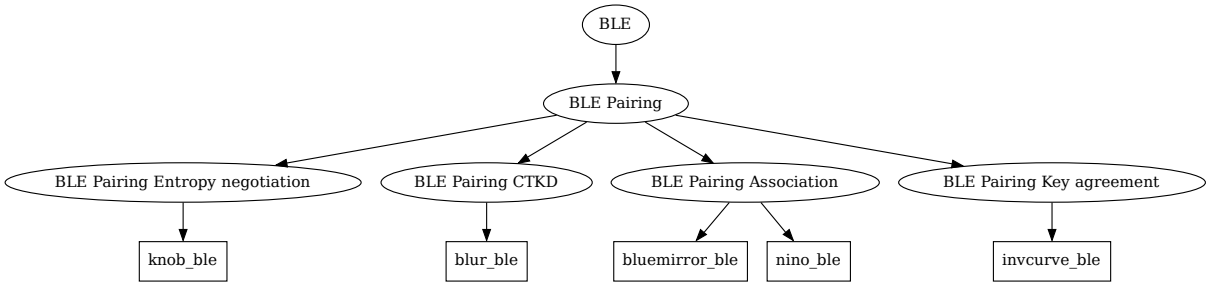

Figure 2.2: ADs attack surface tree for BLE Pairing

Fine-grained and coarse-grained ADs can be visualized using a *tree* structure (directed acyclic graph). This visualization is particularly useful when considering the `surf` field, as it allows the placement of the attack surface as the root of the tree, constructing the tree based on sub-surfaces. For instance, Figure 2.2 depicts a tree of ADs related to protocol-level threats in BLE pairing. Each BLE pairing phase, such as entropy negotiation and CTKD (Cross-Transport Key Derivation), is represented as a sub-tree, with the ADs being the leaves of the tree.

The coverage of a specific AD field can be effectively visualized using *wordclouds* created from ADs. A wordcloud represents a string of text by displaying words in varying sizes, proportional to their frequency within the string. For instance, we can generate a wordcloud (Figure 2.3) to examine the attack surface of a Bluetooth ADs catalog, considering both protocol-level and implementation-level aspects. In the generated wordcloud, the size of each word corresponds to its frequency. Consequently, surface-level words (e.g., BC and BLE) appear larger than sub-surface words (e.g., Pairing, Session), and sub-surface words appear larger than sub-sub-surface words (e.g., Entropy negotiation, CTKD), aligning with our expectations.

Figure 2.3: Wordcloud of the attack surfaces covered by our `bt.yaml` catalog

Table 2.1: Comparison between STRIDE, LINDDUN, ATree, and ADF. Legend: ●: supported, ◐: partially supported, ○: not supported. Overall, ADF complements and extends STRIDE, LINDDUN, and ATree

Requirement	STRIDE	LINDDUN	ATree	**ADF**
R1: Attack & Defense	◐	◐	◐	●
R2: Security & Privacy	◐	◐	●	●
R3: Hardware & Firmware	○	○	●	●
R4: Product & Process	○	○	○	●
R5: Fine-gr. & Coarse-gr.	◐	◐	●	●
R6: Reusable & Updatable	○	○	○	●
R7: Machine-fr. & Human-fr.	◐	◐	◐	●

2.4 Extending STRIDE, LINDDUN and ATree with ADF

Table 2.1 shows how STRIDE, LINDDUN, ATree, and ADF satisfy the seven requirements set in Section 2.1 (i.e., R1, . . . , R7). STRIDE, LINDDUN, and ATree identify either attacks or attacker goals, while ADF put attacks and defenses in context. STRIDE focuses on software security, LINDDUN on privacy, whereas ADF and ATree cover both. STRIDE and LINDDUN cannot handle hardware and firmware attacks, whereas ADF and ATree address these critical areas (especially for ORSHIN). Only ADF covers both the product and its life cycle (e.g., TLC from Task 2.1), whereas STRIDE, LINDDUN, and ATree are only concerned with the product.

Continuing with the comparison in Table 2.1, STRIDE and LINDDUN provide high-level attacks, while ADF and ATree have the capability to represent both high-level and concrete ones and their relations (i.e., Fine-grained and Coarse-grained ADs). STRIDE, LINDDUN, and ATree are known to be hard to reuse and update (even within the same TM team), while ADF is built on the AD object, a reusable, flexible, and updatable data structure. While STRIDE, LINDDUN, and ATree are human-friendly, they are often difficult to process and analyze by machines. In contrast, ADF is designed to be both machine and human-friendly.

We note that a TM practitioner can use ADF *together with* STRIDE, LINDDUN, and ATree as it complements them. For example, the ADF supports by-design mapping functions capable of generating ADs sets based on the STRIDE and LINDDUN taxonomies as detailed in Section 2.2. Our goal is not to replace these TI techniques and their associated methodology. But, to *enhance* TM as a whole by offering better coverage, usability, and automation.

Chapter 3

ADF Implementation

In this Chapter, we describe the relevant details of `ADF`, a toolkit we developed to implement ADF (presented in Chapter 2). Figure 3.1 shows the toolkit's block diagram with its *four* modules: Catalog, Parse, Check, and Analyze and their related libraries. The modules map to the ADF block diagram presented in the left side of Figure 1 and address the seven requirements set in Section 2.1. We will open-source our toolkit with a permissive license (e.g., MIT license) to let others use and improve it. We now describe each toolkit module in detail. The files and folders mentioned below are relative to the `toolkit` folder in our repository.

3.1 Catalog

Catalog contains the developed ADs and is located within a dedicated subfolder in the GitHub repository. These ADs are written in YAML, a language that extends JSON and supports various convenient data types, including integers, strings, lists, dictionaries, and objects. YAML provides features such as auto-indentation, compliance with backward-compatible versioning, and the ability to include comments. It is widely supported by popular programming languages, which offer YAML libraries. Moreover, YAML is developer-friendly, as it has various integrations with developer tools. For instance, the most popular IDEs provide automated YAML completion, linting, checking, and formatting. In the following paragraphs, we present concrete implementations of

Listing 3.1: `knob_ble` AD. Classification: Security, Product, Protocol, Fine-grained

```
knob_ble:
  a: KNOB entropy downgrade attack on BLE pairing
  d:
    Mutually auth entropy negotiation: [Auth entropy with BLE pairing key]
    High key entropy: [Disallow entropy values lower than 16]
  surf: [BLE, Pairing, Entropy negotiation]
  vect: [Entropy downgrade, Key brute force]
  model: [Proximity, MitM]
  tag: [Protocol, SMP]
  risk: [cvss3_high, cvss2_medium]
  year: 2019
  cve: ["9506"]
  cwe: ["310", "327"]
  capec: ["668"]
```

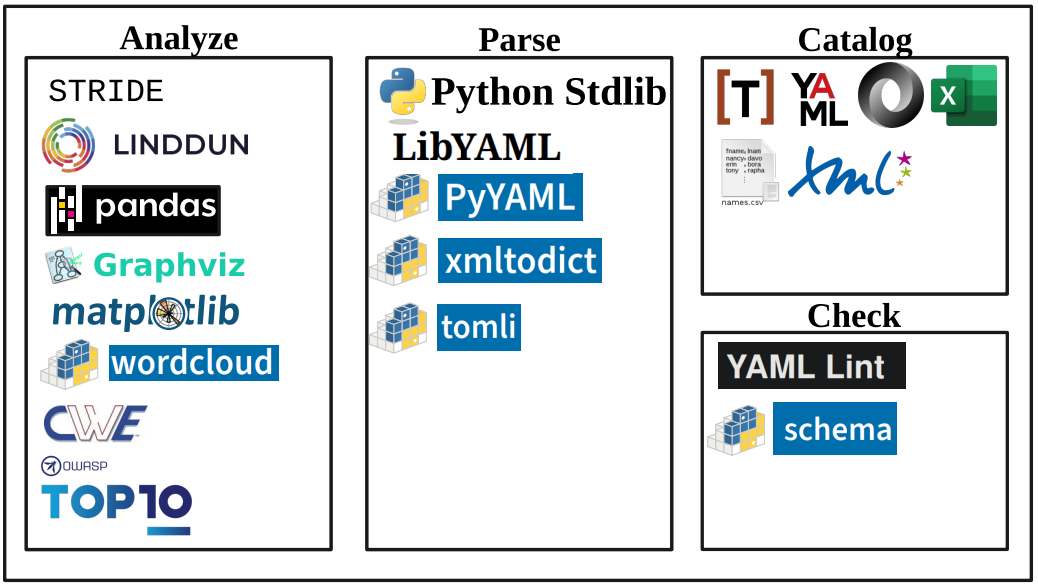

Figure 3.1: `ADF` toolkit block diagram with its four modules: Analyze, Parse, Catalog, and Check and related libraries

YAML ADs from Catalog and classify them based on the four categories introduced in Section 2.2.

Listing 3.1 shows how we implemented `knob_ble`, an AD to model the KNOB attack on BLE [7]. The attack involves an adversary in BLE proximity with the victims (`model`) targeting the entropy negotiation phase of BLE pairing (narrowing down `surf`). The adversary MitM the victims, downgrades the pairing key entropy and brute-forces the key (`vect`). KNOB is effective at the protocol level, attacking the Security Manager Protocol (SMP) (`tag`). Also, the KNOB attack was discovered in 2019 (`year`), it is associated to CVE-2019-9506 (`cve`), CWE-310 and CWE-327 (`cwe`), CAPEC-668 (`capec`), and high CVSSv3 risk and medium CVSSv2 risk (`risk`). We can defend against KNOB either by mutually authenticating the entropy negotiation protocol or forbidding low values for the key entropy (`d`)

One of the unique features of the ADF is its capability to establish security and privacy requirements for a process (e.g., ORSHIN TLC). The implementation of the AD named `sw_orion` presented in Listing 3.2 addresses a severe and known process supply chain attack on the Windows SolarWinds Orion Platform. This attack involves a remote attacker circumventing signature checks to disseminate malicious software updates to a vast number of Windows PCs. This attack can be addressed by properly authenticating software updates with valid certificates.

Another ADF useful feature is the possibility to effortlessly integrate *new* threats into the catalog. Let's assume that in 2030 we are notified about a newly disclosed and critical vulnerability affecting the Linux kernel and we want to include it in our catalog. The new threat enables privileged and proximity-based code execution (PPCE), exploiting a kernel-space stack-based buffer overflow (BoF) in the RFCOMM module of the Linux Bluetooth stack. It was scored as critical with CVSSv3 and assigned CVE-2030-0007.

Listing 3.3 shows how we model the new threats with an AD called `linux_new_bof` of type Security, Product, Software, and Fine-grained. We put a short and self-contained attack description in `a`. In the field `d` we list two defenses. First, we recommend employing a memory-safe programming language (a policy) and we select Rust [30] (a concrete mechanism) because it is supported by Linux. As an alternative mitigation strategy, we suggest sanitizing kernel-space memory (policy) with the Kernel Address Sanitizer (KASAN) [26], a dynamic memory testing tool for the Linux

Listing 3.2: `sw_orion` AD. Classification: Security, Process, Software, Fine-grained

```
sw_orion:
  a: SolarWinds Orion codesign auth bypass
  d:
    Auth software supply chain: [Update and revoke code signing certs]
  surf: [Windows, SolarWinds, Orion Platform]
  vect: [Software mod, Malware distr]
  model: [Remote]
  tag: [SChain, SUNBURST, SUPERNOVA]
  risk: [cvss3_critical, cvss2_high]
  year: 2020
  cve: ["10148"]
  cwe: ["287", "288"]
```

kernel, aiming to find out-of-bounds and use-after-free bugs. We set `surf` to a list of strings progressively narrowing down the attack surface: from Linux to its Bluetooth RFCOMM (Radio frequency communication) subsystem. The `vect` is a stack-based BoF to achieve privileged code execution in kernel space from user space. The `model` is proximity-based as the adversary needs to be in Bluetooth range. We tag the AD with BT (Bluetooth), Impl (implementation-level flaw), and Linux414 (affected Linux version). The remaining AD fields are self-explanatory.

Notably, `linux_new_bof`'s defenses are ordered by effectiveness and cost, the first one fixes `a` by design, but is expensive since it requires rewriting some Linux kernel code. The other defense is cheaper, but it is only a mitigation that enables spotting the bug in the C code. Overall, we do not mandate a specific defense ordering as it is context specific. For instance, an AD might include *complementary* defenses where an ordering is unfeasible.

A catalog also includes *coarse-grained* ADs useful to represent threats' classes (e.g., an attack technique) in a single object. In Listing 3.4 we show how to implement a coarse-grained AD, named `linux_bof`, to represent a generic BoF attack on the Linux kernel. The `a` field states a high-level attack description, `d` lists the defense mechanisms that we combine to protect the Linux kernel stack and the heap against BoFs. Here, other than recommending to use Rust and KASAN, we add other defenses such as the Kernel Memory Sanitizer (KMSAN) [27] to find uninitialized values, musl [85, 84] to reduce libc's attack surface and use a hardened memory allocator, Kernel Address Space Layout Randomization (KASLR) which require a Position Independent Executable PIE kernel built, and CPU NX bit to avoid executing code on the stack. In this case, since

Listing 3.3: `linux_new_bof` AD. Classification: Security, Product, Software, and Fine-grained

```
linux_new_bof:
  a: Linux kernel PPCE via stack-based BoF on Bluetooth stack
  d:
    Memory safe PL: [Rust]
    Sanitize memory: [KASAN]
  surf: [linux, net, bluetooth, rfcomm]
  vect: [Stack BoF]
  model: [Proximity]
  tag: [BT, Impl, Linux414]
  risk: [cvss3_critical]
  year: 2030
  cve: ["0007"]
```

Listing 3.4: `linux_bof` AD. Classification: Security, Product, Software, and Coarse-grained

```
linux_bof:
  a: Stack or heap based BoF on Linux
  d:
    Memory safe PL: [Rust]
    Sanitize memory: [KASAN]
    Hardened memory allocator: [musl]
    Randomized memory layout: [KASLR]
    Non executable stack: [NX]
  surf: [Linux]
  vect: [Stack BoF, Heap BoF]
  model: [Proximity, Remote]
  tag: [Impl]
```

it is a coarse-grained and thus more generic AD, the defenses are not ordered by effectiveness, but *complementary*. The attack surface is Linux, the vectors are stack or heap-based BoFs, the `model` is proximity or remote, and the `tag` is Impl. Optional fields, such as `risk`, `year`, and `cve` are not filled as an attack class applies to multiple threats with different associated risks, years, and CVEs.

Note that, similar coarse-grained ADs can be created to model BoF attacks against other components or other threat classes.

3.2 Parse

Parse, implemented in the `parse.py` file, extracts ADs from YAML, TOML, JSON, and XML files, transforming them into Python dictionaries. For instance, the YAML AD presented in Listing 2.1 converts into the AD dictionary in Listing 3.5. The module can be easily extended to handle other file formats, such as XLS and CSV. However, we recommend using YAML, JSON, or TOML due to their enhanced writability and readability.

We implemented the parser as a high-level function that invokes specialized parsing functions based on the file extension specified in the `path` argument. These functions generate a dictionary representation of the ADs. For instance, the `_parse_yaml` function receives a YAML file containing ADs (as illustrated in Listing 2.1), parses the file using PyYAML [29] with `CSafeLoader`, which is a secure and efficient parser from LibYAML [25]. The output is a dictionary with nested sub-dictionaries, as demonstrated in Listing 3.5. Similarly, we implemented other specialized parsing functions to handle TOML, JSON, and XML files. Some parsers utilize the Python standard library (e.g., `json` and `tomlib`), while others employ dedicated Python modules (e.g., `xmltodict` [13]).

To ensure the correctness of the parsers, we implemented a series of tests using the `pytest` library. The testing code, contained in `parse_test.py`, verifies the flawless operation of all specialized parsers and confirms that they produce the same Python dictionary representation (`AD_PARSE_TEST` from `ad.py`) when parsing identical sets of ADs from YAML, TOML, JSON, and XML files. The testing files are in the `template` folder. The tests can be executed with the command `make test-parse`.

Listing 3.5: Python dictionary parsed from the YAML AD in Listing 2.1

```
ad_name = {
    # Primary fields
    "a": "Attack␣1",
    "d": {"policy1": ["mech1", "mech2"], "policy2": ["mech1", "mech2"]},
    "surf": ["surf", "subsurf", "subsubsurf"],
    "vect": ["vector1", "vector2"],
    "model": ["model1", "model2"],
    "tag": ["tag1", "tag2"],
    # Optional fields
    "risk": ["score1", "score2"],
    "year": 2023,
    "cve": ["123", "456"],
    "cwe": ["123", "456"],
    "capec": ["123", "456"],
    "vref": ["vendor-ref1"],
    ...
}
```

3.3 Check

Check is implemented in `check.py` and automatically validates the syntax and semantics of the ADs. It uses syntax-based checkers on the file containing the ADs, and semantic-based ones on the Python dictionary that is parsed from the ADs file. Check has a top-level function (`check(path; Path, words=None) -> dict`) that calls the relevant syntax and semantics checkers based on the `path` file extension. Parsing is accomplished using the Parse module presented in Section 3.2.

The validation is performed in two main steps: (1) syntax checking using yamllint [136] with its default configuration to avoid duplicate `ad_name`s and wrong indentations, and (2) semantic checking using a custom function that enforces a certain schema with specific types and allowed values on the parsed dictionary.

Listing 3.6 shows an excerpt of our ADs dict schema implemented using the schema [60] Python package. `ad_name` is a lowercase alphanumeric regex, `a` is a non-empty string, `d` is a dictionary of dictionaries where the top level keys are strings (`policy`) and the bottom values are lists of strings (`[mech1, mech2]`). `surf` is a list of strings with specific values that we enforce with a `lambda` checking that the strings are inside the `words["surf"]` list. Using a wordlist is useful to keep consistency among ADs, especially when different teams are working on the same ADs. Moreover, it helps to keep track of what is covered (e.g., the surf wordlist contains all the surfaces covered by our ADs). Finally, `year` is an int between a sensible range.

We automatically test Check with `check_test.py`. This script runs the `check` function on the YAML, TOML, JSON, and XML template ADs files in the `template` folder. To run the tests use `make test-check`.

3.4 Analyze

Analyze is implemented in `analyze.py` and provides useful automation to process a (checked) dictionary of ADs. Internally, it uses pandas [28], matplotlib [36], and graphviz [23, 24] to produce

Listing 3.6: AD dict schema excerpt

```
Regex(r"^[a-z0-9_]*$"): {
  # Primary fields
  "a": And(str, lambda a: len(a) > 0),
  "d": And(Schema({str: [str]})),
  "surf": And(Schema([str]),
    lambda surf: (len(surf) > 0) and
      len(set(surf) - set(words["surf"])) == 0,
  ),
  ...
  # Optional fields
  Optional("year"): And(int,
    lambda year: (1980 <= year <= 2030) or year == 0),
  ...
}
```

its outputs. These are free and powerful open-source libraries. Currently, Analyze can generate ADs sets, maps, trees, wordclouds, and chains to perform flat and hierarchical analyses, and now we describe how.

Analyze's entry point is `get_dataframe`, a function loading ADs from a file and returning an ADs DataFrame, which is a table-like data structure. Each DataFrame's row contains an AD, with the index corresponding to the `ad_name`, and the columns containing the AD fields such as `a`, `d`, and `surf`.

The function `get_set` generates collections of ADS using key-value filters. It allows the creation of sets based on attributes such as `surf`, `model`, and `tag`, which is useful to identify ADs of interest based on high-level requirements, such as retrieving all ADs related to a specific technology or attack technique. The `get_set` function is also used internally to perform other set-based analyzes.

The `get_map` function returns sets of AD based on known security and privacy taxonomies. Table 3.1 lists the taxonomies that we currently support, including those related to security (e.g., STRIDE, CIA), privacy (e.g., LINDDUN, UIT, and PMD), web (e.g., OTT17, OTT21), software and hardware Weaknesses (e.g., CWETH21, and CWETS22). For example, given an ADs file properly tagged with STRIDE categories, we can automatically extract six tables of ADs, one for each category, by filtering them using the AD `tag` column (field). We note that adding a new taxonomy is straightforward, i.e., extending the `taxonomies` dictionary.

Analyze is capable of generating trees of ADs based on the `surf` and `tag` fields. For example, Figure 2.2 shows a tree of protocol level (`tag = Protocol`) ADs related to BLE (`surf = [BLE, ...]`). As explained in Section 2.2, `surf` is an ordered list of strings narrowing down the attack surface. The `get_surf_tree` takes advantage of this ordering to automatically build a tree. In particular, it roots the tree to surf and creates branches for sub-surf and sub-sub-surf. Then each AD is placed as a leaf according to its `surf` list.

Analyze's `get_wordcloud` function outputs ADs wordclouds based on a field using Python's wordcloud [35] and matplotlib [36] packages. For example, Figure 2.3 shows an attack surface wordcloud computed from the `bt.yaml` file discussed in Section 3.1. Internally, the function takes an AD dataframe and a column key, collects all the columns value in a list, and then uses a `Counter` data structure from the `collections` library to count the occurrences of each value. This is better than counting each string as a word, as some words in the cloud contain multiple

Table 3.1: Taxonomies currently supported by ADF's `get_map`. UIT refers to a privacy taxonomy from International Association of Privacy Professionals (IAPP) [47], PMD to a privacy taxonomy from NISTIR 8062 [88], OTT to the OWASP Top Ten Web Application Security Risks [99], CKC to the Cyber Kill Chain by Lockheed Martin [67], CWETH21 to the top ten hardware CWE from 2021 [72], and CWETS22 to the top twenty-five software CWE from 2022 [73]

Taxonomy	Keywords
STRIDE	Spoofing, Tampering, Repudiation, ID, DoS, EoP
CIA	Confidentiality, Integrity, Availability
UIT	Unlinkability, Intervenability, Transparency
PMD	Predictability, Manageability, Dissassociability
LINDDUN	Linkability, Identifiability, Non repudiation, Detectability, ID, Unawareness, Non compliance
OTT21	Broken access control, Cryptographic failure, Injection, Insecure design, Security misconfiguration, Vulnerable and outdated component, Identification and authentication failure, Software and data integrity failure, Security logging and monitoring failure, Server-side request forgery
OTT17	Injection, Broken authentication, Sensitive data exposure, XML external entities, Broken access control, Security misconfiguration, Cross-site scripting, Insecure deserialization, Using components with known vulnerabilities, Insufficient logging and monitoring
CKC	Reconnaissance, Weaponization, Delivery, Exploitation, Installation, Command and control, Actions on objectives
CWETH21	1189, 1191, 1231, 1233, 1240, 1244, 1256, 1260, 1272, 1274, 1277, 1300
CWETS22	787, 79, 89, 20, 125, 78, 416, 22, 352, 434, 476, 502, 190, 287, 798, 862, 77, 306, 119, 276, 918, 362, 400, 611, 94

strings (e.g., "Feature exchange" counts as a single word in the cloud).

Analyze's `get_chains` function generates a chain of ADs, given an ADs dataframe and a target AD using graphviz. For instance, Figure 2.1 shows the iOS Pegasus RCE from 2021 represented as a chain of four ADs. Internally, the function creates a `Digraph` with `strict=False` to avoid double edges and `rankdir = LR` to draw from left to right. Then, it selects from the dataframe the ADs with the same attack surface and sub-surface of the target AD via the `surf` column. Next, it tries to build a chain by attack vector looking at the `vect` column. In particular, if an AD `vect` field is a subset of another, it means that the two are chainable.

Chapter 4

ADF Usage

In this Chapter, we show how to use the ADF to map abstract security (and privacy) requirements into concrete recommendations. Specifically, we introduce a reference scenario where we are tasked to develop a new secure and privacy-preserving cryptowallet with the ORSHIN TLC. Then, we set seven abstract requirements on the cryptowallet and the trustworthiness of its life cycle. Finally, we show how to use the ADF to address these requirements by creating and processing ADs.

4.1 Cryptowallet scenario

We assume a scenario where we are developing a cryptowallet, an embedded device providing privacy and security critical features (e.g., storing cryptocurrency private keys and performing cryptocurrency transactions) with the TLC. Our goal is to map abstract security and privacy requirements on the cryptowallet and its life cycle into concrete recommendations using the ADF.

Figure 4.1 shows a simplified block diagram of our cryptowallet. It has a Secure Element (SE) connected via a secure bus to a general-purpose microcontroller (Micro). The microcontroller runs a Linux-based OS. The SE runs on a separate chip with a dedicated real-time operating system (RTOS) and performs the most sensitive operations (e.g., generation and storage of keys, signing and verifying the transactions). The SE and the Micro employ secure boot via a boot ROM. The cryptowallet supports Bluetooth Low Energy (BLE) for wireless connectivity, and Universal Serial Bus (USB) for wired connections. Moreover, it can be used as a Fast IDentity Online (FIDO) authenticator for two-factor and single-factor (i.e., passwordless) authentication [2].

We develop the cryptowallet employing the *Trusted Life Cycle (TLC)* proposed in Task 2.1. Figure 4.2 shows the TLC phases. During Threat Modeling and Risk Assessment (TM and RA)

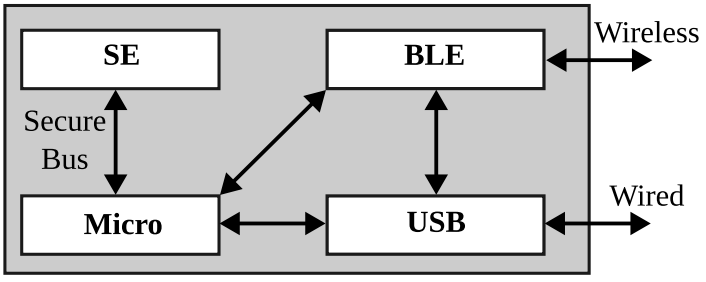

Figure 4.1: Cryptowallet block diagram (simplified)

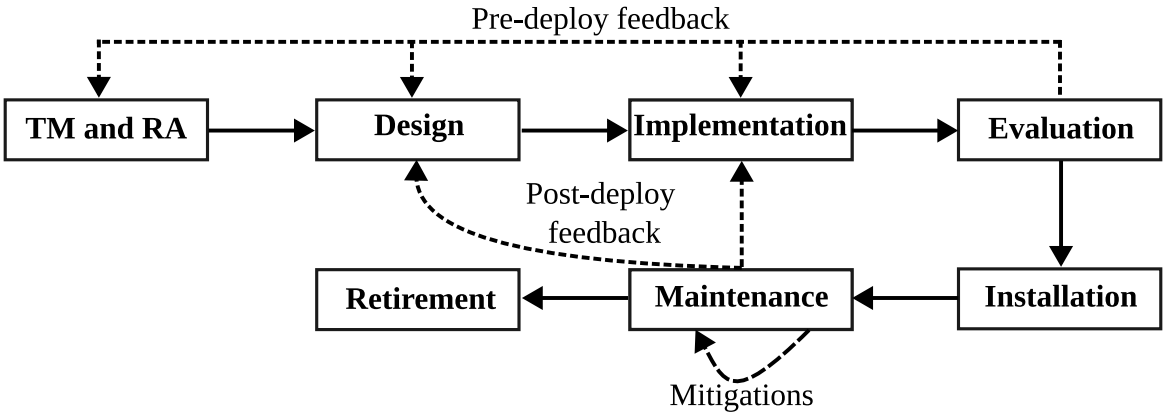

Figure 4.2: ORSHIN Trusted Life Cycle (TLC) used as a reference life cycle for our cryptowallet. TLC has seven phases connected by solid lines. Dotted lines represent mitigations, and pre- and post- deployment feedback.

we come up with the product and process security and privacy requirements for our cryptowallet (presented later in Section 4.2). During Design, we design the hardware, software, and protocol aspects of our cryptowallet based on functional requirements and the ones set in the TM and RA phase. All the parts are implemented in the Implementation phase. During Evaluation, we perform extensive hardware and software testing including presilicon hardware and software testing (e.g., fuzzing, SC, and FI). Evaluation provides useful pre-deployment feedback that we can use to refine any of the first three TLC phases. The cryptowallet will be deployed during Installation and then managed during Maintenance, including firmware updates, and disposed during Retirement. For more information about the TLC refer to ORSHIN Deliverable 2.1 (D2.1).

4.2 Abstract requirements

Now we are in the TM and RA phase of the TLC and we want to address *process related* security issues, such as the ones discussed in the ISA/IEC 62443-4-1 standards [56]. For instance, we want to avoid bad development and access control practices during Design, Implementation, and Evaluation. Moreover, we want to cope with *product-specific* security and privacy threats related to the cryptowallet's software, hardware, firmware, and communication protocols.

In this simplified but realistic scenario we set seven *abstract requirements (AR)*:

AR1: The TLC should follow the ISA/IEC 62443-4-1 standards

AR2: The SE should be resilient against side channel (SC) and fault injection (FI) attacks

AR3: The Micro should be resilient against speculative execution attacks

AR4: The SE should be resilient against relevant presilicon attacks

AR5: The SE should be resilient against physical hardware attacks

AR6: The BLE module should be resilient against protocol and implementation-level attacks

AR7: The FIDO2 module should be resilient against generic and device-specific attacks

4.3 Creating and Using the ADs

To address the seven AR we need to create (or reuse) dedicated ADs catalogs. We think that the best approach is to involve domain-expert knowledge for each AR. Within the ORSHIN consortium we come up with the following ARs assignment:

- SEC works on AR1 (ISA/IEC 62443-4-1) and AR7 (FIDO)
- KUL works on AR2 (SC and FI) and AR3 (Speculative)
- NXP works on AR4 (Presilicon auditing)
- TXP works on AR5 (Physical hardware)
- ECM works on AR6 (BLE)

Since the ADF is novel we had to create the ADs catalogs from scratch. But, in the future we envision high-quality and open-source ADs catalogs that are usable and extendible by experts and non-experts. To simplify the description in the next paragraphs we use the creation and usage of the AD catalog to address *AR6 (BLE)*.

While creating the ADs for an AR, is paramount to properly select the attack surfaces. For AR6 we focused on the BLE attack surface and ignored attacks on Bluetooth Classic (BC) and Bluetooth Mesh (BM). However, we also included cross-transport BLE threats, such as the BLUR attacks [8], as they enable exploiting BLE from BC. Moreover, we decided to TM protocol-level and implementation-level threats on BLE while ignoring relevant surfaces (e.g., iOS or Android bugs since our BLE module runs a custom RTOS).

Once the attack surfaces are properly set we need to select their relevant threats to be modeled with the ADs. There are several ways to list these threats. A team might start from an (old) list of attacks already in their TM. For AR6, we looked at real-world and impactful attacks described in the academic and industrial literature and at high-risk CVEs (some of which are still unpatched). We came up with a list of attacks and then created an AD for each of them. In the process, we decided to cover also Bluetooth Classic (BC) and Bluetooth Mesh (BM) but we ignored them when working on AR6.

One challenging aspect of AD creation is having consistent terminology among the ADs, especially when a team is collaboratively creating a catalog. Inconsistent terminology slows down ADs creation and might introduce subtle errors, such as ADs duplication. For AD6, we used an *allow list* strategy to mitigate these risks. We enforced that the words in the `surf`, `vect`, `model`, and `tag` ADs fields were in our allow list. The allow list was incrementally updated while creating new ADs and now acts as a valuable self-documenting resource. For example, by looking at our attack surface allow list we can quickly assess our attack surface coverage. Note that our allow list support single- and multi- strings words, like “Pairing” or “Key agreement” as shown in Figure 2.3.

A high-quality ADs catalog is all we need to build our defense plan. Within AR6, our ADs are based on real-world and specific vulnerabilities. Each AD provides high-level policies and concrete mitigations, allowing us to quickly identify the best mitigation plan. For example, by implementing all protocol-level defenses in our ADs catalog, we achieve authenticated, confidential, and integrity-protected communication and even defense-in-depth. Whenever a new protocol-level threat is discovered we can update our ADs catalog and refine our defense plan. A similar reasoning holds for implementation-level threats.

We recommend adopting an *iterative* and *incremental* approach when creating and using

ADs. Often, the first version of an AD improperly models an attack and/or a defense or overlooks an important threat aspect. In our experience with AD6, we had to review the Bluetooth ADs multiple times. For example, by using attack trees and wordcloud visualizations as in Figures 2.2 and 2.3, we discovered holes in our threat coverage. Also, when creating AD chains we realized that multiple defenses per AD might be needed.

Chapter 5

ADF Evaluation

We evaluated the ADF in *seven* complementary real-world setups related to the seven abstract requirements introduced in Section 4.2 (i.e., AR1, . . . , AR7). We took full advantage of SEC, KUL, NXP, TXP, and ECM from the excellent ORSHIN consortium. We asked each partner to use ADF to threat model a cryptowallet attack scenario and address one or more ARs within their expertise. Overall, we evaluated traditional TM domains, such as software and protocol security, but also less explored and critical ones, including the TLC, pre-silicon testing, invasive physical attacks, and microarchitectural threats.

Table 5.1 summarizes the results of our seven case studies. The columns show in order the TM domain, the addressed AR, the related consortium member, the associated ORSHIN work package (including WP3, WP4, and WP5), the covered domains (e.g., hardware, software, process, and product), the produced ADs (169), and their related YAML files in the `toolkit` directory.

We are satisfied with the results of our evaluation for several reasons. Firstly, we conducted field testing of the ADF from various perspectives, involving users with diverse backgrounds. This allowed us to assess its effectiveness and usability across different scenarios. Secondly, our threat modeling activities covered many threats, including specific vulnerabilities affecting individual software components as well as generic attacks and attack techniques applicable to multiple components. During the evaluation of the ADF, we identified certain limitations, which we successfully addressed by reviewing the AD object data model. For instance, we enhanced the capabilities of the ADF by introducing the ability to specify multiple entries in the `d` field, enabling the modeling of complementary or alternative defense strategies.

Our evaluation generated high-quality ADs that will be used as blueprints and shared with the community. Sharing our ADs catalogs will be particularly valuable to ADF beginners and security professionals unfamiliar with threat modeling (e.g., hardware and embedded systems developers). In the following sections, we present in detail each case study.

5.1 ISA/IEC 62443-4-1 (AR1, WP2, SEC)

In this case study, SEC employed the ADF to TM the secure life cycle specified in the ISA/IEC 62443-4-1 standard that can be used to develop our cryptowallet (i.e., AR1). SEC developed 40 ADs in seven YAML files, stored in the `62443` folder. ISA stands for International Society for Automation, and IEC for International Electrotechnical Commission. We now report verbatim the

Table 5.1: Evaluation results from our seven complementary case studies. We take advantage of our diverse consortium of academic and industrial experts (SEC, KUL, NXP, TXP, and ECM). We cover all the technical work packages (i.e., WP2, WP3, WP4, and WP5). We cover hardware (HW), software (SW), firmware (FW), protocols (PT), life cycles (LC), security (SE), and privacy (PR) threats. The AR column shows abstract requirements defined in Section 4.2. We developed a total of *169* ADs.

Domain	AR	Mem.	WP	Coverage	ADs	ADs files
62443-4-1	1	SEC	2	LC, SE	40	`62443/*.yaml`
SC FI	2	KUL	3	HW, SE, FW	9	`sc-fi.yaml`
Spec. Exe.	3	KUL	3	HW, SW, SE	8	`microa.yaml`
Presilicon	4	NXP	4	HW, SW, FW, SE	8	`presil.yaml`
Physical	5	TXP	4	HW, FW, SE, PR	26	`physical.yaml`
Bluetooth	6	ECM	5	SW, FW, PT, SE, PR	46	`bt.yaml`
FIDO2	7	SEC	5	HW, SW, FW, PT, SE	21	`fido*.yaml`

SEC report prepared by Stefano Cristalli.

5.1.1 Threat modeling of process requirements with the AD framework

This section describes the application of the ADF to the threat modeling of process requirements. As our use case, we consider the *translation* of the requirements of international standard ISA/IEC 62443-4-1 regarding the Secure Development Life Cycle of secure components.

The ISA/IEC 62443 standard is divided into four tiers, each having multiple work documents. The requirements for the Secure Development Life Cycle are described in the work document ISA/IEC 62443-4-1 (tier 4, part 1), and are divided into eight *practices*, i.e., categories for grouping requirements:

1. Security management
2. Specification of security requirements
3. Secure by design
4. Secure implementation
5. Security verification and validation testing
6. Management of security-related issues
7. Security update management
8. Security guidelines

In total, we developed 40 ADs that map to the requirements described in the eight practices. We named each AD according to the requirement that it maps to (e.g., `sm-1-development_process`). We directed our primary effort towards rewriting the requirements in the form of threats/mitigations, so we leveraged the AD primary `a` and `d` fields. Additionally, we enriched the description of the requirements by using the `surf`, `vect`, and `tag` fields.

The first relevant observation is about the extreme flexibility of the ADF. ADs were designed to allow the modeling threats related to the product (e.g., related to technical features) but also

related to the process (e.g., that impact the processes of design and development of the product) related to design and development.

We can apply the same distinction to the requirements, thus obtaining the `product` requirements (to select the technical features related to product threats) and `process` requirements (to implement a secure development life cycle that is protected against process threats).

While threat modeling as a discipline does consider process requirements (e.g., supply-chain attacks and malicious insiders), in reality, most threat modeling frameworks, methodologies, and attack libraries focus only on product requirements. Consider, for instance, the STRIDE approach to threat modeling: it is evident that its six threat categories mainly apply to product threats.

The lack of threat modeling applied to process requirements means that even consolidated sets of process requirements (such as the ones coming from the ISA/IEC 62443-4-1 standard) are *not* mapped to threat models. Instead, they are fixed in their specification and not flexible depending on their application context. Some sets of requirements are specifically derived from a threat model, like the best practices described by Good Practices for Security of IoT from the European Union Agency for Cybersecurity (ENISA). However, even so, while more justified, once established, they still lack the flexibility for evaluating only a subset of them, depending on the context.

The AD framework is flexible enough to model products and process requirements correctly. We think that once a good number of consolidated AD items will have been made their application could change how process requirements are conceived.

To show the practical benefits of threat modeling process requirements with ADF, we have translated the process requirements contained in ISA/IEC 62443-4-1, namely specifying them in the AD format from an abstract point of view. Moreover, leveraging our experience as a company that is ISA/IEC 62443-4-1 *certified*, we have provided an example of what a concrete application of the requirements may look.

The main benefit of using ADF is that it offers a new way of thinking about process requirements. Specifically, it forces us to think of requirements in terms of threats that the requirement mitigates, thus uncovering the threats themselves (usually hidden when mechanically evaluating a list of process requirements).

As an example, consider the *SM-4: Security expertise* requirement:

> A process shall be employed for identifying and providing security training and assessment programs to ensure that personnel assigned to the organizational roles and duties specified in 5.3, SM-2: Identification of responsibilities have demonstrated security expertise appropriate for those processes.

Our translation of SM-4 to an AD looks like Listing 5.1. First, we point the attention to the `a` field, the threat the requirement addresses. In this case, it refers to the possibility of having inadequate security expertise in the product team. From the start, this makes very clear the purpose of the requirement.

Next, consider our decomposition of the requirement prescriptions into three items in the `d` field:

1. We want to define which level of security expertise is required (without, it is impossible to evaluate whether the expertise is sufficient or not).
2. We need a way to assess the security expertise so that we know if there are any gaps

Listing 5.1: `sm_4_sec-exp` AD excerpt

```
sm_4_sec_exp:
  a: Insufficient security expertise of the product team
  d:
    Define the required level of security expertise of the product team:
    ["..."]
    Assess the current level of security expertise of team members: ["..."]
    Provide adequate security training so that the security expertise of the
    product team matches the required level: ["..."]
  surf: [Processes, Expertise]
  vect: [Unclear definition, Lack of competence]
  tag: [Processes, Security expertise, Roles, Responsibilities]
```

between the desiderata and the actual situation.

3. We want to address any gaps by providing adequate security training.

The full-text form of the requirement contained all the information mentioned above; however, it was implicit and required careful analysis to extract and comprehend. By structuring the requirement using the AD format, it became easier to ensure that all aspects were adequately addressed. This approach also prompted considerations regarding the definition of security expertise and the level of specificity required for product-specific requirements. While our internal processes are well-established and certified, we discovered that our internal documentation might be ambiguous. Through a meticulous examination guided by the AD format, we recognized this as an opportunity to enhance our internal documentation and improve clarity in defining the required level of security expertise.

Introducing the logical structure of ADs in the modeling process requirements has the advantage of revealing hidden hierarchical structures and uncovering links and references among individual requirements. In the context of ISA/IEC 62443-4-1, while the Practices already offer some level of grouping for the requirements, it is common to find requirements from different Practices that reference common themes. Additionally, some requirements may contain directives that are later more precisely specified in other requirements. By utilizing the AD format, these connections and relationships among requirements become more apparent, allowing for a comprehensive understanding of the requirement set and facilitating better specification and organization.

As an example, consider the *SM-1: Development Process* requirement:

> A general product development/maintenance/support process shall be documented and enforced that is consistent and integrated with commonly accepted product development processes that include, but are not limited to: a) configuration management with change controls and audit logging; b) product description and requirements definition with requirements traceability; c) software or hardware design and implementation practices, such as modular design; d) repeatable testing verification and validation process; e) review and approval of all development process records; and f) life-cycle support

This is the very first requirement introduced by ISA/IEC 62443-4-1, and lays the foundation for the setup of a Secure Development Life Cycle. However, most of its sub-points are later better specified by other requirements; for example, the *d* directive (i.e., repeatable testing verification and validation process) is exhaustively handled by the specific requirements of Practice 5, which

Listing 5.2: `sm_1_dev-proc` AD excerpt

```
sm_1_dev-proc:
  a: Undefined development/maintenance/support processes
  d:
    Implement config mgmt with change control and audit logging: ["Redmine"]
    Require product desc and reqs def with req traceability:: ["Redmine"]
    Define design practices: [Addressed in @sd-4-secure-design-best-
   practices]
    Define implementation practices: [Addressed in @si-2-secure-coding-
   standards]
    Implement repeatable testing and validation processes: [Addressed in
   @svv-*]
    Enforce review and approval of all development process records: [
   Addressed in @sm-12-process-verification]
    Implement life-cycle support: ["..."]
  surf: [Processes]
  vect: [Unclear definition]
  tag: [Processes, Requirements, Design, Implementation, Testing, Review,
    Vulnerability management, Maintenance]
```

is entirely dedicated to testing. We think that this hierarchical connection should be highlighted, and the AD model gives us a chance to specify links of this sort.

In Listing 5.2, we present our proposal for modeling the requirement SM-1 with an AD object. We have decided to link other requirements explicitly when appropriate. For instance, we have specified ”Addressed in @svv-*” as our response to point *d* cited above. We use the notation ”@” to indicate a reference to another AD item and the ”*” symbol as a wildcard, so ”@svv-*” means ”all the ADs starting with the prefix svv-”, which are all ADs mapping to the requirements of Practice 5.

The possibility of categorizing information based on metadata, such as the fields `surf`, `vect` and `tag` makes it possible to have a clearer understanding of the context of the requirements. In Practice 1, we see both requirements that inherently refer to the threat of having ”unclear definition” of some processes/procedures, and ones that instead implicitly model some damage to assets, either due to human error or accidental causes, or to malicious action. Thinking about threat modeling, the possibility of distinguishing between these two classes is valuable. Being able to isolate all and only threats that require an attacker makes it possible to understand which items are of higher priority, leaving the implementation of process improvements for a second iteration. The same reasoning holds for many other practical categorizations, for example filtering all the requirements that reference the threat model becomes as simple as filtering using the tag ”Threat model” in the ADs.

Given our experience with other threat modeling approaches, we feel that the AD framework provides a logical and structured way to consolidate information about threats. Although the framework does not provide an enumeration strategy, it is easy to see how a rich library of ADs could serve as a database for other methodologies (e.g., STRIDE) or directly as an attack library, in any case providing highly-detailed threat descriptions. Its flexibility allows for selection, filtering, extension, and customization of the threat set depending on the context, making it the ideal tool for producing context-specific threat sets. Its ability to effectively model process threats make it possible to have a definition of Secure Development Life Cycles (SDLC) that can be adapted based on the context (e.g., software SDLC VS industrial SDLC for high-security crypto chips).

5.2 Side Channel and Fault Injection (AR2, WP3, KUL)

In this case study KUL (COSIC) employed the ADF to TM side channel and fault injection attacks on a generic cryptowallet secure element (i.e., AR2) and constructed 9 ADs (i.e., `sc-fi.yaml`). We now report verbatim the KUL report, prepared by Jesse De Meulemeester and Linde Nouwen.

5.2.1 AD Feedback

We considered the ADF from the perspective of physical side-channel (SC) and fault injection (FI) attacks. Threat modeling for these types of attacks is complex as existing methodologies are defined from a software perspective. For instance, STRIDE starts from a data flow graph where all intended information flows are considered. The key to physical attacks, however, is that these attacks may target information flows that the designer did not intend to be recoverable. This makes them incompatible with these types of methodologies.

In this section, we construct some ADs by considering physical side-channel and fault injection attacks. The resulting list of ADs can then be used when threat modeling a device. Implementing the relevant defenses listed in the ADs can make a device more resilient against these types of attacks.

Generating AD list

AD Structure The structure of ADs provides sufficient detail to describe physical attacks, while still being simple to construct and easy to read. A property that is crucial to model physical attacks is the ability to specify multiple policies and countermeasures per policy (i.e., AD `d` field) This is especially useful as countermeasures against these types of attacks may either *not* exist, or there may exist multiple different ones, each with its benefits and downsides. Similarly, a countermeasure may exist but may not be practical within the considered application. Allowing the AD creators to specify multiple policies and countermeasures is thus a good addition when considering physical attacks.

Another useful property is the ability to specify AD optional fields. For instance, we added a defense reference (`dref`) field to our ADs that references certain sources for the specified countermeasures. This can provide more context about the effectiveness and performance of the countermeasures when threat modeling using the ADF.

An interesting property of the ADF is the ability to create structured representations based on the different fields in the AD, such as `surf`, or `vect`. This, however, requires that the keywords used are coherent across ADs, which when ADs are constructed by different individuals may not always be the case.

Abstraction level One less obvious point from the perspective of physical attacks is the appropriate abstraction level. Side-channel attacks, for instance, can be described on various abstraction levels, ranging from broad descriptions of one type of attack to specific ones describing a precise attack on an exact target. The AD allows modeling threats with different abstraction levels, but deciding which one to use might be challenging.

For example, we will use a *simple power analysis (SPA)* attack against the square and multiply implementation of RSA. This problem can be captured at different abstraction levels in an AD. For

Listing 5.3: High-level (coarse-grained) SPA AD

```
spa_nct:
  a: SPA attacks on non-constant-time implementations
  d:
    Constant time implementation: []
  surf: [Implementation]
  vect: [SCA]
  model: [Physical, passive]
  tag: [Implementation, Side channel, SPA]
```

instance, going down in level of abstraction, one could think of SPA attacks on non-constant time implementations, SPA attacks on RSA, and SPA attacks on square and multiply.

There are pros to both having a high-level description of the attacks and having a more detailed description. As in Listing 5.3, a broader description allows us to describe more generalized attacks, making it potentially more useful for threat modeling as a broader surface is described. In contrast, a broader description makes it more difficult to describe specific countermeasures.

In terms of concrete defenses, having a detailed AD description, as in Listing 5.4 allows for much more fine-grained defense descriptions. However, this approach has the disadvantage of requiring many more AD entries. A similar attack to Listing 5.4 is, for instance, a SPA attack against the double and add algorithm in elliptic curve cryptography (ECC). A broader AD would capture both but would be able to provide less relevant (i.e., attack-specific) countermeasures.

Thread modeling using ADs

Methodology vs Catalog system The current idea is to use ADs as the input of the threat modeling system rather than as the outputs. However, there are some potential problems with this approach. For example, it does not allow to model any attacks not currently present in the AD database. It may also make it more difficult to find all the relevant threats for a specific use case, as there is yet to be a methodology to do this. Furthermore, the AD system still has the challenge of creating a hierarchy among the threats. The latter is solved partially by the threat chaining.

Abstraction level The person creating the ADs decides the abstraction level of the described attacks, as we noted above. Since the AD database is used as input for threat modeling, the specificity of the threats following this methodology will depend on the creator of the database entries rather than the person following the methodology.

Listing 5.4: Specific (fine-grained) SPA AD

```
spa_sm:
  a: SPA attacks on square and multiply
  d:
    Constant-time implementation: [Square and multiply always, Montgomery
    ladder]
  surf: [Crypto algorithm, RSA, Exponentiation, Private key]
  vect: [SCA]
  model: [Physical, passive]
  tag: [Implementation, Side channel, SPA]
```

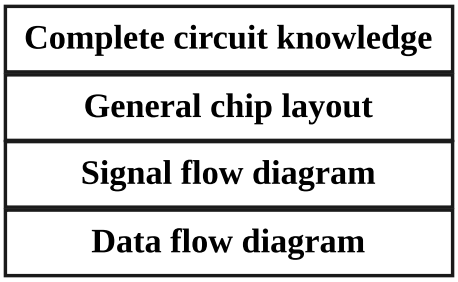

Figure 5.1: STRIDE layers

STRIDE layered extension

The threat modeling idea we were working on was to modify STRIDE to work for various types of threats rather than only software-related ones. Currently, STRIDE starts from a data flow diagram. However, this diagram usually only shows data flows that the manufacturer had envisioned, which makes it impractical to model hardware-related threats. Meanwhile, the STRIDE categories could be used for hardware and software-related threats.

Our idea was to have different layers to use STRIDE on. For example, in one layer, we could rely only on the data flow diagram to model threats. We could add physical access in another layer, yet the chips and internal workings remain a black box for the attacker. In the final layer, the attacker could have full knowledge of the circuit and use it for an invasive attack. In a layer between the data flow diagram one and the full circuit knowledge one, a signal flow diagram could be used to indicate the power line, the electromagnetic radiation, and the clock signal (and potentially also others) for side-channel attacks and fault injection. The higher the layer used in the threat modeling, the higher the capabilities of the hypothetical attacker during the modeling exercise.

Figure 5.1 shows a potential set of layers. Only the data flow diagram is used to model threats at the lowest layer. In this layer, we assume that the attacker has no physical access and can only see the data flow that the manufacturer envisioned. In the second layer, we add a signal flow diagram, and we assume that the attacker can measure the electromagnetic radiation and read and modify specific signals, such as the clock and power signal. Now side-channel and fault injection-related threats can be modeled. The threat modeling process progressively incorporates attackers with increasing expertise in the subsequent layers. Ultimately, in the final layer, the attacker can reverse engineer the entire chip and execute various invasive attacks.

This threat modeling approach offers flexibility in its implementation. It allows for two approaches: constructing the threat model layer by layer or focusing on a specific layer. For instance, one can effectively model software-related threats by solely considering the data flow diagram layer. Additionally, modifying specific layers to enhance the modeling capabilities or introducing additional layers to expand the range of attacks that can be included or excluded during the modeling process is possible. This adaptability enables customization of the threat modeling approach to suit specific requirements and objectives.

5.3 Speculative Execution (AR3, WP3, KUL)

In this case study KUL (DistriNet) employed the ADF to TM speculative microarchitectural attacks and some related covert channels that can target a cryptowallet microprocessor (i.e., AR3). The ADs can be found in `microa.yaml`. We now report verbatim the KUL report, prepared by Marton

Bognar and Frank Piessens.

5.3.1 Threat modeling speculative execution attacks

The purpose of this case study is to evaluate how well the threats related to speculative execution attacks and the underlying microarchitectural side channels can be documented in the ADF. Starting from the description of speculative execution attacks in the original Spectre paper [62] and a classification tree [15], we constructed ADs for the various Spectre variants, along with some cache attacks that can be used to transmit secrets from the transient domain to the architectural domain.

5.3.2 Feedback on the use of the framework

In our overall experience with the framework, we found it to be highly beneficial. The ADF exhibits a flexible data format with well-chosen and self-explanatory fields, making it easy for machines and humans to process. The use of the YAML format proved to be a suitable choice. As demonstrated in this case study, the construction of a catalog of ADs shared similarities with the utilization of checklists in threat modeling. We believe that developing an AD catalog can contribute to achieving completeness and reproducibility in threat modeling activities. However, we encountered several challenges while using the ADF framework. Firstly, when modeling attacks such as speculative execution attacks and microarchitectural attacks, we faced the issue of determining the appropriate level of abstraction. These attacks can be described as general classes, specific variants, or instances tailored to particular processor architectures. The framework lacks clear guidance on selecting the optimal level of abstraction, and thus we had to make somewhat arbitrary choices. Future experiences with the framework may help establish guidelines to assist AD writers. Secondly, we contemplated the broader application of the ADF framework beyond the specific class of Spectre attacks we examined. We pondered how to effectively model attack classes documented in academic literature within the ADF framework. We believe that documenting such attack classes in the ADF enhances the comprehensiveness of the catalogs and facilitates access to academic research findings for practitioners. Instead of reading the entire paper, practitioners can rely on the ADF and associated tools to assess the applicability of threats or attacks to their products.

More specifically, we considered capturing the attack classes documented in survey papers or systematization-of-knowledge papers, for instance the well-known papers:

- “SoK: Eternal War in Memory” [119], a paper that gives a rigorous overview of all memory safety attacks such as code corruption attacks, control flow hijacking attacks (including things such as return-oriented-programming attacks) and data-only attacks.
- “A Systematic Evaluation of Transient Execution Attacks and Defenses” [15], a paper that gives a rigorous overview of the entire class of transient execution attacks, including the Spectre attacks considered in this case study, but also including other transient execution attacks, such as Meltdown-type attacks.

We found that, in general, it is not obvious how to maintain the structure that these papers bring into the class of attacks in the AD description. For instance, [119] provides what they call an *attack model* of memory corruption attacks (see Figure 1 in [119]) where they structure attacks in *stages* and consider different variants of these stages. Defense techniques can then be mapped on this model in a principled way. Even if ADF supports various ways of structuring information

about attacks and defenses, we found it hard to see how all the information captured in the attack model of [119] could be maintained in ADF. Hence, an interesting question for further research on ADF might be how to enrich the underlying data model further.

For the paper on transient execution attacks [15], a similar comment applies. That paper also provides a structured classification tree (See `https://transient.fail/` for an up-to-date version of that classification tree.) While our rendering of Spectre attacks in ADF matches very well with one level of that classification tree (the level corresponding to the microarchitectural buffer used in the attack), higher levels in the tree could likely be represented as tags in the ADs. How to represent all information, including that at the lowest levels, is less obvious. Of course, one could choose to represent all the leaves of the classification tree as ADs, but that seems to lead to an explosion of ADs which might also not be desirable.

In summary, while ADF works very well for specific attacks on a concrete product, the precise representation of more abstract attack classes with their properties may need more research. Finally, for completeness, we report some of the more minor questions or unclarities we encountered while doing this case study:

- It was unclear how to document best that a given surface/vector makes the attack easier but is not required (e.g., controlling the PHT for Spectre-PHT).
- It was not entirely clear what to use tags for: it can be used for *metadata*, but there was little guidance on what meta-data would be relevant or required (which might also be application-specific).

5.4 Presilicon Attacks (AR4, WP5, NXP)

In this case study, NXP employed the ADF to TM pre-silicon testing of CV32E40S, a RISC-V secure core that might be employed by a cryptowallet (i.e., AR4) Furthermore, NXP constructed 8 ADs (i.e., `presil.yaml`). We now report verbatim the NXP report prepared by Volodymyr Bezsmertnyi.

5.4.1 Presilicon TM of the CV32E40S Secure Core with ADF

We took the secure RISC-V core CV32E40S as an example of the usage of ADF for the threat modeling of the core design. We noticed directly during the threat modeling that it is possible to analyze even generic designs like the secure core using ADF. During the pre-silicon stage, we do not consider physical defenses such as shields and sensors, since they cannot be tested without a silicon die. The attack vectors that can be tested during the pre-silicon stage are limited compared to a real application of a fully defined system on chip with precisely specified hardware and software components. However, we managed to identify the most important application-agnostic attack vectors on the secure core. For example, in `presil.yaml`, the `timing_info`, `bus_tampering`, `dfa`, and `data_extraction` ADs can be seen as generic for any core and hence can be reused during the TM of a concrete product, where more information about hardware and software is available.

As another exercise, we added more components and specified a System on Chip (SoC) sample. The SoC features RAM, non-volatile memory storing a firmware image, ROM containing code of a secure bootloader and cryptographic keys, a serial interface, a peripheral bus, and the secure core CV32E40S. This SoC is still generic but can be a foundation for other SoC

designs like crypto wallets and smart cards. New defensive policies also arose since some critical components have been added to the design. The framework's flexibility allows for specifying the policies on different abstraction levels. For more concrete applications, the analysts would need to identify security-relevant assets and goals, critical functionality, and estimate attacker knowledge/capabilities and risks.

The framework brings the following advantages for the threat modeling in the pre-silicon phase:

- Having all attack objects specified, we can compile a list of defensive mechanisms that require implementation and derive a task list.
- By introducing risk factors as an optional field, the attacks can be ordered or grouped by a risk metric which allows for prioritizing the testing effort and efficiently distributing the workload.
- For future usage, it makes sense to develop tools for constructing, visualizing, and analyzing attacks using standardized keywords for vectors, surfaces, models, policies, and tags. This would enable efficient and usable ways of working with the framework.
- Due to the objects' file format and hierarchical structure, we can build trees and attack chains and extract required defenses against more sophisticated attacks with multiple stages. This can be achieved by embedding the attack name `a` of the object into the `model` field of the attack object of the next stage in a complex multi-stage attack scenario. For example, the object `code_exe` is a prerequisite for the `dfa` object, so together, they form an attack chain.

5.5 Physical Attacks (AR5, WP4, TXP)

In this case study, TXP employed the ADF to TM invasive physical attacks on a cryptowallet (i.e., AR5), which are impactful but typically neglected by TM, and constructed 26 ADs (i.e., `physical.yaml`). We now report verbatim the TXP report prepared by Olivier Thomas.

5.5.1 AD Feedback

The ADF offers a systematic approach to documenting properties related to attacks and their mitigations, serving as a valuable resource for design teams, evaluators, and design reviewers. Particularly in the context of physical attacks, where public information is limited, the existence of a comprehensive database of known attacks and guidelines becomes highly advantageous.

By incorporating ADs into a database, accessibility to the provided information is enhanced, regardless of the user's level of expertise in the field. Designers can utilize ADs to architect their designs effectively, enabling them to create targeted guidelines for various aspects of their designs. This facilitates easy access to information for the different teams involved in a project.

The utilization of the AD database can vary depending on the specific context. In the case of physical attacks, ADs can assist in assessing the required protections based on the elements that need safeguarding and the types of attackers to be defeated. The policy and attack surface are primary indicators for IC designers during their threat modeling tasks. Identifying what needs protection is insufficient; understanding the attack type and vector describes the attacker's capabilities. With this information, known attacks can be identified, and appropriate mitigations can be

implemented.

For instance, in ROMs and boot ROMs, preventing access to physical adversaries (e.g., probing and imaging attacks) is crucial. The AD database contains four entries showcasing different attack types and their mitigations. By extracting information from the policy and attack surface, designers can determine the expertise attackers should possess. Less capable attackers may attempt to extract binary data using images of physical bits, which can be countered by implementing proper scrambling schemes within the ROM itself.

Encryption can be employed to enhance security against attackers capable of reverse engineering the ROM's scrambling circuitry. When implemented correctly with a ROM external decryption circuit, attackers are forced to utilize fully invasive techniques, significantly reducing the pool of potential attackers. Dedicated countermeasures can be implemented if the application necessitates protection against highly skilled invasive attackers (e.g., for long IC lifetimes).

This example highlights the need for a comprehensive attacker classification system that could be incorporated as an additional tag within the framework. The database's flexibility in adding tags addresses the evolving nature of security considerations.

The database also describes attack techniques such as *Focused Ion Beam (FIB)* modification, which should ideally be prevented altogether. These techniques represent attack paths that can lead to various possibilities depending on the target. In the case of FIB modification, multiple mitigations are presented to significantly raise the bar for potential attackers.

Furthermore, listing potential attacks and attack vectors based on the attack surface proves valuable. For instance, instruction corruption corresponds to several unique ADs that link to different attack vectors and various mitigations. Suppose semi- or fully invasive attacks are not feasible due to restricted access to the IC. In that case, the circuit must be protected solely against Voltage Fault Injection (VFI), which might require the implementation of power filtering and glitch detectors.

The examples mentioned above demonstrate the effectiveness of the AD database in aiding designers during the early stages of threat modeling. Thus, we strongly recommend its utilization.

5.6 BLE Prot. and Impl.-Level Attacks (AR6, WP5, ECM)

In this case study, ECM focused on Bluetooth's protocol-level and implementation-level threats on a cryptowallet (i.e., AR6). First, they tried threat modeling it using STRIDE and pytm and encountered some limitations (e.g., generic or out-of-scope threats). Then, ECM employed the ADF and constructed 46 Bluetooth ADs (i.e., `bt.yaml`) using as a main reference the awesome-bluetooth-security list [40], and TM the cryptowallet BLE module against protocol-level and implementation-level threats that are relevant and complementary. We now report verbatim the ECM report, prepared by Tommaso Sacchetti and Daniele Antonioli.

5.6.1 BLE TM with STRIDE and pytm

We wanted to investigate the efficacy of the STRIDE methodology in threat modeling and its applicability in identifying vulnerabilities in a hardware token that implements the Bluetooth Low Energy (BLE) protocol. Additionally, we tried to use the novel open-source threat modeling tool pytm from OWASP (Open Worldwide Application Security Project) [121]. The following paragraphs present

Table 5.2: STRIDE generated BLE threats. S, T, I, D, and E stands for Spoofing, Tampering, Information disclosure, DoS, and EoP

Threat	Cat.	Mitigation
Device MAC address spoofing	S	Implement strong authentication mechanisms, such as mutual authentication and encryption, to ensure device authenticity.
Malicious packet or code injection	T	Employ cryptographic measures, such as digital signatures or message authentication codes, to detect tampering and ensure data integrity.
Unauthorized access to sensitive information	I	Apply encryption to protect data confidentiality, and enforce proper access controls to limit data exposure
Packet flooding	D	Employ adaptive algorithms to dynamically adjust resource allocation based on demand
Device takeover through vulnerability exploitation	E	Regularly update firmware and apply security patches to address known vulnerabilities. Implement secure communication protocols and encryption to protect against privilege escalation.

a quick introduction to BLE, a summary of the encountered challenges and issues during the execution of the task.

BLE is a wireless communication protocol for low-power devices to establish short-range connections. It operates in the 2.4 GHz frequency band and is specifically optimized for applications that require low energy consumption. BLE enables efficient data transfer between devices, balancing transmission range, data rate, and power consumption. It provides reliable and secure communication while minimizing energy usage. BLE is the de-facto standard technology for low-power communications, and it is supported by several crypto wallets (e.g., Ledger Nano X).

Our evaluation of the STRIDE methodology revealed its limited emphasis on implementation and protocol-specific threats. The methodology primarily focuses on high-level threats that may not directly address the unique characteristics of the specific device under analysis. In the context of our case study, we found that STRIDE's high-level approach was insufficient for our needs. It did not provide the depth and granularity required to effectively identify and assess threats that may arise in the BLE component. An example of threats identified by STRIDE is shown in Table 5.2. These threats are too generic and do not cover concrete and relevant attacks, such as KNOB [7] or BLUR [8].

pytm was also problematic as its usage with communication protocols has significant limitations. pytm's pre-defined classes are not covering protocols and related chip/firmware. Moreover, pytm's threat library does not include protocol-level and implementation-level threats on BLE. We tried to extend pytm to address this gap by introducing an additional abstraction layer above the existing objects. Unfortunately, the framework primarily focuses on web-related or client-server, making it challenging to adapt effectively to other scenarios, especially to Bluetooth protocol-level or implementation-level threats. However, we believe that pytm could be expanded to support new threats. We will further investigate this possibility even if, after a brief analysis, we see that our ADF has a different level of granularity and details that make it difficult to be compatible with pytm.

Table 5.3: List of 18 BLE ADs used from our catalog of 46 ADs in `bt.yaml`. Here we report the AD name, a field, `a` part of the `tag` field. For the full ADs please look at `bt.yaml`

ad_name	a	tag
sco_ble	Downgrade attacks on BLE SCO [137]	Impl
sweyntooth_ble_1	Link Layer Length Overflow [45]	Impl
sweyntooth_ble_2	Link Layer LLID Deadlock [45]	Impl
sweyntooth_ble_3	BLE Crafted packet buffer overflow [45]	Impl
sweyntooth_ble_4	Key Size Overflow [45]	Impl
sweyntooth_ble_5	Zero LTK Installation [45]	Impl
blesa_ble	BLE reconnection spoofing [133]	Impl
bleedingbit_ble_1	Malformed packet BoF in BLE beacons parsing [111]	Impl
frankenstein_ble_1	Heap overflow in BLE PDUs parsing [104]	Impl
knob_ble	Key Negotiation of Bluetooth (KNOB) [7]	Proto
blur_ble	BLUR Cross-Transport Key Derivation attacks [8]	Proto
nino_ble	MitM on BLE SSP [51]	Proto
bluemirror_ble	Reflection attack on passkey entry [21]	Proto
invcurve_ble	Invalid Curve Attack [12]	Proto
pairing_meth_conf_ble	Method confusion attack [127]	Proto
crackle_ble	BLE Key Derivation [105]	Proto
injectable	PHY packet injection [17]	Proto
gatt_fp_ble	GATT Fingerprinting and Tracking [18]	Proto

5.6.2 BLE TM with ADF

In our case study, we are interested in the protocol-level and implementation-level threats affecting BLE in the context of the cryptowallet. Currently, in our `bt.yaml` catalog, there are 18 ADs for BLE, nine of them are protocol-level threats, while nine are implementation-level. Table 5.3 shows the list of BLE ADs with their type. We note that `bt.yaml` can also be used to TM *Bluetooth Classic (BC)* and *Bluetooth Mesh (BM)*.

Using ADs, we can address implementation-level and protocol-level threats on BLE, regardless of their level of abstraction. For example, we can model a concrete attack like No Input No Output (NINO) that is a MitM attack on BLE SSP (Secure Simple Pairing) protocol. This attack is specific to BLE and currently is not identified by STRIDE or pytm.

We use the AD in Listing 5.5 to model NINO. The AD, among others allow specifying the BLE-specific protocol phases and security mode involved in the attacks, i.e., association during LESC pairing and a high-level policy and concrete mechanism to prevent the attack, i.e., use

Listing 5.5: `nino_ble` AD

```
nino_ble:
  a: MitM on BLE SSP
  d: Out of band pairing: [Use NFC as OOB channel]
  surf: [BLE, Pairing, Association]
  vect: [No IO downgrade]
  model: [Proximity, MitM]
  tag: [Protocol, SMP, LESC]
```

Listing 5.6: `blur_ble` AD

```
blur_ble:
  a: Bluetooth Cross-Transport Key Derivation (BLUR)
  d:
    Prevent cross-transport key tampering: [Disable key overwrite with
   weaker keys]
    Enforce strong association mechanisms: [Track associations for paired
   devices and abort on downgrade request]
    Prevent role switching: [Track asymmetries in roles between BT and BLE]
  year: 2020
  surf: [BLE, Pairing, CTKD]
  vect: [Cross-transport pairing, SC downgrade, No IO downgrade]
  model: [Proximity, Impersonation, MitM, Unintended session]
  tag: [Protocol, SMP, LESC]
  risk: [TODO]
  cve: ["15802", "20361"]
  cwe: ["287"]
```

out-of-band pairing with Near Field Communication (NFC). As a result, a designer might consider adding NFC to the cryptowallet to defend against NINO and other BLE attacks related to the association phase.

Another example of a threat not identified by STRIDE and pytm is BLUR tracked by CVE-2020-15802 and CVE-2020-20361. BLUR presents attacks on Bluetooth's Cross-Transport Key Derivation (CTKD), and we model it as in Listing 5.6. As in the previous case, with the AD, we can model Bluetooth's specific aspects not captured by STRIDE and pytm, such as attacking BC from BLE and vice versa or adopting concrete mitigation strategies (e.g., disabling the possibility of overwriting keys with weaker ones, or tracking the associations with paired devices and abort when receiving downgrade requests, or prevent role switching by tracking asymmetries in the roles). Overall, we are satisfied with the ADF, and we will continue extending and using it for our Bluetooth TM exercises.

5.7 FIDO2 (AR7, WP5, SEC)

In this case study, SEC first tried to threat model the FIDO functionality of a cryptowallet (i.e., AR7) with STRIDE alone. Then, they constructed a catalog of 21 ADs for FIDO (i.e., `fido_system.yaml`, `fido_device.yaml`, and `fido_solokey.yaml`) and used them to TM the same scenario and compare the two experiences. We now report verbatim the SEC report prepared by Arianna Gringiani.

5.7.1 FIDO2 TM with STRIDE

This work aimed to study the STRIDE method for threat modeling and apply it to identify vulnerabilities of a hardware token implementing the FIDO2 protocol. This report summarises the challenges and problems faced during the task.

FIDO2 [3] is an authentication protocol designed to allow online services to offer multi-factor and single-factor authentication. A new and unique cryptographic key pair is created for each service credential in the initial registration phase. The public key is sent to the service, while the

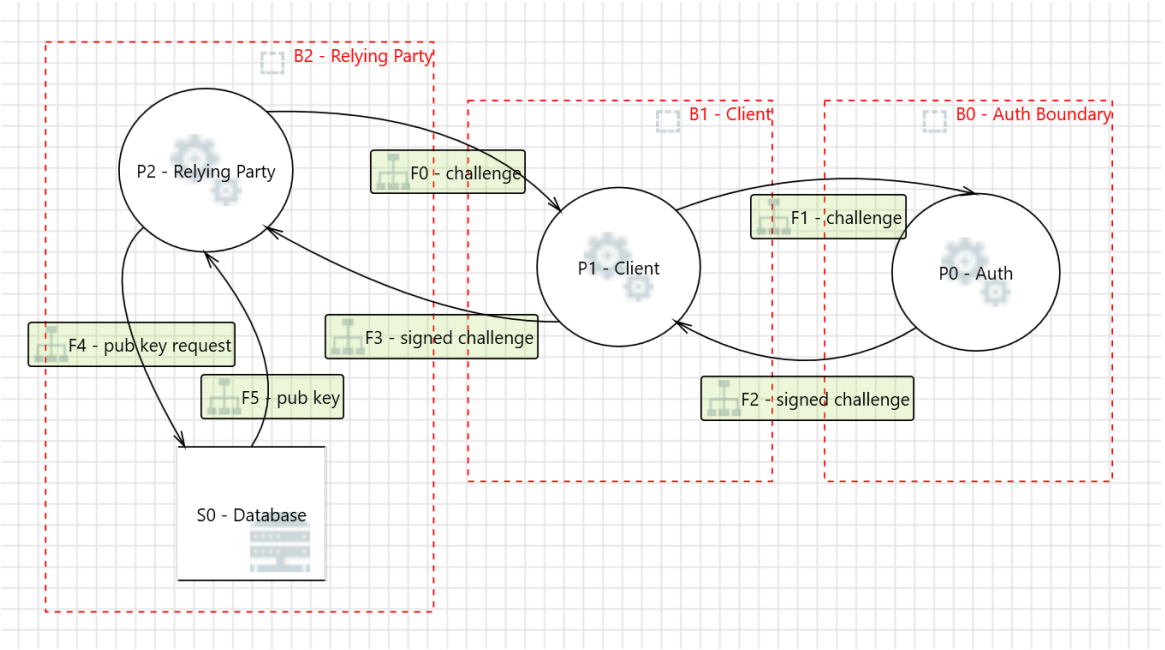

Figure 5.2: FIDO2 DFD generated by SEC

private key remains on the authenticator, which in our case, is a hardware token. The application authenticates a user through a cryptographic challenge to the token via a client API. After the user authenticates by pressing a button on the token, the client device proves possession of the private key by signing a challenge. Then it sends it back to the application, which can verify it using the corresponding stored public key.

FIDO2 DFD The first step in threat modeling is to illustrate the system with a data flow diagram, which in our case, represents the FIDO2 authentication process. The data flow diagram was created using the Microsoft threat modeling tool [70]. As shown in Figure 5.2, the main actors are the hardware authenticator, the client, and the online service (i.e., relying party). In addition, we have the relying party database containing the credentials and public keys.

As the objective was to focus on the authenticator, the first challenge was to understand if it was more convenient to represent the client and the relying party as external entities or as normal processes. To gain a better vision of the possible threats, we opted to treat model the involved entities as internal processes. Another aspect considered was whether or not to include the human user. The user was not considered, as their only interaction with the system consisted of pushing a button on the authenticator device. To capture the time dimension of the protocol as much as possible, the data flows are numbered following the FIDO2 message order [128].

STRIDE threats Starting from the data flow diagram of a system, the tool applies the STRIDE approach to generate possible threats. Tables 5.4 and 5.5 list the *48 attacks* generated from our diagram ordered by interaction, with the corresponding STRIDE category. For some threats we report a mitigation strategy suggested by the tool.

Compare STRIDE with real-world threats The next step was to research realistic attack scenarios for a FIDO2 system and compare them with the STRIDE threats automatically generated from the FIDO2 DFD. We studied documented attacks and bugs related to FIDO2 tokens and

Table 5.4: STRIDE generated FIDO2 threats (F0, F1)

Id	Title	Cat.	Int.	Mitigation
G1	P1 - Client impersonates the context of P2 - Relying Party to gain additional privilege	E	F0	
G2	Elevation by Changing the Execution Flow in P1 - Client	E	F0	
G3	P2 - Relying Party remotely executes code for P1 - Client	E	F0	
G4	Data Flow F0 - challenge Is Potentially Interrupted	D	F0	
G5	Potential Process Crash or Stop for P1 - Client	D	F0	
G6	Data Flow Sniffing	I	F0	Encrypt the data flow
G7	Potential Data Repudiation by P1 - Client	R	F0	Use logs or audits
G8	Data Flow Tampering	T	F0	Input integrity validation
G9	Spoofing the P1 - Client Process, which may lead to information disclosure by P2 - Relying Party	S	F0	Use an authentication mechanism
G10	Spoofing the P2 - Relying Party Process, which may lead to unauthorized access to P1 - Client	S	F0	Use an authentication mechanism
G11	Cross Site Request Forgery	E	F0	
G12	Data Flow Tampering	T	F1	Input integrity validation
G13	Spoofing the P0 - Auth Key Process, which may lead to information disclosure by P1 - Client	S	F1	Use an authentication mechanism
G14	Spoofing the P1 - Client Process, which may lead to unauthorised access to P0 - Auth Key	S	F1	Use an authentication mechanism
G15	Data Flow Sniffing	I	F1	Encrypt the data flow
G16	Potential Data Repudiation by P0 - Auth Key	R	F1	Use logs or audits
G17	Potential Process Crash or Stop for P0 - Auth Key	D	F1	
G18	Data Flow F1 - challenge Is Potentially Interrupted	D	F1	
G19	P0 - Auth Key impersonates the context of P1 - Client to gain additional privilege	E	F1	
G20	P1 - Client remotely executes code for P0 - Auth Key	E	F1	
G21	Elevation by Changing the Execution Flow in P0 - Auth Key	E	F1	
G22	Cross Site Request Forgery	E	F1	

integrated them with the security concerns expressed in [4]. In Table 5.6 we report 5 of the most exploited and impactful threats. For each threat, we report the closest STRIDE-generated ones.

STRIDE Impressions We can notice how real-world threats *differ* from those STRIDE generated. STRIDE describes vague and generic attacks, which simple countermeasures can mitigate. The method suggests that the potential spoofing of an entity could be solved by enabling a standard authentication mechanism. However, in threat 3 from Table 5.6, the client is controlled by the attacker (via malware), therefore a standard authentication mechanism is not sufficient to solve the issue.

The STRIDE categories being too vague relates to the fact that they are specific to a system component. Observing the system too closely by focusing on single elements or interactions could result in missing the bigger picture. Indeed, the main limitation of STRIDE is its lack of consideration for procedural threats and its limited understanding of the context and ecosystem in which a system operates.

STRIDE primarily focuses on data flow analysis, which may overlook threats that arise from specific procedural steps or the broader environment. It fails to capture the vulnerabilities and risks associated with complex processes, such as user interactions or device life cycle events. As a result, STRIDE may not provide comprehensive coverage or accurately identify threats dependent on the specific context or ecosystem.

Table 5.5: STRIDE generated FIDO2 threats (F2, F3, F4, F5)

Id	Title	Cat.	Int.	Mitigation
G23	Spoofing the P0 - Auth Key Process, which may lead to unauthorized access to P1 - Client	S	F2	Use an authentication mechanism
G24	Spoofing the P1 - Client Process, which may lead to information disclosure by P0 - Auth Key	S	F2	Use an authentication mechanism
G25	Data Flow Tampering	T	F2	Input integrity validation
G26	Potential Data Repudiation by P1 - Client	R	F2	Use logs or audits
G27	Data Flow Sniffing	I	F2	Encrypt the data flow
G28	Potential Process Crash or Stop for P1 - Client	D	F2	
G29	Data Flow F2 - signed challenge Is Potentially Interrupted	D	F2	
G30	P1 - Client impersonates the context of P0 - Auth Key to gain additional privilege	E	F2	
G31	P0 - Auth Key remotely executes code for P1 - Client	E	F2	
G32	Elevation by Changing the Execution Flow in P1 - Client	E	F2	
G33	Cross Site Request Forgery	E	F2	
G34	P2 - Relying Party impersonates the context of P1 - Client to gain additional privilege	E	F3	
G35	Spoofing the P1 - Client Process, which may lead to unauthorised access to P2 - Relying Party	S	F3	Use an authentication mechanism
G36	Spoofing the P2 - Relying Party Process, which may lead to information disclosure by P1 - Client	S	F3	Use an authentication mechanism
G37	Data Flow Tampering	T	F3	Input integrity validation
G38	Potential Data Repudiation by P2 - Relying Party	R	F3	Use logs or audits
G39	Data Flow Sniffing	I	F3	Encrypt the data flow
G40	Potential Process Crash or Stop for P2 - Relying Party	D	F3	
G41	Data Flow F3 - signed challenge Is Potentially Interrupted	D	F3	
G42	P1 - Client remotely executes code for P2 - Relying Party	E	F3	
G43	Elevation by Changing the Execution Flow in P2 - Relying Party	E	F3	
G44	Cross Site Request Forgery	E	F3	
G45	Spoofing of Destination Data Store S0 - Database	S	F4	Use an authentication mechanism
G46	Potential Excessive Resource Consumption for P2 - Relying Party or S0 - Database	D	F4	
G47	Weak Access Control for a Resource	I	F5	
G48	Spoofing of Source Data Store S0 - Database	S	F5	Use an authentication mechanism

Moreover, considering six threat categories for each component results in many potentially overlapping attacks. Although our real-world evaluation is relatively simple with only four entities, the tool generates 48 threat scenarios, many of which are repeated or similar to each other (e.g., G2-G32 or G9-G35). This explosion of cases slows down and overcomplicates the study.

In the attempt to link our threats with the STRIDE-generated ones, we found each real-world threat to be related to multiple attacks from the first table. However, STRIDE does *not* allow categorizing a threat into multiple classes. For threats 4 and 5, we could not find a correspondence, as STRIDE focuses on software; thus, it does not consider physical hardware attacks. On the other hand, deriving the real-world table from the STRIDE ones is more difficult due to the previously mentioned STRIDE issues.

In [116] the author stresses that STRIDE should *not* be used as a categorization method, but solely as an exercise to help find threats. Shostack's idea is that once an attack has been identified, there is no need to waste effort in putting it in one of the six STRIDE categories.

Table 5.6: Instances of FIDO real-world threats with STRIDE correspondence

Id	Title	Related STRIDE threats
1	A malicious device can intercept and modify communication between Client and authenticator exploiting misconfigurations in USB or BLE protocols	G15, G27
2	MitM attack to the Transport Layer Security (TLS) session between Client and Relying Party	G1, G8, G9, G10, G34, G35, G36, G37
3	Attacker executes code for the Client exploiting malware or compromised browser extensions and mounts a relay attack	G2, (G32)
4	Manipulations of the device occur during the supply or distribution chain	G21
5	Evil maid attack to extract credentials private key on the Authenticator through side-channel attacks (fault injection, timing, etc)	G21

However, we found this reasoning to be confusing: if it is not clear to which STRIDE category a threat belongs, or if it falls under multiples, it may be difficult to come up with that threat in the first place. Also, in case the threat modeler succeeds in identifying a miscellaneous threat, not being able to univocally categorize it may lead to confusion about the validity of their thought process.

Overall, we found STRIDE to be a valuable tool for a first-time approach of the threat modeling world as it gives a simple vision of the most common attack strategies. However, the method is too plain to analyze complex scenarios, thus the need to develop more elaborate solutions.

5.7.2 FIDO2 TM with ADF

Our first exercise consisted of studying the STRIDE method for threat modeling and attempting to apply it to identify vulnerabilities of a hardware token implementing the FIDO2 protocol. We reported the challenges and limitations of STRIDE in Section 5.7.1. Afterward, we applied the ADF to construct a threat library (i.e., ADs in YAML format) for our case study.

We divided the AD library according to *three* levels of abstraction: system-level threats (`fido_system.yaml`), device-level threats (`fido_device.yaml`) and those related to the Solo FIDO2 token by SoloKeys [32] (`fido_solokey.yaml`).

System-level threats System-level threats are high-level attacks concerning the FIDO2 ecosystem where a hardware authenticator, a client, and a relying party interact. Such threats come from FIDO security references and the generalization of specific attacks. Some mentions of identified ADs:

1. A Man in The Middle attack between client and authenticator, and between the relying party and the client. Corruption/spoofing of the client, or related to the relying party app, on the user device. For certain policies, we specified concrete mechanisms implemented by certain authenticators to mitigate these risks, such as a transaction confirmation message allowing the user to identify the relying party correctly.
2. Side-channel attack on the authenticator. This high-level AD presents various defense policies such as 'robust device' or 'secure microcontroller', for which concrete mechanism specifications are left to particular tokens. We associated this AD with various attack vectors describing the possible types of side-channel analysis.

3. Malicious relying party mounts a cryptographic attack on key handles.
4. Manipulations of the device occur during the supply or distribution chain.

We attempted to link these generic attacks with one or more CWE and CAPEC, while the attack surfaces remain high-level (authenticator, client, and relying party). We previously attempted to model these types of threats using the STRIDE method. Some of these high-level vulnerabilities can be easily traced back to the STRIDE categories, such as man-in-the-middle attacks and spoofing issues from point 1. However, it is unlikely that side-channel attacks will come up within a STRIDE approach, mainly because the method does not consider the hardware world, which constitutes a significant limitation.

Device-level threats Device-level threats refer to actual vulnerabilities discovered in one or more models of hardware tokens, coming from reported CVEs and other online articles and documentation. The majority of those are side-channel or fault-injection attacks to the token device, which exploit vulnerabilities in microcontrollers, secure elements, and other components, the communication among them, or generic implementation issues specific to a model of the token. Other threats concern WebAuthN [128] implementation vulnerabilities and bugs, which is the protocol for communications between the client and relying party. It is clear how STRIDE struggles to help identify these specific threats due to the generality of the method and, once again, its lack of focus on hardware issues.

SoloKeys threats Lastly, we focused on Solo, an open-source FIDO2 token. We reported physical threats associated with a CVE or documented in the SoloKeys online blog [31]. By leveraging the AD structure, we assessed whether some previously documented threats applied to the token. The policies of the generic side-channel AD can now have a concrete mechanism. For instance, under 'secure microcontroller', we specified the name of the microcontroller used by SoloKeys and the security measures implemented by it. Concerning the specific threats, many of them target components not present in the token design, and in one case, we found that the token employs a time-insensitive key derivation function to mitigate timing SC attacks.

FIDO2: ADF vs. STRIDE

While STRIDE can be a helpful method for identifying generic threats in software systems, it does not cover all possible threats and does not provide specific mitigations tailored to a system's unique characteristics. Additionally, its software-centric approach limits its applicability to systems where hardware vulnerabilities are a concern. The possibility of utilizing a pre-existing ADs library for TM holds significant promise in the future. This approach would offer distinct advantages over using STRIDE alone. The ADs *flexible* and *rich* data model allows for more precise and relevant analysis and offers a valuable tool for enhancing system security.

Carefully compiling and organizing information about attacks within the AD object allows for filtering threats effectively. The AD object captures detailed data about attacks, including surfaces, vectors, techniques, and models, enabling precise threat identification and analysis. In practical terms, the AD method enables developers to make informed decisions when selecting system components. By referencing the AD object's repository of attacks, developers can identify components or behaviors that have previously been targeted or exploited. This information empowers them to implement appropriate defenses or choose more secure alternatives.

The AD object's inclusion of optional and extensible fields, such as CVE, CWE, and CAPEC

identifiers facilitate easy access to information about known vulnerabilities and attack patterns. Indeed, while these identifiers provide valuable information about known vulnerabilities and attack patterns, they are not exhaustive and may not encompass all possible threats. Also, enabling a hierarchical structure of layered ADs can offer a holistic perspective on the threat landscape. It facilitates the identification of complex attack patterns and assesses their potential impact on different system layers or components.

Compiling AD for a specific context can ultimately involve multiple stakeholders distributing the effort. The resulting AD becomes reusable, effectively addressing trivial threats with existing mitigations. This approach directs the focus toward critical threats that require unique solutions. By leveraging shared efforts and established mitigations, the AD framework optimizes resource allocation, prioritizing identifying and mitigating high-impact threats. This approach has the potential to greatly enhance the accuracy and applicability of the threat modeling process, surpassing the limitations of generic approaches like STRIDE.

Related Work

In this Chapter, we comment on research work related to ADF. We cover TM methodologies, real-world examples, automation tools, and domain-specific case studies. We also discuss some relevant work in the areas of threat intelligence and process security.

TM Methodologies

Historically, fault trees can be considered the first TM methodology, where each threat represents a failure, and failures are hierarchically represented in a tree [126]. ATree were extended, among others, with attack-defense-trees [63], profiles [65], and case-study driven methodologies [10]. STRIDE was also extended with STRIDE per element and per interaction [122]. ATree were augmented with formal methods [130]. Another popular methodology is called PASTA (Process for Attack Simulation and Threat Analysis) that combines threat identification, modeling, scoring, remediation, and simulation using a seven-stage approach [125].

There have been industrial efforts to standardize threat modeling across platforms such as the OTM (Open Threat Model) format by IriusRisk [55] and MITRE's ATT&CK, an adversary-centric framework designed to threat model post-compromise enterprise security, including lateral movement, and privilege escalation [74, 75]. The framework is in constant expansion and currently includes attack techniques and sub-techniques for enterprise, mobile, and industrial systems. Recently, the US Cybersecurity and Infrastructure Security Agency (CISA) released Decider, an open-source web application to help map a threat to ATT&CK [20]. MITRE also published D3FEND [58, 81, 82], the defensive counterpart of the ATT&CK framework. Best practices related to TM methodologies were also created, such as OWASP's threat modeling project [98], and collections of survey and comparison of different TM methodologies [113, 112]. Recently, we have seen attempts to use a large language model (LLM), like Generative Pre-trained Transformer 4 (GPT-4), and Pathway Language Model (PaLM), to aid threat modeling with moderate success [117, 123, 87].

Vendors also implemented their own TM methodologies. Intel proposed TARA (Threat Agent Risk Assessment), an attack-centric methodology based on seven steps. TARA is currently unmaintained [110], but is still popular in some sectors, such as automotive. Lockheed Martin proposed the Cyber Kill Chain (CKC) [67] to model cyber intrusion activities, like advanced persistent threats (APT). The chain has seven steps: reconnaissance, weaponization, delivery, exploitation, installation, command and control, and actions.

Card games were also developed to *gamify* threat modeling, notable examples are Elevation of Privilege [115], Control-Alt-Hack™ [96], LINDDUN GO [66], and Cornucopia [41]. These games are especially useful to engage newcomers.

TM real-world examples

There are few real-world threat modeling exercises freely available [49]. In 2018, researchers quantitatively measured the efficacy of threat modeling in a real-world enterprise scenario (i.e., the New York City Cyber Command (NYC3) which is the agency defending New York City from cyber threats [118]). In 2019, Trail of Bits did a comprehensive security audit of Kubernetes and released the produced threat model and related pytm code [95]. OWASP also maintains a list of threat model examples with the Threat Model Cookbook project [97], including attack trees and flow diagrams.

TM automation tools

There are several (open-source) TM automation tools [120]. These tools allow, among others, to parse threats from code comments (e.g., Threatspec [34]), build system models and threats catalogs from code or configuration languages (e.g., Pytm [121], hcltm [43], threagile [107], and Threat Dragon [100]), and speed up TM using agile best practices, e.g., Rapid Threat Model Prototyping (RTMP) [50]. Moreover, there are commercial TM tools such as the ones provided by IriusRisk [54] and Tutamantic [124]. The most popular closed-source but free TM tool is Microsoft's Threat Modeling Tool (TMT) [70] that natively supports STRIDE.

Domain-Specific TM

Prior work also performed domain-specific threat modeling using (and extending) one or more TM methodologies. For example, in [59] the authors show how to adapt STRIDE and TARA to threat model a connected car adhering to the AUTOSAR standard [44]. The NCC Group extended the MS TMT with a template for automotive TM [86].

Other domain-specific areas of TM extension are cyber-physical system (CPS) [61], industrial control systems (ICS) [5], Internet of Things (IoT) [1] and mobile (cellular) networks [103]. Recently, six popular end-to-end messenger desktop clients STRIDE and LINDDUN threat models were evaluated along the space and time dimensions [19].

Threat intelligence

Actual incidents are collected using threat intelligence platforms that can help threat modeling with real world data. The Malware Information Sharing Platform (MISP) is an open-source threat intelligence platform born out of an academic effort and now used by the industry [129, 102]. MISP enables, among others, to store, share, collaborate on cyber security Indicators of Compromise (IoC), malware analysis, and also to use IoCs to detect and prevent attacks. There are other useful free and open source projects related to threat intelligence, such as Open Cyber Threat Intelligence (OpenCTI) [101], Structured Threat Information Expression (STIX) [53], Threat Report ATT&CK Mapper (TRAM) [42], and TheHive incident response platform [33].

Process security

Most of process security efforts so far focused on the hardware and software *supply chains*. Researchers extensively analyzed vulnerable dependencies from package repositories for interpreted programming language, like Node package manager (Npm), Python Package Index (PyPI), and RubyGems [38] and extracted valuable security lessons from the software supply

chain [39]. Other works developed attack taxonomies for open source software (OSS) via attack trees [64] and uncovered new attack techniques via the software supply chain, such as the GitHub fork attack vector [16]. Recently, some works started exploring automated ways to analyze the security of closed-source software supply chains [11].

Conclusion

This document presents the design, implementation, and evaluation of ADF, a framework we developed to address Task 2.2 and aid state-of-the-art threat modeling techniques. We presented its design requirements, the AD object building block, flat and hierarchical representations, and how it can be used to enhance STRIDE, LINDDUN, and ATree threat identifications. We show how we implemented it with `ADF`, a toolkit we will open-source with a permissive license and extend in the future. Currently, `ADF` has four modules: Catalog, Parse, Check, and Analyze, usable to create and process ADs. Next, we show how to exploit the ADF using a reference scenario where we are asked to develop a new cryptowallet and satisfy seven abstract requirements. We successfully evaluate the ADF in seven case studies related to the AR mentioned earlier. We now draw some conclusions based on our work.

> WP2 - Task 2.2: "Security requirements for a product to be built are typically abstract or complex. In this task, we develop a mapping of these abstract security requirements into concrete policies for the life cycle phases and into concrete security requirements for the components, the device will be composed of. These policies and concrete security requirements will drive the research in WP3, WP4 and WP5."

The ADF successfully addresses the two goals we set in Task 2.2. First, it enables to map abstract security and privacy requirements to ADs modeling high-level and low-level attacks and defenses on the TLC and its related products. For instance, in Chapter 4 we map seven AR to a cryptowallet build with the TLC, and we evaluate the related ADs in Chapter 5.

Second, ADF is usable in WP3 (Models for formal verification), WP4 (Effective security audits), and WP5 (Secure auth and comms). These three technical WPs are already benefiting from ADF as demonstrated in Chapter 5, where we use ADF in TM scenarios related to SC, FI and microarchitectural speculative execution attacks (WP3), presilicon and invasive physical attacks on hardware (WP4), and attacks on inter (i.e., BLE) and intra (i.e., FIDO) device communication protocols (WP5).

Task 2.2 produced valuable and novel contributions aligned with what we developed in Task 2.1. For example, we used the Trusted Life Cycle (TLC), the main contribution of T2.1, to build process-specific ADs and set requirements for the process. These ADs catalog might be used as blueprints to develop a device via a *trustable* life cycle. For example, given ADs covering supply chain attacks on a life cycle, a user of such a life cycle can trust its resilience against a measurable set of supply chain attacks.

Task 2.2 generated added value for the ORSHIN *industrial* partners. SEC was capable of translating a state-of-the-art process standard for which it is certified (i.e., ISA/IEC 62443-4-1) into a set of actionable AD objects to threat model a process. This is a novel application of TM that we enabled with ADF. Moreover, they managed to represent real-world FIDO threats (i.e., system, device, and token level) not captured by STRIDE. NXP modeled for the first time

presilicon attacks and defenses on one of their RISC-V secure cores. TXP states that ADF is particularly useful for invasive physical attacks as there is no TM library about this threat category and building the ADs enabled them to create a self-documenting catalogs that can be (re)used *across* TM exercises *regardless of* the user level of expertise with the topic.

We also got valuable feedback from the ORSHIN *academic* partners. KUL (COSIC) managed to take advantage of the AD object flexibility to model physical SC and FI attack techniques even in cases where no defenses exist. KUL (DistriNet) believes that the ADF could be used to achieve completeness, reproducibility, and speed-ups during TM. ECM used the ADF to model Bluetooth, a complex communication standard with a huge attack surface (e.g., protocol-level and implementation-level attacks).

We will continue improving the ADF and submit it as a core contribution in a *research paper*. We will improve the data model based on the excellent feedback we got from the case studies (e.g., better guidance on how to create and manage ADs with different layers of abstraction). We will work on a new module capable of (semi)automatically generating ADs from well-structured sources. For example, we will take advantage of the top ten CWE list for hardware and their related CVEs and CAPECs to generate a dedicated ADs catalog. Moreover, we will develop a mechanism to (semi)automatically score the risk of ADs via their `risk` field and based on context-specific information. For example, the same AD can be scored differently according to the TM scenario. Furthermore, we will take advantage of the AD categories that we defined in Section 2.2 (e.g., Security or Privacy, Product or Process) and encode them in the AD object. For example, we can add an optional list of strings storing the AD categories such as `adtype = [cat1, cat2, cat3]`.

Bibliography

[1] Ioannis Agadakos, Chien-Ying Chen, Matteo Campanelli, Prashant Anantharaman, Monowar Hasan, Bogdan Copos, Tancrède Lepoint, Michael Locasto, Gabriela F Ciocarlie, and Ulf Lindqvist. Jumping the air gap: Modeling cyber-physical attack paths in the internet-of-things. In *Proceedings of the 2017 workshop on cyber-physical systems security and privacy*, pages 37–48, 2017.

[2] FIDO Alliance. FIDO: Simpler, Stronger Authentication. https://fidoalliance.org/.

[3] FIDO Alliance. FIDO2. https://fidoalliance.org/fido2/.

[4] FIDO Alliance. FIDO Security reference. https://fidoalliance.org/specs/common-specs/fido-security-ref-v2.1-ps-20220523.pdf, 2021.

[5] Luca Allodi and Sandro Etalle. Towards realistic threat modeling: attack commodification, irrelevant vulnerabilities, and unrealistic assumptions. In *Proceedings of the 2017 Workshop on Automated Decision Making for Active Cyber Defense*, pages 23–26, 2017.

[6] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. BIAS: Bluetooth Impersonation AttackS. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2020.

[7] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. *ACM Transactions on Privacy and Security (TOPS)*, 23(3):1–28, 2020.

[8] Daniele Antonioli, Nils Ole Tippenhauer, Kasper Rasmussen, and Mathias Payer. BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy. In *Proceedings of the Asia conference on computer and communications security (ASIACCS)*, May 2022.

[9] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B Rasmussen. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1047–1061, 2019.

[10] Maxime Audinot, Sophie Pinchinat, and Barbara Kordy. Is my attack tree correct? In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I 22*, pages 83–102. Springer, 2017.

[11] Frederick Barr-Smith, Tim Blazytko, Richard Baker, and Ivan Martinovic. Exorcist: Automated differential analysis to detect compromises in closed-source software supply chains. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, pages 51–61, 2022.

[12] Eli Biham and Lior Neumann. Breaking the Bluetooth pairing–the fixed coordinate invalid curve attack, 2020.

[13] Martín Blech. xmltodict - Makes working with XML feel like you are working with JSON. https://github.com/martinblech/xmltodict.

[14] Matteo Cagnazzo, Markus Hertlein, Thorsten Holz, and Norbert Pohlmann. Threat modeling for mobile health systems. In *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 314–319. IEEE, 2018.

[15] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. A systematic evaluation of transient execution attacks and defenses. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 249–266. USENIX Association, 2019.

[16] Alan Cao and Brendan Dolan-Gavitt. What the fork? finding and analyzing malware in github forks. In *Proc. of NDSS*, volume 22, 2022.

[17] Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, and Géraldine Marconato. InjectaBLE: Injecting malicious traffic into established Bluetooth Low Energy connections. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 388–399. IEEE, 2021.

[18] Guillaume Celosia and Mathieu Cunche. Fingerprinting Bluetooth Low Energy devices based on the generic attribute profile. In *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, pages 24–31, 2019.

[19] Partha Das Chowdhury, Maria Sameen, Jenny Blessing, Nicholas Boucher, Joseph Gardiner, Tom Burrows, Ross Anderson, and Awais Rashid. Threat Models over Space and Time: A Case Study of E2EE Messaging Applications. *arXiv preprint arXiv:2301.05653*, 2023.

[20] CISA. Decider web application. https://github.com/cisagov/Decider/.

[21] Tristan Claverie and José Lopes Esteves. Bluemirror: reflections on Bluetooth pairing and provisioning protocols. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 339–351. IEEE, 2021.

[22] Mina Deng, Kim Wuyts, Riccardo Scandariato, Bart Preneel, and Wouter Joosen. A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. *Requirements Engineering*, 16(1):3–32, 2011.

[23] Graphviz developers. Graphviz is an open source graph visualization software. https://graphviz.org/.

[24] Graphviz developers. Package facilitating the creation and rendering of graph descriptions in the DOT language of Graphviz. https://pypi.org/project/graphviz/.

[25] LibYAML developers. LibYAML - A C library for parsing and emitting YAML. https://github.com/yaml/libyaml.

[26] Linux Kernel Developers. The Kernel Address Sanitizer (KASAN). https://www.kernel.org/doc/html/latest/dev-tools/kasan.html.

[27] Linux Kernel Developers. The Kernel Memory Sanitizer (KMSAN). https://www.kernel.org/doc/html/latest/dev-tools/kmsan.html.

[28] Pandas developers. Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. https://pandas.pydata.org/.

[29] PyYAML developers. PyYAML is a full-featured YAML framework for the Python programming language. https://pyyaml.org/.

[30] Rust developers. Rust: A language empowering everyone to build reliable and efficient software. https://www.rust-lang.org/.

[31] SoloKeys developers. Solokeys blog. https://solokeys.com/blogs/news.

[32] SoloKeys developers. Solokeys homepage. https://solokeys.com/.

[33] TheHive developers. TheHive is a FOSS security incident response platform. https://github.com/TheHive-Project/TheHive.

[34] Threatspec developers. Threatspec - continuous threat modeling, through code. https://github.com/threatspec/threatspec.

[35] Wordcloud developers. A little word cloud generator in Python. https://pypi.org/project/wordcloud/.

[36] Matplotlib development team. Matplotlib: Visualization with Python. https://matplotlib.org/.

[37] Guardian Digital. Linux Security Advisories. https://linuxsecurity.com/advisories.

[38] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. Towards measuring supply chain attacks on package managers for interpreted languages. *arXiv preprint arXiv:2002.01139*, 2020.

[39] William Enck and Laurie Williams. Top five challenges in software supply chain security: Observations from 30 industry and government organizations. *IEEE Security & Privacy*, 20(2):96–100, 2022.

[40] engn33r. Awesome Bluetooth Security (BR, EDR, LE, and Mesh). https://github.com/engn33r/awesome-bluetooth-security.

[41] Darío De Filippis and OWASP. Cornucopia card game. https://shostack.org/games/elevation-of-privilege.

[42] The Center for Threat-Informed Defense. Threat Report ATT&CK Mapping (TRAM). https://github.com/center-for-threat-informed-defense/tram.

[43] Christian Frichot. hcltm: Threat Modeling with HCL. https://github.com/xntrik/hcltm.

[44] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkelin, Kenji Nishikawa, and Klaus Lange. Autosar–a worldwide standard is on the road. In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, volume 62, page 5. Citeseer, 2009.

[45] Matheus E Garbelini, Chundong Wang, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. Sweyntooth: Unleashing mayhem over bluetooth low energy. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, pages 911–925, 2020.

[46] Google. Android Security Bulletins. https://source.android.com/docs/security/bulletin.

[47] Marit Hansen, Meiko Jensen, and Martin Rost. Protection goals for privacy engineering. In *2015 IEEE Security and Privacy Workshops*, pages 159–166. IEEE, 2015.

[48] MG Hardy. Beyond continuous monitoring: Threat modeling for real-time response. *SANS Institute*, 2012.

[49] Geoffrey Hill. Awesome Threat Modeling. https://github.com/geoffrey-hill-tutamantic/awesome-threat-modelling.

[50] Geoffrey Hill. Rapid Threat Model Prototyping (RTMP). https://github.com/geoffrey-hill-tutamantic/rapid-threat-model-prototyping-docs.

[51] Konstantin Hypponen and Keijo MJ Haataja. "Nino" Man-in-the-Middle attack on Bluetooth Secure Simple Pairing. In *2007 3rd IEEE/IFIP International Conference in Central Asia on Internet*, pages 1–5. IEEE, 2007.

[52] Securin Inc. Pegasus Spyware Snoops on Political Figures Worldwide. https://www.securin.io/articles/pegasus-spyware-snoops-on-political-figures-worldwide/.

[53] OASIS Cyber Threat Intelligence. Sharing threat intelligence just got a lot easier! https://oasis-open.github.io/cti-documentation/.

[54] IriusRisk. IriusRisk is the industry leader in Automated threat modeling and secure software design. https://www.iriusrisk.com/.

[55] IriusRisk. The Open Threat Modeling Format (OTM) defines a platform independent way to define the threat model of any system. https://github.com/iriusrisk/OpenThreatModel.

[56] ISA/IEC. ISA/IEC 62443 Series of Standards. https://www.isa.org/standards-and-publications/isa-standards/isa-iec-62443-series-of-standards.

[57] Pontus Johnson, Robert Lagerström, and Mathias Ekstedt. A meta language for threat modeling and attack simulations. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–8, 2018.

[58] Peter E Kaloroumakis and Michael J Smith. Toward a knowledge graph of cybersecurity countermeasures. *The MITRE Corporation*, page 11, 2021.

[59] Adi Karahasanovic, Pierre Kleberger, and Magnus Almgren. Adapting threat modeling methods for the automotive industry. In *Proceedings of the 15th ESCAR Conference*, pages 1–10, 2017.

[60] Vladimir Keleshev. Schema validation just got Pythonic. https://github.com/keleshev/schema.

[61] Rafiullah Khan, Kieran McLaughlin, David Laverty, and Sakir Sezer. STRIDE-based threat modeling for cyber-physical systems. In *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pages 1–6. IEEE, 2017.

[62] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1–19. IEEE, 2019.

[63] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of attack–defense trees. In *Formal Aspects of Security and Trust: 7th International Workshop, FAST 2010, Pisa, Italy, September 16-17, 2010. Revised Selected Papers 7*, pages 80–95. Springer, 2011.

[64] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. Taxonomy of attacks on open-source software supply chains. *arXiv preprint arXiv:2204.04008*, 2022.

[65] Aleksandr Lenin, Jan Willemson, and Dyan Permata Sari. Attacker profiling in quantitative security assessment based on attack trees. In *Secure IT Systems: 19th Nordic Conference, NordSec 2014, Tromsø, Norway, October 15-17, 2014, Proceedings 19*, pages 199–212. Springer, 2014.

[66] KU Leuven. LINDDUN GO: Lean team approach to privacy threat modeling. https://linddun.org/go/, 2023.

[67] Lockheed Martin. The Cyber Kill Chain. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html, 2022.

[68] Microsoft. Microsoft Security Development Lifecycle (SDL). https://www.microsoft.com/en-us/securityengineering/sdl/.

[69] Microsoft. Microsoft Threat Modeling Tool threats. https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model.

[70] Microsoft. Microsoft Threat Modeling Tool (TMT). https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool.

[71] Microsoft. The DREAD approach to threat assessment. https://learn.microsoft.com/en-us/windows-hardware/drivers/driversecurity/threat-modeling-for-drivers.

[72] Mitre. 2021 CWE Most Important Hardware Weaknesses. https://cwe.mitre.org/scoring/lists/2021_CWE_MIHW.html.

[73] Mitre. 2022 CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html.

[74] Mitre. ATT&CK framework. https://attack.mitre.org/.

[75] Mitre. ATT&CK framework GitHub project. https://github.com/mitre-attack.

[76] Mitre. CAPEC-667: Bluetooth Impersonation AttackS (BIAS). https://capec.mitre.org/data/definitions/667.html.

[77] Mitre. CAPEC-668: Key Negotiation of Bluetooth Attack (KNOB). https://capec.mitre.org/data/definitions/668.html.

[78] Mitre. Common Attack Pattern Enumerations and Classifications. https://capec.mitre.org/.

[79] Mitre. Common Vulnerabilities and Exposures. https://www.cve.org/.

[80] Mitre. Common Weakness Enumeration. https://cwe.mitre.org/.

[81] Mitre. D3FEND framework. https://d3fend.mitre.org/.

[82] Mitre. D3FEND framework GitHub project. https://github.com/d3fend.

[83] Michael Muckin and Scott C Fitch. A threat-driven approach to cyber security. *Lockheed Martin Corporation*, 2014.

[84] musl developers. musl libc for Linux git repository. https://git.musl-libc.org/cgit/musl.

[85] musl developers. musl libc for Linux homepage. https://musl.libc.org/.

[86] nccgroup. The Automotive Threat Modeling Template. https://github.com/nccgroup/The_Automotive_Threat_Modeling_Template, 2016.

[87] Rusty Newton. Threat Modeling Example with ChatGPT. https://blog.infosec.business/how-to-use-chatgpt-to-learn-threat-modeling/, 2023.

[88] NIST. An Introduction to Privacy Engineering and Risk Management in Federal Systems. https://csrc.nist.gov/publications/detail/nistir/8062/final.

[89] NIST. Common Vulnerability Scoring System (CVSS). https://nvd.nist.gov/vuln-metrics/cvss.

[90] NIST. CVSS Version 2 Calculator. https://nvd.nist.gov/vuln-metrics/cvss/v2-calculator.

[91] NIST. CVSS Version 2 Calculator. https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator.

[92] NIST. CVSS Version 4. https://www.first.org/cvss/v4-0.

[93] NIST NVD. CVE-2019-9506 Detail (KNOB). https://nvd.nist.gov/vuln/detail/CVE-2019-9506.

[94] NIST NVD. CVE-2020-10135 Detail (BIAS). https://nvd.nist.gov/vuln/detail/CVE-2020-10135.

[95] Trail of Bits. Trail of Bits Audit of Kubernetes. https://github.com/trailofbits/audit-kubernetes/tree/master.

[96] University of Washington. Control-Alt-Hack™ card game. http://www.controlalthack.com/, 2010.

[97] OWASP. OWASP Threat Model Cookbook Project. https://github.com/OWASP/threat-model-cookbook.

[98] OWASP. OWASP Threat Modeling Project. https://owasp.org/www-project-threat-model.

[99] OWASP. OWASP Top Ten. https://owasp.org/www-project-top-ten/, 2021.

[100] OWASP and Mike Goodwin. Threat Dragon is a free, open-source, cross-platform threat modeling application. https://github.com/OWASP/threat-dragon.

[101] OpenCTI Platform. OpenCTI allows orgs to manage cyber threat intelligence knowledge and observables. https://github.com/OpenCTI-Platform/opencti.

[102] MISP Project. MISP - Threat Intelligence Sharing Platform. https://github.com/MISP/MISP.

[103] Siddharth Prakash Rao, Hsin-Yi Chen, and Tuomas Aura. Threat modeling framework for mobile communication systems. *Computers & Security*, 125:103047, 2023.

[104] Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick. Frankenstein: Advanced wireless fuzzing to exploit new bluetooth escalation targets. In *Proceedings of the 29th USENIX Conference on Security Symposium*, pages 19–36, 2020.

[105] Mike Ryan. Crack and decrypt BLE encryption. https://github.com/mikeryan/crackle, Accessed: 2019-07-30, 2019.

[106] Chris Salter, O Sami Saydjari, Bruce Schneier, and Jim Wallner. Toward a secure system engineering methodolgy. In *Proceedings of the 1998 workshop on New security paradigms*, pages 2–10, 1998.

[107] Christian Schneider. Threagile is an open-source toolkit for agile threat modeling:. https://github.com/Threagile/threagile.

[108] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[109] Bruce Schneier. Chinese Supply-Chain Attack on Computer Systems. https://www.schneier.com/blog/archives/2021/02/chinese-supply-chain-attack-on-computer-systems.html, 2021.

[110] Intel Security. Prioritizing Information Security Risks with Threat Agent Risk Assessment. https://media10.connectedsocialmedia.com/intel/10/5725/Intel_IT_Business_Value_Prioritizing_Info_Security_Risks_with_TARA.pdf, 2009.

[111] Ben Seri, Gregory Vishnepolsky, and Dor Zusman. BLEEDINGBIT: The hidden Attack Surface within BLE chips, 2019.

[112] Nataliya Shevchenko. Threat Modeling: 12 Available Methods. https://insights.sei.cmu.edu/blog/threat-modeling-12-available-methods/, 2018.

[113] Nataliya Shevchenko, Timothy A Chick, Paige O'Riordan, Thomas P Scanlon, and Carol Woody. Threat modeling: a summary of available methods. Technical report, Carnegie Mellon University Software Engineering Institute Pittsburgh United . . . , 2018.

[114] Adam Shostack. Experiences Threat Modeling at Microsoft. 2008.

[115] Adam Shostack. Elevation of Privilege card game. https://shostack.org/games/elevation-of-privilege, 2010.

[116] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[117] Adam Shostack. More on GPT-3 and threat modeling. https://shostack.org/blog/more-on-gpt3/, 2023.

[118] Rock Stevens, Daniel Votipka, Elissa M Redmiles, Colin Ahern, Patrick Sweeney, and Michelle L Mazurek. The battle for new york: A case study of applied digital threat modeling at the enterprise level. In *USENIX Security Symposium*, pages 621–637, 2018.

[119] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 48–62. IEEE Computer Society, 2013.

[120] Kristen Tan and Vaibhav Garg. An analysis of open-source automated threat modeling tools and their extensibility from security into privacy. 2022.

[121] Izar Tarandach. pytm: A Pythonic framework for threat modeling. https://github.com/izar/pytm.

[122] Izar Tarandach and Matthew J Coles. *Threat modeling: a practical guide for development teams*. O'Reilly, 2021.

[123] Google Cloud Security Team. Sec-PaLM: Supercharging security with generative AI. https://cloud.google.com/blog/products/identity-security/rsa-google-cloud-security-ai-workbench-generative-ai, 2023.

[124] Tutamantic. Tutamen Threat Model Automator. https://www.tutamantic.com/.

[125] Tony UcedaVelez and Marco M Morana. *Risk Centric Threat Modeling: process for attack simulation and threat analysis*. John Wiley & Sons, 2015.

[126] William E Vesely, Francine F Goldberg, Norman H Roberts, and David F Haasl. Fault tree handbook. Technical report, Nuclear Regulatory Commission Washington DC, 1981.

[127] Maximilian von Tschirschnitz, Ludwig Peuckert, Fabian Franzen, and Jens Grossklags. Method confusion attack on Bluetooth pairing. In *2021 IEEE symposium on security and privacy (SP)*, pages 1332–1347. IEEE, 2021.

[128] W3C. Web Authentication: An API for accessing Public Key Credentials - Level 2. https://www.w3.org/TR/webauthn/, 2021.

[129] Cynthia Wagner, Alexandre Dulaunoy, Gérard Wagener, and Andras Iklody. Misp: The design and implementation of a collaborative threat intelligence sharing platform. In *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*, pages 49–56. ACM, 2016.

[130] Wojciech Wideł, Maxime Audinot, Barbara Fila, and Sophie Pinchinat. Beyond 2014: Formal methods for attack tree–based security modeling. *ACM Computing Surveys (CSUR)*, 52(4):1–36, 2019.

[131] Jake Williams. What You Need to Know About the SolarWinds Supply-Chain Attack. https://www.sans.org/blog/what-you-need-to-know-about-the-solarwinds-supply-chain-attack/, 2020.

[132] TMM working group. Threat Modeling Manifesto. https://www.threatmodelingmanifesto.org, 2020.

[133] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave (Jing) Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy. 2020.

[134] Kim Wuyts, Riccardo Scandariato, and Wouter Joosen. LIND (D) UN privacy threat tree catalog. 2014.

[135] Kim Wuyts, Laurens Sion, and Wouter Joosen. Linddun GO: A lightweight approach to privacy threat modeling. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 302–309. IEEE, 2020.

[136] yamllint developers. YAMLlint: the YAML validator. https://www.yamllint.com/.

[137] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks. pages 37–54, 2020.