AttackDefense Framework (ADF): Enhancing IoT Devices and Lifecycles Threat Modeling

Tommaso Sacchetti, Marton Bognar, Jesse de Meulemeester, Benedikt Gierlichs, Frank Piessens, Volodymyr Bezsmertnyi, Maria Chiara Molteni, Stefano Cristalli, Arianna Gringiani, Olivier Thomas, et al.

AttackDefense Framework (ADF): Enhancing IoT Devices and Lifecycles Threat Modeling

TOMMASO SACCHETTI, Digital Security, EURECOM, Sophia Antipolis, France
MARTON BOGNAR, KU Leuven, Leuven, Belgium
JESSE DE MEULEMEESTER, KU Leuven, Leuven, Belgium
BENEDIKT GIERLICHS, KU Leuven, Leuven, Belgium
FRANK PIESSENS, KU Leuven, Leuven, Belgium
VOLODYMYR BEZSMERTNYI, NXP, Hamburg, Germany
MARIA CHIARA MOLTENI, Security Pattern, Mazzano, Italy
STEFANO CRISTALLI, Security Pattern, Mazzano, Italy
ARIANNA GRINGIANI, Security Pattern, Mazzano, Italy
OLIVIER THOMAS, Texplained, Valbonne, France
DANIELE ANTONIOLI, Digital Security, EURECOM, Sophia Antipolis, France

Threat modeling (TM) is essential to manage, prevent, and fix security and privacy issues in our society. TM requires a data model to represent threats and tools to exploit such data. Current TM data models and tools have significant limitations preventing their usage in real-world scenarios. For example, it is challenging to TM embedded devices with current data models and tools as they cannot model their hardware, firmware, and low-level software. Moreover, it is impossible to TM a device lifecycle or security-privacy tradeoffs as these data models and tools were developed for other use cases (e.g., software security or user privacy).

We fill this relevant gap by presenting the AttackDefense Framework (ADF), which provides a novel data model and related tools to augment TM. ADF's building block is the AD object that can be used to represent heterogeneous and complex threats. Moreover, ADF provides automations to process a collection of AD objects, including ways to create sets, maps, chains, trees, and wordclouds of AD objects. We present `ADF`, a toolkit implementing ADF composed of four modules (Catalog, Parse, Check, and Analyze).

We confirm that the data model and tools provided by ADF are useful by running an extensive set of experiments while threat modeling a crypto wallet and its lifecycle. Our experiments involved seven expert groups from academia and industry, each using the ADF on an orthogonal threat class. The evaluation generated 175 high-quality ADs covering ISA/IEC 62433-4-1 SecDev Lifecycle, side-channels, fault injection, microarchitectural attacks, speculative execution, pre-silicon testing, invasive physical chip modifications, Bluetooth protocol and implementation threats, and FIDO2 authentication.

Authors' Contact Information: Tommaso Sacchetti, Digital Security, EURECOM, Sophia Antipolis, France; e-mail: tommaso.sacchetti@gmail.com; Marton Bognar, KU Leuven, Leuven, Flanders, Belgium; e-mail: marton.bognar@kuleuven.be; Jesse De Meulemeester, KU Leuven, Leuven, Flanders, Belgium; e-mail: jesse.demeulemeester@student.kuleuven.be; Benedikt Gierlichs, KU Leuven, Leuven, Flanders, Belgium; e-mail: benedikt.gierlichs@esat.kuleuven.be; Frank Piessens, KU Leuven, Leuven, Flanders, Belgium; e-mail: frank.piessens@kuleuven.be; Volodymyr Bezsmertnyi, NXP, Hamburg, Germany; e-mail: volodymyr.bezsmertnyi@nxp.com; Maria Chiara Molteni, Security Pattern, Mazzano, Italy; e-mail: m.molteni@securitypattern.com; Stefano Cristalli, Security Pattern, Mazzano, Italy; e-mail: s.cristalli@securitypattern.com; Arianna Gringiani, Security Pattern, Mazzano, Italy; e-mail: a.gringiani@securitypattern.com; Olivier Thomas, Texplained, Valbonne, France; e-mail: olivier@texplained.com; Daniele Antonioli, Digital Security, EURECOM, Sophia Antipolis, France; e-mail: daniele.antonioli@eurecom.fr.

ACM 1558-3465/2024/10-ART
https://doi.org/10.1145/3698396

CCS Concepts: • **Security and privacy → Embedded systems security**; *Hardware attacks and countermeasures*; *Tamper-proof and tamper-resistant designs.*

Additional Key Words and Phrases: Threat Modeling, Embedded Systems Security, Hardware Security

1 Introduction

Threat Modeling (TM) is essential to manage our society's security and privacy risks. In short, TM allows systematically listing, prioritizing, and addressing digital threats [111, 121, 127], thus providing tangible security benefits compared to other more common but potentially less effective practices, such as compliance with security standards [52, 87, 124]. TM includes data formats to represent threats, tools to process them, and methodologies to identify, prioritize, and address them.

Current TM data formats and tools cannot cover critical aspects of real-world (embedded) devices. For instance, there is no way to accurately represent and process threats related to *hardware* (e.g., invasive physical attacks), *firmware* (e.g., bootloader), *hardware-software interface* (e.g., speculative execution and microarchitectural issues), and *communication protocols* (e.g., protocol- or implementation-level threats). Moreover, they focus on security or privacy attacks on products, neglecting possible *security-privacy tradeoffs*. Other shortfalls are that they cannot model the *products' lifecycle* (e.g., supply chain attacks) and the *defenses* associated with the attacks. Finally, current TM data formats and tools try to be human and machine-friendly, but they usually lack the latter, not allowing for automation and optimization of TM exercises. Hence, the TM community tends to focus on specialized classes of threats (e.g., software security or user privacy) and has limited tooling available.

We fill this gap by presenting the **AttackDefense Framework (ADF)**, a novel framework to enhance TM coverage, effectiveness, automation, and (re)usability. ADF's building block is the *AttackDefense Object (AD)*, a new data structure for representing threats. The AD object satisfies seven requirements that we set based on the state of the art. In particular, it covers attacks and defenses, security and privacy, hardware and firmware, product and lifecycle, and fine- and coarse-grained threats. Moreover, it is developed to be reusable and updatable (i.e., future-proof), friendly to machines and humans, and compatible with any TM methodology (e.g., STRIDE, LINDDUN, and ATree). An AD object can be written in any serialization language. We recommend using YAML or JSON. Moreover, we implement valuable automations on the AD objects, unlocking novel TM capabilities. For example, we show how to create *flat* AD sets or maps or *hierarchical* AD chains, trees, or wordclouds. These capabilities significantly increase TM's effectiveness, coverage, and speed.

We implement the ADF design in the `ADF` toolkit that contains *four* modules: Catalog, Check, Analyze, and Parse. A high-level overview of the ADF is shown in Figure 1. Catalog contains the ADs that we develop in our case studies. Parse can extract ADs from YAML, JSON, TOML, and XML files and can be easily extended to parse other file types. Check automatically validates the syntax, semantics, and content of the ADs using a combination of checkers such as yamllint and Python schema. Analyze provides 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.

We describe the results of *seven case studies* run by industrial and academic expert groups covering a broad spectrum of threats using a *crypto wallet* and its *life cycle* as a reference. In particular, we evaluate attacks and defenses related to ISA/IEC 62433-4-1 SecDev Lifecycle, side-channels, fault injection, microarchitectural attacks, speculative execution, pre-silicon testing, invasive physical chip modifications, Bluetooth protocol and implementation, and FIDO2 authentication. As a result of our evaluation, the seven experts created and used 175 ADs and provided invaluable feedback to improve the ADF. Our toolkit is open-source and available at https://github.com/francozappa/adf.

We summarize our contributions as follows:

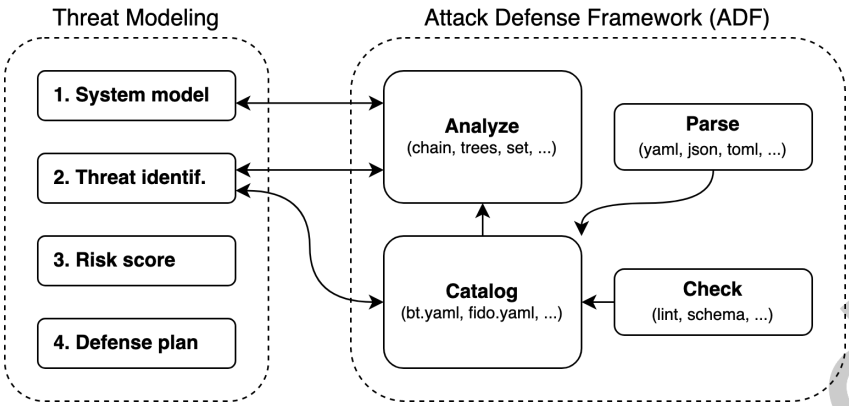

Fig. 1. ADF high-Level Overview

- We present the ADF, the first TM framework to model real-world product and lifecycle threats using a new data format called AD object and automations built around it. ADF satisfies seven requirements to achieve maximum coverage, usability, and compatibility with the state of the art.
- We describe `ADF`, a toolkit implementing the ADF in four modules: Catalog, Parse, Check, and Analyze. For each module, we discuss the relevant technical implementation details.
- We evaluate ADF in a real-world case study on a crypto wallet and its lifecycle. We involved seven expert groups that developed AD objects for different relevant threat classes. Our evaluation generated 175 high-quality AD objects covering lifecycle and product threats. We evaluated overlooked but critical threats such as side-channels, fault injection, microarchitectural attacks, speculative execution, and invasive physical attacks.

2 Background

This section provides background information on threat modeling and its methodologies, catalogs, and tools.

2.1 Threat Modeling

Threat modeling (TM) is an approach to identify threats to a target, prioritize them, and develop appropriate countermeasures. It consists of four sequential phases: (1) *system and attacker modeling*, (2) *threat identification*, (3) *threat ranking*, (4) *building a defense strategy*. From a top-down perspective, the four phases map to *four questions (Q1, Q2, Q3, and Q4)* [138]:

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 communication protocols with *sequence diagrams (SD)*. Usually, a DFD represents the system's components with solid lines, the trust boundaries with dotted lines, and the data flows with numbered arrows across components and boundaries. An SD represents the parties involved in a protocol and the messages they exchange.

Q2: What can go wrong? The second phase is about performing *threat identification (TI)* on our target. We define our attack surface, which is the set of components that we want to protect. Then, we consider different attacker models targeting our surface to achieve some goals using one or more techniques. TI is a laborious and mostly manual process. TI methodologies differentiate according to the threat domains and their relation.

Common TI methodologies include *STRIDE* [73] for software security, *LINDDUN* [23] for system privacy, and *Attack Trees (ATree)* [114] for attacker goals. More TI techniques are discussed in [118, 119].

Q3: What are we going to do about it? The third phase involves *risk or severity scoring*. In this phase, we score the threats identified in the previous phase using metrics usually based on the attack's impact, cost, and scalability. The most common threat scoring scheme is the *Common Vulnerability Scoring System (CVSS)* [93] that is employed by the US National Vulnerability Database (NVD) to score the severity of all currently known vulnerabilities. A CVSS score has three metrics: Base, Temporal, and Environmental. The Base metric provides a score from zero to ten and can be adjusted by scoring the Temporal and Environmental metrics. The Temporal metric takes into account the severity of a threat over time. The Environmental metric enables re-weighting the severity according to the target. The most widely used CVSS scores are v2 and v3, while v4.0 was released recently [96]. The scores can be computed using the NVD calculator web pages [94, 95].

Q4: Did we do a good enough job? The fourth phase of TM starts with the creation of a *defense plan*, which typically is a document containing a summary of the first three phases and a section explaining which threats are fixed, mitigated, and accepted as risks. Such a document also provides a list of countermeasures or fixes based on the identified threats. Following the development of the defense plan, the phase proceeds with its delivery, management, and refinement, involving active system monitoring for incidents. Ideally, the defense plan should be regularly reviewed and updated based on feedback from various channels, such as security logs, bug bounties, security advisories, and user input.

2.2 TM Methodologies

STRIDE. STRIDE was developed by Kohnfelder et al. in 1999 and adopted by Microsoft in 2002 [121] as part of its Secure Development Life Cycle (SDLC) [72] and Threat Modeling Tool (TMT) [74]. STRIDE focuses on identifying threats violating software security, emphasizing networked systems (e.g., web and cloud applications). 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 takes a DFD (or other system models) and for each element in the attack surface lists 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 [120].

LINDDUN. LINDDUN is a privacy-focused TM methodology developed by Deng et al in 2010. LINDDUN complements STRIDE as it uses the same reference system model (i.e., a DFD). But it 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 similar threats. The LINDDUN developers also provide a reference LINDDUN threat catalog extracted from empirical experiments [140] and LINDDUN GO, a lightweight LINDDUN version for newcomers [141].

ATree. ATree is a TM methodology proposed by Schneier in 1999. Each attack tree has the attacker goal 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). An ATree differs from STRIDE and LINDDUN. It focuses on goals other than threat classes, produces a hierarchical representation of threats rather than a list, and is scalable as a tree addresses multiple threats. On the flip side, an Atree is difficult to create and maintain because of its hierarchical structure. For example, a tree might become useless by missing just one sub-goal (sub-tree).

2.3 Threat Catalogs

Threat modeling methodologies and tools rely on threat catalogs (i.e., collections of known threats). For example, STRIDE and LINDDUN have bundled catalogs of high-level security and privacy threats [140]. Additionally, the three relevant catalogs – all maintained by MITRE – are the Common Attack Pattern Enumeration and Classification (CAPEC) [81], Common Weakness Enumeration (CWE) [83], and Common Vulnerabilities Exposures (CVE) [82].

CAPEC. CAPEC is a catalog of attack patterns extracted from real-world threats. An attack pattern describes the adversary's approach to exploit known weaknesses. Each CAPEC entry has the following fields: unique ID, description, likelihood, severity, relationship, execution flow, prerequisites, skills required, consequences, mitigations, related weaknesses (CWE), taxonomy mappings, and content history.

CWE. CWE is a collection of known software and hardware weakness types. A CWE entry has the following attributes: UID, description, relationship, modes of introduction, consequences, demonstrative examples, observed examples (CVE), membership, notes, taxonomy mappings, related attack patterns (CAPEC), references, and content history. CWE entries are organized in a tree hierarchy of multiple levels of abstraction.

CVE. CVE is a standard format to store, discover, analyze, and correlate vulnerabilities. Each CVE has a unique ID in the form of `CVE-YYYY-NNNNNN`, description, references, assigner, creation record, and other fields.

2.4 Tools

Among the various open-source threat modeling tools, we focus on those that provide a custom and extensible threat catalog [67] and a data representation structure to represent the threats, i.e., `pytm` [126] and `threagile` [113].

pytm. Pytm is a Python-based framework developed by the Open Worldwide Application Security Project (OWASP). It introduces the paradigm of threat modeling as code, simplifying TM integration into the development process. The framework has some pre-defined and extendible classes for constructing a system model. In particular, the user writes a script instantiating Python objects for each system component and specifies how the components are connected. Then, the tool generates a visual system diagram (e.g., DFD or SQ) and automatically identifies potential threats using a threat library built upon a custom data representation model that uses JSON. The threat library contains approximately 100 entries at the time of writing.

Threagile. Threagile is a tool following threat modeling as a code paradigm written in Golang. The tool provides a catalog with 41 threats in the form of hardcoded rules. Each rule is also associated with its STRIDE category, allowing it to be easily integrated with TM exercises requiring adherence to the STRIDE taxonomy classification. The user is required to build a system model using YAML. The tool tries to apply the rules to the system model and then automatically generates DFDs and other outputs (i.e., Excel and JSON).

3 ADF Design

This section motivates the need for ADF and presents its seven requirements, the AttackDefense (AD) object and related automations, and ADF's compatibility with the STRIDE and LINDDUN TM methodologies.

3.1 Motivation

We started working on ADF because we could not find *threat data models* to TM real-world (embedded) devices and lifecycles. CWE, CVE, and CAPEC (introduced in Section 2) represent the state-of-the-art data formats to model devices' weaknesses, vulnerabilities, and attack patterns. However, they are problematic for several reasons. Studies demonstrated that they are challenging to use in scenarios requiring automation [10, 70]. Other

work showed their limited precision for vulnerability management [102] and even wrong entries in the CVE database [39, 59]. There is no threat data model to represent risks associated with a device lifecycle, such as supply chain attacks.

Let's take as an example CAPEC-668 [80], which models the Key Negotiation Of Bluetooth (KNOB) attack technique presented in [5, 7]. And use CAPEC-668 and the associated CWEs and CVEs to TM a crypto wallet supporting Bluetooth. What can we achieve? Very little, since we get generic information from the entries, such as high-level description of the KNOB attack, related patches, and links to external resources. But, we miss essential threat modeling data, like KNOB attack surface, vector, technique, and defensive fixes and mitigations.

A second issue that motivates the need for ADF is the lack of *tools* capable of TM real-world (embedded) devices and lifecycles. Pytm and Threagile are industrial and open-source TM tools offering machine-readable threat representations, catalogs, and automation. But, they target specific TM domains, such as securing web applications, and are not suited to TM devices with hardware, firmware, software, and communication protocol issues. For instance, we cannot TM a crypto wallet with a microcontroller, a secure element, a secure bus connecting them, a USB stack, and a Bluetooth wireless interface. Moreover, we know *no tool* that can TM a device lifecycle, e.g., pre-deployment issues with its design and implementation. Returning to the crypto wallet example, we would like to TM hardware and software supply chain risks (i.e., device lifecycle threats), but there is no capable tool.

3.2 Requirements

Based on the two issues presented in Section 3.1, we set *seven requirements* for ADF to enable TM real-world (embedded) devices and their lifecycles. We now describe each requirement.

R1: Attacks and Defenses. Current TM frameworks focus on the attacker. We want to focus on the attacker and the defender at the same time. This approach enables reasoning about fine-grained and coarse-grained mitigation strategies, identifying critical attacks with and without 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 an attack.

R2: Security and Privacy. Existing TM frameworks treat security and privacy separately, leading to issues like overlooked security-privacy trade-offs. We want to consider security and privacy simultaneously to capture their unavoidable joint benefits and trade-offs. For instance, we could model a scenario to explore the competing goals of confidentiality and integrity (security) vs. repudiability and traceability (privacy).

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 during TM. Besides traditional TM areas, we want to cover novel hardware and firmware threat classes, including invasive and non-invasive physical attacks like side-channels, fault injection, and physical chip manipulations. Furthermore, we want to address threats at the intersection of hardware and software, such as microarchitectural and speculative execution attacks.

R4: Product and Lifecycle. Existing TM frameworks focus on analyzing devices or systems, not their development lifecycle. For instance, current frameworks cannot model hardware and software supply chain attacks. We want to include lifecycle threats in our framework to enable TM practitioners to manage critical process risks such as SolarWinds [137] and Supermicro [115].

R5: Fine- and Coarse-grained Threats. While current TM frameworks focus on generic classes of threats, we want to support threats at different levels of abstraction. For instance, we model coarse-grained threats (e.g., generic attack techniques), such as buffer overflows, and fine-grained ones (e.g., real-world attack instances), like Heartbleed on OpenSSL. By doing so, we enhance our TM analysis capabilities. For instance, we can automatically generate hierarchies of threats based on abstraction levels, including trees, chains, and graphs.

Listing 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", ...]
  ...: ...
```

R6: Reusable and Updatable. Present TM frameworks are difficult to combine, update, and automate. We want to prevent duplication of the same TM exercises by providing reusable data formats and tools. We want to interoperate with existing TM methodologies such as STRIDE and LINDDUN. 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 to cover situations where new threats are identified or old threats are patched or reintroduced.

R7: Machine- and Human-friendly. Current TM frameworks are either machine- or human-friendly. We want a framework that minimizes friction between humans and machines. Users should be able to read, write, analyze, and share attack and defense strategies. The framework should accommodate users with varying expertise in TM, including developers and threat modeling experts. The framework should enable machines to automatically generate valuable TM 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, and machine-checkable data formats.

3.3 AttackDefense Object (AD)

Starting from the seven requirements outlined in Section 3.2, we developed the *AttackDefense Object (AD)*, our threat data model for ADF. We indicate an AD Object as *AD* and multiple ones as *ADs*. AD is a programming language agnostic data structure representing a threat (i.e., attack, defenses, and useful metadata). We use AD objects to model real-world threats on devices and lifecycles, such as the design, implementation, evaluation, and shipment of a crypto wallet.

Listing 1 shows how to write an AD using YAML, but note that other serialization languages, including JSON, TOML, or XML can be used. Each AD has a unique name (`ad_name`), six *primary* fields, and *optional* fields. A field is a key-value pair and supports various data types, including dictionaries, lists, strings, and integers.

The AD has six primary fields:

Listing 2. knob_ble AD

```
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"]
```

- `a` contains a string describing an attack with an arbitrary level of abstraction (e.g., coarse-grained or fine-grained).
- `d` stores 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]`) satisfying such policy. The sub-dicts could be ordered according to some criteria (e.g., from the most effective to the least effective).
- `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.
- `vect` is a list of strings containing the attack vectors (i.e., techniques) related to the attack.
- `model` stores the adversary models capable of performing the attack 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.

Additionally, there are some *optional* fields to enhance the AD:

- `risk` is a list of strings storing risk scores associated with the attack (e.g., CVSS).
- `year` is an int storing the year when the attack was first discovered.
- `cve`, `cwe`, and `capec` are lists of strings storing identifiers from those catalogs related to the attack.
- `vref` is a list of vendor reference strings associated with the attack, including security advisory identifiers from Linux [38] or Android [48].

Recall that in Section 3.1, we showed that the CAPEC entry for KNOB misses many details about the attack. In Listing 2, we show how we modeled the KNOB attack using the `knob_ble` AD. KNOB involves an adversary with a man-in-the-middle (MitM) position in the BLE proximity range (`model`) targeting the entropy negotiation phase of the BLE pairing protocol (narrowing down `surf`). The attack involves downgrading the pairing key entropy and brute-forcing the key (`vect`). KNOB is a protocol-level attack involving BLE's Security Manager Protocol (SMP) (`tag`). The attack was discovered in 2019 (`year`). It is associated with CVE-2019-9506 (`cve`), CWE-310 and CWE-327 (`cwe`), and CAPEC-668 (`capec`). It has high CVSSv3 risk and medium CVSSv2 risk (`risk`). We can defend against KNOB either by mutually authenticating the entropy negotiation protocol or by forbidding low entropy values for the BLE pairing key (`d`).

The AD threat data model *satisfies* the seven requirements we set in Section 3.2. It stores information about an attack and its associated defenses (R1). It can cover relevant threat domains, including security, privacy, hardware, firmware, product, and lifecycle threats (R2, R3, R4). Furthermore, the AD allows for different abstraction levels

Fig. 2. iOS Pegasus RCE chain from 2021. Each chain element is an AD.

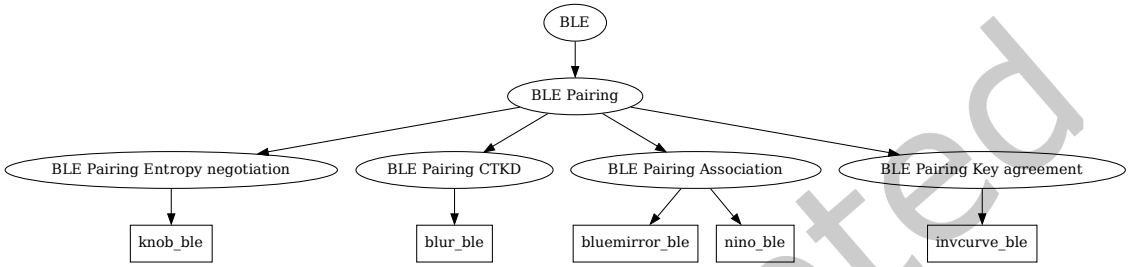

Fig. 3. ADs attack surface tree for BLE Pairing. The tree leafs are the ADs.

(R5), and its simple structure is straightforward to update and reuse (R6). ADs are human and machine-friendly as they are representable with programmable and human-readable serialization languages, such as YAML [143] or other serialization languages (R7).

3.4 Flat and Hierarchical ADs

We design ways to manipulate ADs in flat and hierarchical ways to enhance TM (e.g., better threat selection, identification, visualization, and report). Specifically, we create functions to produce *flat* and *hierarchical* combinations of ADs from a collection.

Flat (set, maps). By filtering ADs based on relevant AD field values, we can create AD *sets* (i.e., unordered lists). A set allows selecting ADs based on any combination of fields, such as attack surface or technique. Additionally, we can create AD *maps* to link them to known threat taxonomies such as CIA (Confidentiality, Integrity, Availability), STRIDE, and LINDDUN. Hence, ADs are compatible with existing TM methodologies. For instance, information from STRIDE or LINDDUN TM exercises can be easily encoded and exploited using ADF. Our mappings support 11 taxonomies and can be extended. See Table 2 in Appendix A for more details.

Hierarchical (chains, trees, wordclouds). We can represent complex threats using hierarchical collections of ADs. This representation is relevant when modeling exploits involving a chain of attacks exploiting multiple vulnerabilities. For example, we can model the Pegasus Remote Code Execution (RCE) exploit on iOS from 2021 [55] 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), and the fourth represents the remote privileged read exploit for iMessage CVE-2019-8646 (see ellipse AD in Figure 2).

We can represent fine-grained and coarse-grained ADs using a *tree* structure. This visualization is useful because it allows placing the attack surface (`surf` field) as the tree's root, building the ramifications based on sub-surfaces. For instance, Figure 3 depicts a tree of ADs related to protocol-level threats in BLE pairing. Each

Fig. 4. Wordcloud of the attack surfaces covered by our `bt.yaml` catalog

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.

We can show the coverage of a specific AD using *wordclouds* created from ADs. A wordcloud represents a text string by displaying words in varying sizes, proportional to their frequency within the string. For instance, we can generate a wordcloud (Figure 4) to examine the attack surface of a Bluetooth AD 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) have a larger size than sub-surface words (e.g., Pairing, Session), and sub-surface words than sub-sub-surface words (e.g., Entropy negotiation, CTKD), aligning with our expectations.

4 ADF Implementation

We describe the implementation details of the `ADF` toolkit introduced in Section 3. The toolkit is organized in *four* modules: Catalog, Parse, Check, and Analyze (shown in Figure 1). We now describe each toolkit module in detail. The files and folders mentioned below are relative to the repository root folder.

4.1 Catalog

The catalog module contains the developed ADs and is located within a dedicated subfolder in our GitHub repository. To write these ADs, we used 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. Moreover, YAML is developer-friendly, supported by all popular programming languages, and 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 show selected ADs from the catalogs.

One of the unique features of ADF is its capability to establish security and privacy requirements for a device lifecycle in addition to the ones for the device itself. The `sw_orion` AD shown in Listing 3 models 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, and it can be addressed by properly authenticating software updates with valid certificates.

Another useful feature of ADF is the possibility to integrate new threats into a catalog effortlessly. Let us assume that in 2030, we are notified about a newly disclosed and critical vulnerability affecting the Linux kernel and 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

Listing 3. sw_orion AD

```
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"]
```

Listing 4. linux_new_bof AD

```
  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"]
```


stack. It was scored as critical with CVSSv3 and assigned CVE-2030-0007. How can we add this threat to our AD catalog?

Listing 4 shows how we model the new threats with an AD called `linux_new_bof`. We describe a short and self-contained attack in `a`. In `d`, we list two defenses: (i) we recommend employing a memory-safe programming language (a policy) and we select Rust [31] (a concrete mechanism) because Linux supports it; (ii) we suggest sanitizing kernel-space memory (policy) with the Kernel Address Sanitizer (KASAN) [27], a dynamic memory testing tool for the Linux 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.

A catalog can also include *coarse-grained* ADs, which can represent threats' classes (e.g., an attack technique) in a single object. In Listing 5, 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 using Rust and KASAN, we add other defenses such as the Kernel Memory Sanitizer (KMSAN) [28] to find uninitialized values, musl [88, 89] to reduce libc's attack surface and use a hardened memory allocator, Kernel Address Space Layout Randomization (KASLR) which requires a Position Independent Executable (PIE)

Listing 5. `linux_bof` AD

```
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]
```

kernel built, and CPU NX bit to avoid executing code on the stack. Since it is a coarse-grained and thus a generic AD, the defenses are not ordered by effectiveness. For the same reason, optional fields, such as `risk`, `year`, and `cve` are not considered as an attack class applies to multiple threats and not a specific one. The AD attack surface is Linux. The vectors are stack or heap-based BoFs. The attacker model is proximity or remote, and the `tag` is Impl.

4.2 Parse

The parse module, implemented in the `parse.py` file, contains the functions to parse ADs from YAML, TOML, JSON, and XML files into Python dictionaries. For instance, the YAML AD presented in Listing 1 converts into the AD dictionary in Listing 6. 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 1), parses the file using PyYAML [30] with `CSafeLoader`, which is a secure and efficient parser from LibYAML [26]. The output is a dictionary with nested sub-dictionaries, as demonstrated in Listing 6. Similarly, we implemented other specialized parsing functions to handle TOML, JSON, and XML files.

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. The testing files are in the `template` folder. The tests can be executed with the command `make test-parse`.

4.3 Check

The check module, implemented in `check.py`, 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 parsed Python dictionary. The module is designed with a top-level function (`check(path; Path, words=None) -> dict`) that calls the relevant syntax and semantics checkers based on the `path` file extension. The parse module presented in Section 4.2 performs the parsing.

The validation is performed in two main steps: (1) syntax checking using yamllint [144] with its default configuration to avoid duplicate `ad_names` and wrong indentations, and (2) semantic checking using a custom function that enforces a particular schema with specific types and allowed values on the parsed dictionary.

Listing 6. Python dictionary parsed from the YAML AD in Listing 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"],
    ...
}
```

Listing 7. 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),
  ...
}
```


Listing 7 shows an excerpt of our AD dict schema implemented using the schema [62] 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 ADs consistent, especially when different teams work on the same ADs. Moreover, it helps to track 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 can automatically test the module with `check_test.py`. This script runs the `check` function on the YAML, TOML, JSON, and XML template AD files in the `template` folder. The command to run the tests is `make test-check`.

4.4 Analyze

The analyze module, implemented in `analyze.py`, provides various automation to process a (checked) dictionary of ADs. Internally, it uses pandas [29], matplotlib [37], and graphviz [24, 25] to produce its outputs. These are free and powerful open-source libraries. Currently, Analyze can generate sets, maps, trees, wordclouds, and chains of ADs 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 AD DataFrame, a table-like data structure. Each row of the DataFrame 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 can be used 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 analyses.

The `get_map` function returns sets of AD based on known security and privacy taxonomies. Table 2 lists the 11 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, CWETS22, and CWETS23). For example, given an AD 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.

The module can also generate trees of ADs based on the `surf` and `tag` fields. For example, Figure 3 shows a tree of protocol level (`tag = Protocol`) ADs related to BLE (`surf = [BLE, ...]`). As Section 3.3 explains, `surf` is an ordered list of strings narrowing down the attack surface. The `get_surf_tree` uses this ordering to build a tree automatically. In particular, it roots the tree to surf and creates branches for sub-surf and sub-sub-surf. Then, each AD appears as a leaf according to its `surf` list.

The `get_wordcloud` function outputs AD wordclouds based on a field using Python's wordcloud [36] and matplotlib [37] packages. For example, Figure 4 shows an attack surface wordcloud computed from the `bt.yaml` file discussed in Section 4.1. Internally, the function takes an AD dataframe and a column key, collects all the column values in a list, and then uses a `Counter` data structure from the `collections` library to count the occurrences of each value. This approach is better than counting each string as a word, as some words in the cloud contain multiple strings (e.g., "Feature exchange" counts as a single word in the cloud).

The `get_chains` function generates a chain of ADs, given an AD dataframe and a target AD using graphviz. For instance, Figure 2 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.

5 ADF Evaluation

We evaluated the ADF with a case study where we involved *seven* groups of IoT security and privacy experts, and we asked them to develop dedicated AD object catalogs and use them to TM a *crypto wallet* and its *lifecycle*. Our evaluation shows that the ADF covers the seven requirements presented in Section 3.2 in a real-world use case. Next, we present our evaluation setup, results summary, and details.

5.1 Setup

Figure 5 shows a simplified block diagram of our target crypto wallet. At its core is a Secure Element (SE) linked to a general-purpose microcontroller (MCU) through a secure bus. The SE runs a real-time operating system (RTOS)

and performs all the security and privacy-sensitive operations, such as generating and storing cryptographic keys and signing and verifying transactions. The MCU, which is separate from the SE, employs secure boot via a boot ROM and operates on a Linux-based OS to provides all the other functionalities. The crypto wallet supports Bluetooth Low Energy (BLE) for wireless connectivity, USB for wired connections, and Fast IDentity Online (FIDO) for two-factor and single-factor (i.e., passwordless) authentication [2].

The crypto wallet development process relies on a new seven-phase lifecycle shown in Figure 6 that we call *trusted lifecycle (TLC)*. In the Threat Modeling and Risk Assessment (TM & RA) phase, we outline the security and privacy requirements for both the product and process. In the Design phase, we design the hardware, software, and protocol aspects of our crypto wallet based on functional requirements and the ones set in the TM and RA phase. The wallet is then concretely realized in the Implementation phase. Successively comes the Evaluation phase, where we perform extensive hardware and software testing (e.g., fuzzing, side-channel analysis, and fault injection experiments). The first four phases are pre-deployment, and each can provide feedback to the others. For example, fuzzing experiments can find implementation bugs that we can fix by updating our implementation. The crypto wallet is deployed during Installation, managed during Maintenance, and disposed of during Retirement. Maintenance includes secure firmware updates and provides post-deployment feedback that informs the design and implementation of the next-generation crypto wallet.

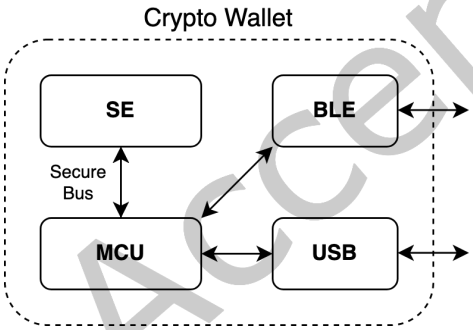

Fig. 5. Crypto wallet block diagram (simplified)

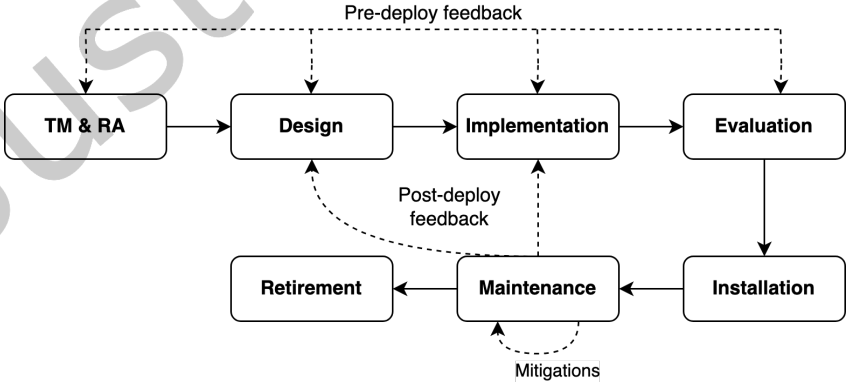

Fig. 6. Trusted Life Cycle (TLC) used to develop the crypto wallet. TLC has seven phases connected by solid lines. Dotted lines represent mitigations pre- and post-deployment feedback.

5.2 Results Summary

Table 1 presents a summary of the results. The columns show, in order, the TM domain, a pointer to the relevant section, the covered aspects (e.g., hardware, software, process, and product), the number of produced ADs (for a specific domain), and their related YAML files in the toolkit directory. We note that ADs from different domains are independent, while ADs belonging to the same domain may intersect at different granularity levels (e.g., a specific attack technique may be part of a broader attack class). Overall, we evaluated *traditional* TM domains, such as software and protocol security, but also *original* ones, including the TLC, pre-silicon testing, invasive physical attacks, and microarchitectural threats.

We conducted field testing of the ADF from various perspectives, involving users with diverse backgrounds. This heterogeneous testing allowed us to assess its effectiveness and usability across different scenarios. Our threat modeling activities covered many threats, including specific vulnerabilities affecting individual software components and generic attacks and attack techniques applicable to multiple components.

During the evaluation of the ADF, we identified some 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 serve as blueprints and are publicly available.

5.3 ISA/IEC 62443-4-1 SecDev Lifecycle

The ISA/IEC 62443 standard is divided into four tiers, each having multiple work documents. ISA stands for International Society for Automation, and IEC for International Electrotechnical Commission. 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

Table 1. Evaluation results from our case studies where seven expert groups TM a crypto wallet (Figure 5) and its TLC (Figure 6). We covered hardware (HW), software (SW), firmware (FW), protocols (PT), life cycles (LC), product (PO), security (SE), and privacy (PR) threats. We developed a total of *175* ADs. The AD YAML files are available at https://anonymous.4open.science/r/adf-anon-CC3F/ in the `yaml` folder.

TM domain	Sec	Coverage	ADs	Files
ISA/IEC 62443-4-1 SecDev Lifecycle	5.3	LC, SE	40	`62443-4-1/*.yaml`
Physical Side-Channel and Fault inj.	5.4	PO, HW, SE, FW	20	`sc-fi.yaml`
Microarch. and Speculative Execution	5.5	PO, HW, SW, SE	14	`microa.yaml`
Presilicon RISC-V SE Testing	5.6	PO, HW, SW, FW, SE	8	`presil.yaml`
Invasive Physical IC Attacks	5.7	PO, HW, FW, SE, PR	26	`physical.yaml`
Bluetooth Protocol and Impl. Attacks	5.8	PO, SW, FW, PT, SE, PR	46	`bt.yaml`
FIDO2 Authentication Attacks	5.9	PO, HW, SW, FW, PT, SE	21	`fido*.yaml`

In total, we developed 40 ADs that map to the requirements described in the eight practices, and our AD files are in the `62443-4-1` folder. We named each AD according to the requirement it maps to (e.g., sm-1-development process). We directed our primary effort toward rewriting the requirements as combinations of threats and 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 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 requirements are 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, while more justified, they still lack the flexibility to evaluate only a subset of the requirements, depending on the context.

With the logical structure of ADs, it is possible, when modeling process requirements, to reveal hidden hierarchical structures and uncover links and references among individual requirements. In the context of ISA/IEC 62443-4-1, while the Practice already enables requirements grouping, it is likely 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. Utilizing the AD data structure enables us to highlight these connections and relationships among requirements, 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

The SM-1 is the first requirement of ISA/IEC 62443-4-1. It lays the foundation to establish 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 covered by the specific requirements of Practice 5, which is dedicated to testing. The AD model allows us to highlight this hierarchical connection. Listing 8 shows a possible AD object for the SM-1.

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 $@svv-*$ translates to “all the ADs starting with the prefix *svv−*”, which are all ADs mapping to the requirements of Practice 5.

5.3.1 Practical use case: advanced search on process requirements. As a practical evaluation of the AD framework on the process requirements of ISA/IEC 62443-4-1, we tried to search the knowledge base of ADs based on real-world needs. The multiple ways of categorizing ADs allow us to perform this kind of operation more precisely than with the grouping by *Practice*. For example, if we are searching all the requirements to produce a “threat model”, relying on the `tags` field allows us to immediately identify the following 17 requirements throughout ISA/IEC 62443-4-1: *sr-1, sr-2, sr-3, sr-4-*, sr-5 dm-1, dm-2, dm-3, dm-4, dm-5, dm-6 sg-1, sg-2, sg-3, sg-4, sg-5, sg-6*. We can observe that although the grouping by *Practice* maintains a certain degree of cohesion among requirements, some scattering may still be present when searching for high-level concepts. This extraction proves very helpful in situations where adopters of a standard such as ISA/IEC 62443 are interested in gathering all requests from the standard about specific processes and procedures. Other similar searches in which the AD model has proven effective have been:

Listing 8. 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]
```

- identifying all requirements about the management of cryptographic secrets within two different standards
- search for all process requirements mitigating the presence of a malicious insider

5.4 Physical Side-Channel and Fault-Injection Attacks

A crypto wallet secure element could be susceptible to physical side-channel [65] and fault-injection [13] attacks. Side channel attacks use side information, such as timing or power measurements, to break security mechanisms, while fault injection attacks inject software or hardware faults to achieve the same goal.

A crucial aspect to consider when analyzing these attacks is the threats' *abstraction level*. For example, a possible implementation of elliptic curve cryptography (ECC) scalar multiplication is the double-and-add method. In this implementation, the sequence of operations depends on the key bits, making it vulnerable to *Simple Power Analysis (SPA)* [65] attacks.

We can look at SPA from different levels of abstraction: SPA attacks on non-constant time/flow implementations, SPA attacks on ECC scalar multiplication, and SPA attacks on ECC scalar multiplication with the double-and-add implementation. A broader description allows us to describe more generalized attacks, covering a broader attack surface but making it more difficult to define specific countermeasures. Conversely, a detailed description allows for more fine-grained defense descriptions but requires more ADs. For example, with a generic AD about SPA attacks on non-constant time/flow implementations, we could cover both ECC and RSA (e.g., square-and-multiply algorithm), but it would provide less relevant countermeasures.

A higher abstraction level is favorable for certain classes of threats. For instance, we can apply Differential Power Analysis (DPA) [65] to constant-time/flow implementations of various cryptographic primitives (e.g., ECC, AES, Kyber, ...). The most popular countermeasure to counter DPA is data randomization, which is agnostic to the specific cipher, allowing for a high abstraction level. For symmetric key primitives, data randomization is typically achieved through masking [18, 49]. Binding instead is used to randomize public key primitives [22].

Another specific property of physical side-channel and fault-injection attacks is that it is possible to perform analogous attacks through *different attack vectors*. For example, we can do side-channel attacks with both power [65] or EM measurements [46, 106]. Similarly, we can cause an instruction skip using various fault injection techniques (e.g., voltage fault injection [112], laser fault injection [123], EM fault injection [86], ...). While the outcome is often similar, different injection techniques may require unrelated countermeasures.

Moreover, in practice, some attack vectors may be excluded from the analysis when deemed out-of-scope due to their associated cost or required sophistication.

5.4.1 Threat modeling using ADF. Suppose our crypto wallet's SE uses an ECC-based signature scheme to sign transactions. We now TM a naive implementation of ECC scalar multiplication used for signing. As for the threat model, we consider a passive physical attacker who wants to extract the signing private key. We created 20 ADs, contained in the `sc-fi.yaml` file.

By filtering fields `"surf"` on "Cryptographic algorithm implementation→ ECC" and filtering "model" on "Physical→ Passive", we can find threats specific to ECC. We find two ADs specific to ECC, and both describe a SPA attack on the double-and-add scalar multiplication method using either the power consumption of the embedded device or its EM emanations. We observe that both the ADs mention implementing the scalar multiplication in a constant-time fashion by using the Montgomery ladder as a countermeasure. That would be the preferred choice since it allows solving two problems simultaneously.

As a second step, we consider ADs one level of abstraction higher by filtering `"surf"` on "Cryptographic algorithm implementation" and "model" on "Physical→ Passive". Here, we find 7 ADs. For instance, we find that the secret key of our now-constant-time ECC implementation can be extracted using DPA, using both power consumption and EM emanations (respectively, `sca-power-dpa`, and `sca-em-dpa`). The suggested countermeasure in both cases is to apply blinding [22]. Additionally, we find that an implementation with blinding may be vulnerable to higher-order DPA attacks. However, in this example, we consider higher-order DPA attacks out-of-scope.

The resulting implementation is a constant-time/flow implementation with blinding, resistant to SPA and first-order DPA attacks. This methodology can be applied to other cryptographic primitives implemented in software or hardware, allowing them to be threat-modeled in the context of physical attacks. If active attacks are in scope, we proceed by filtering fields `"surf"` on "Cryptographic algorithm implementation" and filtering "model" on "Physical→ Active".

5.5 Microarchitectural and Speculative Execution Attacks

Microarchitectural attacks exploit vulnerabilities in a processor microarchitecture (i.e., the implementation of an instruction set architecture (ISA)). Meanwhile, speculative execution threats abuse processors capable of speculatively executing instructions to gain performance, which can lead to information leakage on incorrect speculations.

While early microarchitectural attacks typically targeted desktop and server environments, small embedded devices are also vulnerable to microarchitectural attacks. These attacks target either the same microarchitectural components that are found on high-end devices, or implementation aspects that are specific to these devices [12, 131]. As a result, developers of embedded applications and devices, such as a crypto wallet, also need to consider these attacks when threat modeling and apply mitigations accordingly.

To demonstrate the process in our case study, we developed AD objects for an abstract Spectre attack [64] and specific sub-variants [14] to allow modeling at different abstraction levels. As Spectre attacks use a microarchitectural side-channel to transmit secret information, we also developed several AD objects that represent potential microarchitectural leaks [12, 50, 98, 103, 131, 142, 145]. We have a total of 14 ADs in the `microa.yaml` file. Listing 9 shows an AD representing the Spectre-BTB variant, in which an attacker can leak information from transient instructions executing due to the mistraining of the branch target buffer (BTB).

5.5.1 Threat modeling using ADF. Considering microarchitectural attacks in a threat modeling exercise is a challenging exercise. As the name implies, microarchitectural attacks heavily depend on the given CPU's internal design and optimization features. In an ideal scenario, the threat modeling of the device is performed by the

Listing 9. AD object for Spectre-BTB

```
spectre-btb:
  a: Transient execution resulting from mispredicted indirect branches can cause persistent
    changes in the microarchitecture, which can be used to intentionally leak secrets from a
    victim process using a covert channel.
  d:
    "preventing speculation altogether" : [ "Inserting fence instructions at every indirect
    jump", "Disabling speculation in the hardware" ]
    "preventing speculation on secrets" : [ "Implementing a secure speculation scheme in the
    hardware, such as ProSpeCT" ]
    "removing the covert channel": [ "Cache partitioning", "Disabling hyperthreading", "(more
    depending on the microarchitectural side channels)" ]
  surf : [ "Shared resource enabling a covert channel between the victim and the attacker", "
    Shared branch target buffer (BTB) between the victim and the attacker" ]
  vect : [ "Controlling a shared resource leading to the covert channel", "Poisoning the BTB"
    ]
  model : [ "code execution", "remote" ]
  tag : [ "transient attack" ]
  year : 2018
  cve : [ "CVE-2017-5753", "CVE-2017-5715" ]
```

manufacturer, or the relevant ADs are provided with the device's datasheet. Threat modeling an existing CPU as an outsider is a complicated exercise requiring sizeable reverse engineering and attack experimentation efforts [108], as the hardware features that enable the microarchitectural attacks are often undocumented.

The threat modeling process proceeds as follows. As microarchitectural attacks aim to extract secret information from a CPU, we narrow down which system elements process secret information directly. In our case, it is the SE only, meaning that we can exclude the MCU. Then, we can filter the relevant ADs based on the SE, its microarchitectural features, and the attacker model, searching for all relevant hardware optimization features. For example, if the SE features a branch target buffer (BTB), we would search for this in the `"surf"` field of ADs and find that this microarchitectural optimization can enable the Spectre-BTB variant. Spectre-BTB also requires a shared covert channel between the victim and the attacker process. However, if the device also features a shared cache across processes, and we search for this among the ADs, we will find that this can enable Prime+Probe cache attacks, which can function as the covert channel for Spectre.

5.6 Presilicon Testing

Pre-silicon verification mainly targets logic design errors or unauthorized modifications of an integrated circuit. During this phase, the attack vectors tested are limited compared to a complete System-on-Chip (SoC) with precisely specified hardware and software components. For instance, the pre-silicon stage testing omits physical defenses (e.g., shields and sensors) since they cannot be tested without a silicon die.

Starting from a CV32E40S RISC-V secure core [97], we generated 8 ADs shown in `presil.yaml`. Successively, we added more components and specified a SoC sample featuring 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 CV32E40S secure core. While it remains a generic SoC, it can be the starting point for designing other SoC. New defensive policies arose from introducing new critical components to the design, and thanks to the framework's flexibility, it is possible to specify different abstraction levels for them.

The SE stores sensitive data for various applications, including:

(1) Cryptocurrency private keys

(2) BLE communication keys
(3) Firmware verification keys
(4) User authentication data
(5) FIDO authentication data

The MCU communicates securely with the SE over a dedicated bus, enabling the execution of the following functions:

(1) Transaction signing
(2) SE/MCU firmware image verification and update
(3) BT key generation
(4) BT packet encryption/decryption
(5) User authentication
(6) FIDO challenge-response generation

The bootloader stage is crucial during pre-silicon testing, as it resides in ROM and is not easily updatable. The MCU and SE boot simultaneously, the MCU's bootloader initiates the SE to verify the Micro's firmware image. In this generic system design, we have identified attack scenarios applicable during pre-silicon testing, with measures aimed at prevention. While general countermeasures overlap with those derived from complete threat modeling, we highlighted specific attacks mitigated in the pre-silicon testing stage. For example, a firmware verification skip attack, described in the provided AD object, bypasses the firmware verification process by physically introducing faults. Multiple defensive means can be implemented and tested during pre-silicon testing, including software-implemented fault tolerance and fault injection emulation. Although the actual silicon is not available, it is possible to simulate the potential effects of a fault injection on the bootloader execution determine concrete suitable defenses. Then, before manufacturing the final product, software techniques for fault tolerance and control flow attestation can be implemented. However, given the common threat of fault attacks, installing physical defenses on the chip is necessary to ensure the SE bootloader's control flow integrity (fault protection).

Given the fixed hardware components of the SE and MCU, a significant challenge is ensuring the security of chosen cryptographic algorithms and the software components implementing them in the SE and MCU firmware. Although pre-silicon testing has limitations compared to a fully defined system design, we have successfully identified crucial application-agnostic attack vectors. An example of such an attack would be a sensitive data extraction through a side-channel leakage. Consequently, we listed several countermeasures that can be preemptively applied during different TLC stages, particularly in the Design, Implementation, and Evaluation stages. These include noise introduction, power balancing, or constant-time implementation, which can be considered and tested without a physical device.

The level of system abstraction was not problematic, as many pre-silicon testing techniques address a broad range of attacks. The concrete impact of more specific designs on threat modeling in the pre-silicon phase primarily lies in tailoring test cases to cover different software features. Most AD objects remain generic, applicable to any system design with high-security requirements, allowing for their reuse during the modeling of concrete products where more detailed information about hardware and software is available.

5.7 Invasive Physical IC Attacks

We employed ADF to TM invasive physical Integrated Circuit (IC) attacks on our crypto wallet. An invasive attack focuses on a specific IC target, like a ROM or RAM. It uses specialized instruments and techniques such as lasers, optics, and micro-probing to achieve a goal. Despite being impactful, these attacks are *not* typically covered by TM. We built 26 ADs, and we provide them in `physical.yaml`.

For example, we modeled a focused ion beam (FIB) attack, which was never done before, as in Listings 10. In a FIB attack, the adversary shoots an ion beam at the IC to achieve different goals, including skipping or modifying

Listing 10. attack_4 AD

```
attack_4:
  a: FIB modification
  d:
    Modifying or accessing internal signals should be rendered difficult.:
      - Packing the signals of interest.
  model:
    - invasive
  surf:
    - instruction skip
    - instruction modification
    - execution flow modification
    - counter-measure deactivation
    - read internal signals
  vect:
    - FIB editing
```

an instruction by physically overwriting some bytes in ROM. We recommend packing signals of interest (i.e., chip regions we want to protect) as a defense.

5.7.1 Threat modeling using ADF. 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. A comprehensive AD catalog of known attacks and guidelines becomes highly advantageous with physical attacks, where public information is limited.

Designers can utilize ADs to architect their designs effectively, enabling them to create targeted guidelines for various aspects of their designs. The ADs might be developed in-house or incorporated from state-of-the-art catalogs produced by hardware security experts. This facilitates easy access to information for the different teams involved in a project.

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. During threat modeling tasks, the policy and attack surface are primary indicators for IC designers. Identifying what needs protection is insufficient. Understanding the attack type and vector is what 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 mitigated by implementing proper scrambling schemes within the ROM. More advanced attackers may be capable of reverse engineering the ROM's scrambling circuit. We can employ encryption to enhance security against such attackers. Correct encryption implementation will force attackers to use fully invasive techniques, significantly reducing the pool of potential attackers. Other dedicated countermeasures can be implemented if the application needs 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.

Listing 11. 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]
```

The database also describes attack techniques such as *FIB* modifications, 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.

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

5.8 Bluetooth Protocol and Implementation Level Attacks

Bluetooth 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 optimized for low-energy consumption applications. It enables efficient data transfer between devices, balancing transmission range, data rate, and power consumption. It provides reliable and secure communication while minimizing energy usage. Bluetooth has two flavors: Classic (BC) and Low Energy (BLE). We focus on BLE as it is supported by real-world crypto wallets (e.g., Ledger Nano X).

We built 46 Bluetooth ADs covering relevant protocol and implementation level Bluetooth threats [42]. We present our ADs in `bt.yaml`. Currently, our file has 18 ADs specific to BLE: nine related to the protocol and nine related implementations. Table 3 in Appendix A, lists our BLE ADs and more information.

Using ADs, we can cover implementation-level and protocol-level threats on BLE, regardless of their level of abstraction. For example, we can model a MitM attack on BLE Secure Simple Pairing (SSP) like No Input No Output (NINO), as shown in Listing 11. In the `nino_ble` AD, we specify 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 of out-of-band pairing with Near Field Communication (NFC)). As a result, a designer might consider adding NFC to the design of the crypto wallet to defend against NINO and other BLE attacks related to a weak association phase.

Another relevant example is BLUR, shown in Listing 12, which models an attack on Bluetooth's Cross-Transport Key Derivation (CTKD). The AD structure allows us to model some of Bluetooth's specific aspects, such as attacking BC from BLE and vice versa or adopting concrete mitigation strategies. For example, we can include disabling weak key overwriting, tracking the associations with paired devices, and preventing role switching by tracking asymmetries in the roles.

5.9 FIDO2 Authentication Attacks

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 private key remains on the authenticator,

Listing 12. 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]
  cve: ["15802", "20361"]
  cwe: ["287"]
```

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.

The FIDO2 authentication process is depicted in Figure 7. The main actors are the hardware authenticator, the client, and the online service (i.e., the relying party). In addition, the relying party database contains the credentials and public keys. As the focus was on the authenticator, we considered the involved entities (i.e., the client and relying party) as internal processes. Moreover, we did not include the user as its only interaction with the system is the press of a button. The data flows are numbered following the FIDO2 message order [134] to capture the time dimension of the protocol.

We built a catalog of 21 ADs for FIDO2 and focused on system and device-level threats. As an extra, we also covered SoloKeys [33] related threats. The ADs are provided in `fido_device.yaml`, `fido_system.yaml`, and `fido_solokey.yaml`.

5.9.1 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 MitM attack between the client and the 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 defense policies such as `robust device` or `secure microcontroller`, for which concrete mechanism specifications depend on the specific token. 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 maintaining high-level attack surfaces (authenticator, client, and relying party). For instance, some of the previously listed threats can be mapped to the STRIDE categories, such as man-in-the-middle attacks and spoofing issues.

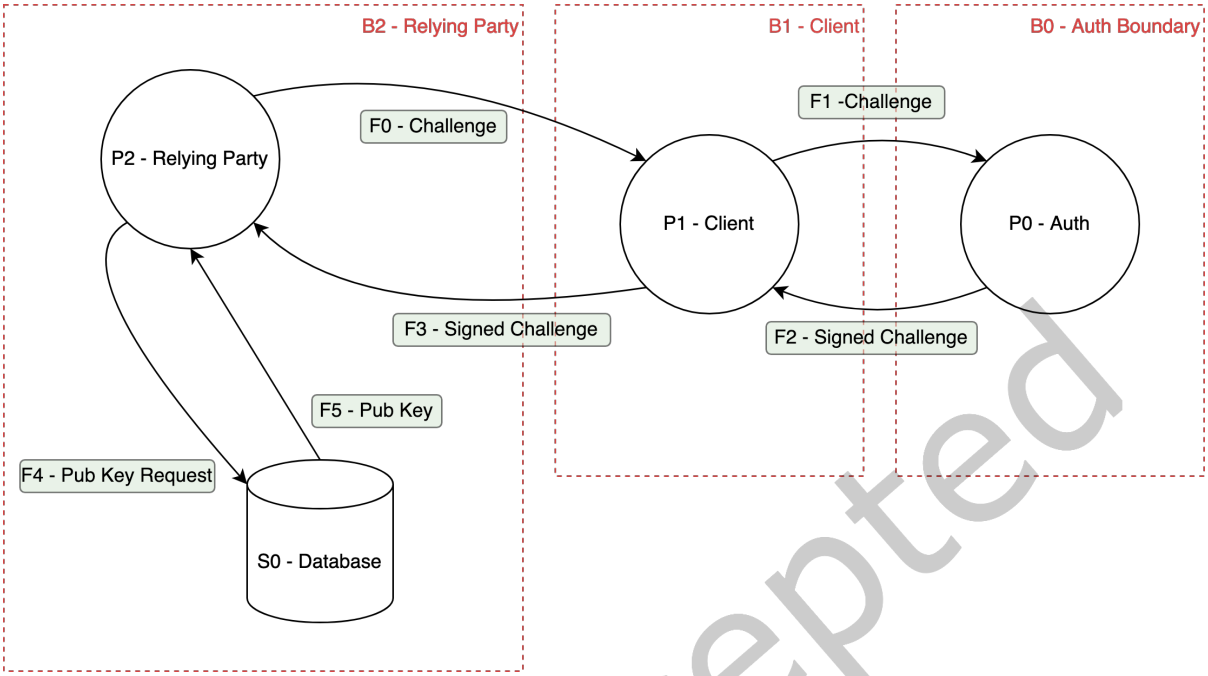

Fig. 7. FIDO2 DFD

5.9.2 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 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 [134] implementation vulnerabilities and bugs, which is the communication protocol between the client and the relying party.

5.9.3 SoloKeys threats. Solo is an open-source FIDO2 token. We reported physical threats associated with a CVE or documented in the SoloKeys online blog [32]. 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 it implements. 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.

6 Related Work

TM Methodologies. Fault trees can be considered the first TM methodology, where each threat represents a failure, and failures are hierarchically represented in a tree [132]. ATree were extended, among others, with attack-defense-trees [66], profiles [69], and case-study driven methodologies [8]. STRIDE was also extended with STRIDE per element and per interaction [127]. ATree were augmented with formal methods [136]. 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 [130].

There have been industrial efforts to standardize threat modeling across platforms such as the OTM (Open Threat Model) format by IriusRisk [58] and MITRE's ATT&CK, an adversary-centric framework designed to threat model post-compromise enterprise security, including lateral movement, and privilege escalation [78, 79]. The framework is constantly growing and 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 [60, 84, 85], the defensive counterpart of the ATT&CK framework. Best practices related to TM methodologies were also created, such as OWASP's threat modeling project [100], and collections of survey and comparison of different TM methodologies [118, 119]. 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 [91, 122, 128].

Vendor-specific TM Methodologies. Vendors also tend to implement their own TM methodologies. Intel proposed TARA (Threat Agent Risk Assessment) [116], an attack-centric methodology based on seven phases. While currently unmaintained, TARA is still used, for example, in the automotive sector. Lockheed Martin proposed the Cyber Kill Chain (CKC) [71] to model cyber intrusion activities, like advanced persistent threats (APTs), also based on seven phases: reconnaissance, weaponization, delivery, exploitation, installation, command and control, and actions.

TM automation tools. There are several open-source TM automation tools [125]. These tools allow, among others, to parse threats from code comments (e.g., Threatspec [35]), build system models and threats catalogs from code or configuration languages (e.g., Pytm, hcltm [44], threagile, and Threat Dragon [101]), and speed up TM using agile best practices, e.g., Rapid Threat Model Prototyping (RTMP) [53]. Moreover, there are commercial TM tools, such as the ones provided by IriusRisk [57] and Tutamantic [129]. The most popular closed-source but free TM tool is Microsoft's Threat Modeling Tool (TMT) [74] 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 [61] the authors show how to adapt STRIDE and TARA to threat model a connected car adhering to the AUTOSAR standard [45]. The NCC Group extended the MS TMT with a template for automotive TM [90].

Other domain-specific areas of TM extension are cyber-physical system (CPS) [63], industrial control systems (ICS) [4], Internet of Things (IoT) [1] and mobile (cellular) networks [107]. Recently, six popular end-to-end messaging applications were evaluated with STRIDE and LINDDUN 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 [105, 135]. 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) [104], Structured Threat Information Expression (STIX) [56], Threat Report ATT&CK Mapper (TRAM) [43], and TheHive incident response platform [34].

Process security. Process security mostly focuses on the hardware and software *supply chains*. Researchers extensively analyzed vulnerable dependencies from package repositories for interpreted programming languages, e.g., Node package manager (npm), Python Package Index (PyPI), and RubyGems [40] and extracted valuable security lessons from the software supply chain [41]. Other works developed attack taxonomies for open-source software via attack trees [68] and uncovered new attack techniques via the software supply chain, such as the

GitHub fork attack vector [15]. Recently, some works started exploring automated ways to analyze the security of closed-source software supply chains [9].

7 Conclusion

We presented ADF, a new framework to augment TM with a novel threat data format (AD object) and related automations (flat and hierarchical representations). ADF satisfies seven requirements, the combination of which is not provided by other TM frameworks (e.g., pytm or threagile). As a result, ADF has comprehensive coverage (attacks, defenses, security, privacy, hardware, firmware, product, and lifecycle) and is (re)usable by humans and machines. We implemented the ADF in the `ADF` toolkit, consisting of four modules: Catalog, Parse, Check, and Analyze, and described its salient technical details. We ran a large-scale evaluation to confirm ADF's usefulness empirically. We involved seven expert groups from academia and industry. We asked them to threat model a crypto wallet in their area of expertise (e.g., hardware, firmware, software, protocol, security, privacy, and lifecycle). They generated 175 ADs spanning heterogeneous and valuable threat classes, like invasive IC manipulations, physical side-channel, fault injection, and secure development lifecycles. The ADF toolkit includes the developed AD objects, it is open-source and available at https://github.com/francozappa/adf.

We discovered that using ADF offers a new way of thinking about process requirements, such as the ISA/IEC 62443-4-1. Specifically, it forces us to think of requirements in terms of threats that the requirement mitigates, thus uncovering the threats themselves. We modeled pre-silicon attacks and defenses on a RISC-V SE for the first time. The ADs' hierarchical structure helped build trees and attack chains, therefore enabling the individuation of defenses against more sophisticated attacks with multiple stages. We built several ADs covering invasive physical attacks, which no documented TM library does. We also managed to model physical SC and FI attack techniques even in cases where no defenses exist, all due to the AD object flexibility.

Acknowledgments

This work was partially funded by the European Union under grant agreement no. 101070008 (ORSHIN project) and by the European Commission through the Horizon 2020 research and innovation program Belfort ERC Advanced Grant 101020005 695305. 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. It was additionally supported by the CyberSecurity Research Flanders with reference number VR20192203. Moreover, it has been partially supported by the French National Research Agency under the France 2030 label (NF-HiSec ANR-22-PEFT-0009) and the Apricot/ENCOPIA ANR MESRI-BMBF project (ANR-20-CYAL-0001). The views reflected herein do not necessarily reflect the opinion of the French government. Jesse De Meulemeester is funded by an FWO fellowship (11PFE24N).

References

[1] Ioannis Agadakos, Chien-Ying Chen, Matteo Campanelli, Prashant Anantharaman, Monowar Hasan, Bogdan Copos, Tancrède Lepoint, Michael Locasto, Gabriela F Ciocarlie, and Ulf Lindqvist. 2017. 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*. 37–48.

[2] FIDO Alliance. 2024. FIDO: Simpler, Stronger Authentication. https://fidoalliance.org/.

[3] FIDO Alliance. 2024. FIDO2. https://fidoalliance.org/fido2/.

[4] Luca Allodi and Sandro Etalle. 2017. 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*. 23–26.

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

[6] Daniele Antonioli, Nils Ole Tippenhauer, Kasper Rasmussen, and Mathias Payer. 2022. 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)*.

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

[8] Maxime Audinot, Sophie Pinchinat, and Barbara Kordy. 2017. 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*. Springer, 83–102.

[9] Frederick Barr-Smith, Tim Blazytko, Richard Baker, and Ivan Martinovic. 2022. 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*. 51–61.

[10] Bernhard J. Berger, Karsten Sohr, and Rainer Koschke. 2016. Automatically Extracting Threats from Extended Data Flow Diagrams. In *Proceedings of the 8th International Symposium on Engineering Secure Software and Systems - Volume 9639* (London, UK) *(ESSoS 2016)*. Springer-Verlag, Berlin, Heidelberg, 56–71. https://doi.org/10.1007/978-3-319-30806-7_4

[11] Eli Biham and Lior Neumann. 2020. Breaking the Bluetooth pairing–the fixed coordinate invalid curve attack. In *Selected Areas in Cryptography–SAC 2019: 26th International Conference, Waterloo, ON, Canada, August 12–16, 2019, Revised Selected Papers 26*. Springer, 250–273.

[12] Marton Bognar, Jo Van Bulck, and Frank Piessens. 2022. Mind the Gap: Studying the Insecurity of Provably Secure Embedded Trusted Execution Architectures. In *43rd IEEE Symposium on Security and Privacy (S&P)*. IEEE, 1638–1655.

[13] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. 1997. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 1233)*. Springer, 37–51.

[14] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. 2019. A Systematic Evaluation of Transient Execution Attacks and Defenses. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 249–266.

[15] Alan Cao and Brendan Dolan-Gavitt. 2022. What the Fork? Finding and Analyzing Malware in GitHub Forks. In *Proc. of NDSS*, Vol. 22.

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

[17] Guillaume Celosia and Mathieu Cunche. 2019. 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*. 24–31.

[18] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO (Lecture Notes in Computer Science, Vol. 1666)*. Springer, 398–412.

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

[20] CISA. 2024. Decider web application. https://github.com/cisagov/Decider/.

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

[22] Jean-Sébastien Coron. 1999. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *Cryptographic Hardware and Embedded Systems - CHES'99 (LNCS, Vol. 1717)*, Çetin Kaya Koç and Christof Paar (Eds.). Springer, 292–302.

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

[24] Graphviz developers. 2024. Graphviz is an open source graph visualization software. https://graphviz.org/.

[25] Graphviz developers. 2024. Package facilitating the creation and rendering of graph descriptions in the DOT language of Graphviz. https://pypi.org/project/graphviz/.

[26] LibYAML developers. 2024. LibYAML - A C library for parsing and emitting YAML. https://github.com/yaml/libyaml.

[27] Linux Kernel Developers. 2024. The Kernel Address Sanitizer (KASAN). https://www.kernel.org/doc/html/latest/dev-tools/kasan.html.

[28] Linux Kernel Developers. 2024. The Kernel Memory Sanitizer (KMSAN). https://www.kernel.org/doc/html/latest/dev-tools/kmsan.html.

[29] Pandas developers. 2024. 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/.

[30] PyYAML developers. 2024. PyYAML is a full-featured YAML framework for the Python programming language. https://pyyaml.org/.

[31] Rust developers. 2024. Rust: A language empowering everyone to build reliable and efficient software. https://www.rust-lang.org/.

[32] SoloKeys developers. 2024. SoloKeys Blog. https://solokeys.com/blogs/news.

[33] SoloKeys developers. 2024. SoloKeys Homepage. https://solokeys.com/.

[34] TheHive developers. 2024. TheHive is a FOSS security incident response platform. https://github.com/TheHive-Project/TheHive.

[35] Threatspec developers. 2024. Threatspec - continuous threat modeling, through code. https://github.com/threatspec/threatspec.

[36] Wordcloud developers. 2024. A little word cloud generator in Python. https://pypi.org/project/wordcloud/.

[37] Matplotlib development team. 2024. Matplotlib: Visualization with Python. https://matplotlib.org/.

[38] Guardian Digital. 2024. Linux Security Advisories. https://linuxsecurity.com/advisories.
[39] Ying Dong, Wenbo Guo, Yueqi Chen, Xinyu Xing, Yuqing Zhang, and Gang Wang. 2019. Towards the Detection of Inconsistencies in Public Security Vulnerability Reports. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 869–885. https://www.usenix.org/conference/usenixsecurity19/presentation/dong
[40] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. 2020. Towards measuring supply chain attacks on package managers for interpreted languages. *arXiv preprint arXiv:2002.01139* (2020).
[41] William Enck and Laurie Williams. 2022. Top five challenges in software supply chain security: Observations from 30 industry and government organizations. *IEEE Security & Privacy* 20, 2 (2022), 96–100.
[42] engn33r. 2024. Awesome Bluetooth Security (BR, EDR, LE, and Mesh). https://github.com/engn33r/awesome-bluetooth-security.
[43] The Center for Threat-Informed Defense. 2024. Threat Report ATT&CK Mapping (TRAM). https://github.com/center-for-threat-informed-defense/tram.
[44] Christian Frichot. 2024. hcltm: Threat Modeling with HCL. https://github.com/xntrik/hcltm.
[45] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkelin, Kenji Nishikawa, and Klaus Lange. 2009. AUTOSAR–A Worldwide Standard is on the Road. In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, Vol. 62. Citeseer, 5.
[46] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic Analysis: Concrete Results. In *CHES (Lecture Notes in Computer Science, Vol. 2162)*. Springer, 251–261.
[47] Matheus E Garbelini, Chundong Wang, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. 2020. Sweyntooth: Unleashing mayhem over Bluetooth Low Energy. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*. 911–925.
[48] Google. 2024. Android Security Bulletins. https://source.android.com/docs/security/bulletin.
[49] Louis Goubin and Jacques Patarin. 1999. DES and Differential Power Analysis (The "Duplication" Method). In *CHES (Lecture Notes in Computer Science, Vol. 1717)*. Springer, 158–172.
[50] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+Flush: A Fast and Stealthy Cache Attack. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9721)*, Juan Caballero, Urko Zurutuza, and Ricardo J. Rodríguez (Eds.). Springer, 279–299. https://doi.org/10.1007/978-3-319-40667-1_14
[51] Marit Hansen, Meiko Jensen, and Martin Rost. 2015. Protection goals for privacy engineering. In *2015 IEEE Security and Privacy Workshops*. IEEE, 159–166.
[52] MG Hardy. 2012. Beyond continuous monitoring: Threat modeling for real-time response. *SANS Institute* (2012).
[53] Geoffrey Hill. 2024. Rapid Threat Model Prototyping (RTMP). https://github.com/geoffrey-hill-tutamantic/rapid-threat-model-prototyping-docs.
[54] Konstantin Hypponen and Keijo MJ Haataja. 2007. "Nino" Man-in-the-Middle attack on Bluetooth Secure Simple Pairing. In *2007 3rd IEEE/IFIP International Conference in Central Asia on Internet*. IEEE, 1–5.
[55] Securin Inc. 2021. Pegasus Spyware Snoops on Political Figures Worldwide. https://www.securin.io/articles/pegasus-spyware-snoops-on-political-figures-worldwide/.
[56] OASIS Cyber Threat Intelligence. 2024. Sharing threat intelligence just got a lot easier! https://oasis-open.github.io/cti-documentation/.
[57] IriusRisk. 2024. IriusRisk is the industry leader in Automated threat modeling and secure software design. https://www.iriusrisk.com/.
[58] IriusRisk. 2024. The Open Threat Modeling Format (OTM) defines a platform independent way to define the threat model of any system. https://github.com/iriusrisk/OpenThreatModel.
[59] Yuning Jiang, Manfred Jeusfeld, and Jianguo Ding. 2021. Evaluating the Data Inconsistency of Open-Source Vulnerability Repositories. In *Proceedings of the 16th International Conference on Availability, Reliability and Security* (Vienna, Austria) *(ARES '21)*. Association for Computing Machinery, New York, NY, USA, Article 86, 10 pages. https://doi.org/10.1145/3465481.3470093
[60] Peter E Kaloroumakis and Michael J Smith. 2021. Toward a knowledge graph of cybersecurity countermeasures. *The MITRE Corporation* (2021), 11.
[61] Adi Karahasanovic, Pierre Kleberger, and Magnus Almgren. 2017. Adapting threat modeling methods for the automotive industry. In *Proceedings of the 15th ESCAR Conference*. 1–10.
[62] Vladimir Keleshev. 2024. Schema validation just got Pythonic. https://github.com/keleshev/schema.
[63] Rafiullah Khan, Kieran McLaughlin, David Laverty, and Sakir Sezer. 2017. STRIDE-based threat modeling for cyber-physical systems. In *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. IEEE, 1–6.
[64] 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. 2019. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 1–19.
[65] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *Advances in Cryptology - CRYPTO '99 (LNCS, Vol. 1666)*, Michael J. Wiener (Ed.). Springer, 388–397.

[66] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. 2011. 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*. Springer, 80–95.
[67] Vaibhav Garg Kristen Tan. 2022. An Analysis of Open-source Automated Threat Modeling Tools and Their Extensibility from Security into Privacy. https://www.usenix.org/publications/loginonline/analysis-open-source-automated-threat-modeling-tools-and-their
[68] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. 2022. Taxonomy of attacks on open-source software supply chains. *arXiv preprint arXiv:2204.04008* (2022).
[69] Aleksandr Lenin, Jan Willemson, and Dyan Permata Sari. 2014. 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*. Springer, 199–212.
[70] Tong Li, Elda Paja, John Mylopoulos, Jennifer Horkoff, and Kristian Beckers. 2016. Security attack analysis using attack patterns. In *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*. 1–13. https://doi.org/10.1109/RCIS.2016.7549303
[71] Lockheed Martin. 2022. The Cyber Kill Chain. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.
[72] Microsoft. 2024. Microsoft Security Development Lifecycle (SDL). https://www.microsoft.com/en-us/securityengineering/sdl/.
[73] Microsoft. 2024. Microsoft Threat Modeling Tool threats. https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats.
[74] Microsoft. 2024. Microsoft Threat Modeling Tool (TMT). https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool.
[75] Mitre. 2021. CWE Most Important Hardware Weaknesses 2021. https://cwe.mitre.org/scoring/lists/2021_CWE_MIHW.html.
[76] Mitre. 2022. CWE Top 25 Most Dangerous Software Weaknesses 2022. https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html.
[77] Mitre. 2023. CWE Top 25 Most Dangerous Software Weaknesses 2023. https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html.
[78] Mitre. 2024. ATT&CK framework. https://attack.mitre.org/.
[79] Mitre. 2024. ATT&CK framework GitHub project. https://github.com/mitre-attack.
[80] Mitre. 2024. CAPEC-668: Key Negotiation of Bluetooth Attack (KNOB). https://capec.mitre.org/data/definitions/668.html.
[81] Mitre. 2024. Common Attack Pattern Enumerations and Classifications. https://capec.mitre.org/.
[82] Mitre. 2024. Common Vulnerabilities and Exposures. https://www.cve.org/.
[83] Mitre. 2024. Common Weakness Enumeration. https://cwe.mitre.org/.
[84] Mitre. 2024. D3FEND framework. https://d3fend.mitre.org/.
[85] Mitre. 2024. D3FEND framework GitHub project. https://github.com/d3fend.
[86] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. 2013. Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller. In *FDTC*. IEEE Computer Society, 77–88.
[87] Michael Muckin and Scott C Fitch. 2014. A threat-driven approach to cyber security. *Lockheed Martin Corporation* (2014).
[88] musl developers. 2024. musl libc for Linux git repository. https://git.musl-libc.org/cgit/musl.
[89] musl developers. 2024. musl libc for Linux homepage. https://musl.libc.org/.
[90] nccgroup. 2016. The Automotive Threat Modeling Template. https://github.com/nccgroup/The_Automotive_Threat_Modeling_Template.
[91] Rusty Newton. 2023. Threat Modeling Example with ChatGPT. https://blog.infosec.business/how-to-use-chatgpt-to-learn-threat-modeling/.
[92] NIST. 2017. An Introduction to Privacy Engineering and Risk Management in Federal Systems. https://csrc.nist.gov/publications/detail/nistir/8062/final.
[93] NIST. 2024. Common Vulnerability Scoring System (CVSS). https://nvd.nist.gov/vuln-metrics/cvss.
[94] NIST. 2024. CVSS v2 Calculator. https://nvd.nist.gov/vuln-metrics/cvss/v2-calculator.
[95] NIST. 2024. CVSS v3 Calculator. https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator.
[96] NIST. 2024. CVSS v4. https://www.first.org/cvss/v4-0.
[97] openhwgroup. 2024. OpenHW Group CORE-V CV32E40S RISC-V IP. https://github.com/openhwgroup/cv32e40s.
[98] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache Attacks and Countermeasures: The Case of AES. In *Topics in Cryptology – CT-RSA 2006*, David Pointcheval (Ed.). 1–20.
[99] OWASP. 2021. OWASP Top Ten. https://owasp.org/www-project-top-ten/.
[100] OWASP. 2024. OWASP Threat Modeling Project. https://owasp.org/www-project-threat-model.
[101] OWASP and Mike Goodwin. 2024. Threat Dragon is a free, open-source, cross-platform threat modeling application. https://github.com/OWASP/threat-dragon.
[102] Shengyi Pan, Lingfeng Bao, Xin Xia, David Lo, and Shanping Li. 2023. Fine-grained Commit-level Vulnerability Type Prediction by CWE Tree Structure. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 957–969. https://doi.org/10.1109/ICSE48619.2023.00088
[103] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 565–581.

[104] OpenCTI Platform. 2024. OpenCTI allows orgs to manage cyber threat intelligence knowledge and observables. https://github.com/OpenCTI-Platform/opencti.
[105] MISP Project. 2024. MISP - Threat Intelligence Sharing Platform. https://github.com/MISP/MISP.
[106] Jean-Jacques Quisquater and David Samyde. 2001. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In *E-smart (Lecture Notes in Computer Science, Vol. 2140)*. Springer, 200–210.
[107] Siddharth Prakash Rao, Hsin-Yi Chen, and Tuomas Aura. 2023. Threat modeling framework for mobile communication systems. *Computers & Security* 125 (2023), 103047.
[108] Xida Ren, Logan Moody, Mohammadkazem Taram, Matthew Jordan, Dean M. Tullsen, and Ashish Venkat. 2021. I See Dead μops: Leaking Secrets via Intel/AMD Micro-Op Caches. In *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, June 14-18, 2021*. IEEE, 361–374. https://doi.org/10.1109/ISCA52012.2021.00036
[109] Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick. 2020. Frankenstein: Advanced wireless fuzzing to exploit new Bluetooth escalation targets. In *Proceedings of the 29th USENIX Conference on Security Symposium*. 19–36.
[110] Mike Ryan. 2019. Crackle: crack and decrypt BLE encryption. https://github.com/mikeryan/crackle, Accessed: 2019-07-30.
[111] Chris Salter, O Sami Saydjari, Bruce Schneier, and Jim Wallner. 1998. Toward a secure system engineering methodolgy. In *Proceedings of the 1998 workshop on New security paradigms*. 2–10.
[112] Jörn-Marc Schmidt and Christoph Herbst. 2008. A Practical Fault Attack on Square and Multiply. In *FDTC*. IEEE Computer Society, 53–58.
[113] Christian Schneider. 2024. Threagile is an open-source toolkit for agile threat modeling:. https://github.com/Threagile/threagile.
[114] Bruce Schneier. 1999. Attack trees. *Dr. Dobb's journal* 24, 12 (1999), 21–29.
[115] Bruce Schneier. 2021. Chinese Supply-Chain Attack on Computer Systems. https://www.schneier.com/blog/archives/2021/02/chinese-supply-chain-attack-on-computer-systems.html.
[116] Intel Security. 2009. 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.
[117] Ben Seri, Gregory Vishnepolsky, and Dor Zusman. 2019. BLEEDINGBIT: The hidden Attack Surface within BLE chips.
[118] Nataliya Shevchenko. 2018. Threat Modeling: 12 Available Methods. https://insights.sei.cmu.edu/blog/threat-modeling-12-available-methods/.
[119] Nataliya Shevchenko, Timothy A Chick, Paige O'Riordan, Thomas P Scanlon, and Carol Woody. 2018. Threat modeling: a summary of available methods.
[120] Adam Shostack. 2008. Experiences Threat Modeling at Microsoft. In *CEUR Workshop*.
[121] Adam Shostack. 2014. *Threat modeling: Designing for security*. John Wiley & Sons.
[122] Adam Shostack. 2023. More on GPT-3 and threat modeling. https://shostack.org/blog/more-on-gpt3/.
[123] Sergei P. Skorobogatov and Ross J. Anderson. 2002. Optical Fault Induction Attacks. In *CHES (Lecture Notes in Computer Science, Vol. 2523)*. Springer, 2–12.
[124] Rock Stevens, Daniel Votipka, Elissa M Redmiles, Colin Ahern, Patrick Sweeney, and Michelle L Mazurek. 2018. The Battle for New York: A Case Study of Applied Digital Threat Modeling at the Enterprise Level. In *USENIX Security Symposium*. 621–637.
[125] Kristen Tan and Vaibhav Garg. 2022. An Analysis of Open-source Automated Threat Modeling Tools and Their Extensibility from Security into Privacy. https://www.usenix.org/publications/loginonline/analysis-open-source-automated-threat-modeling-tools-and-their. *USENIX Login* (2022).
[126] Izar Tarandach. 2024. pytm: A Pythonic framework for threat modeling. https://github.com/izar/pytm.
[127] Izar Tarandach and Matthew J Coles. 2021. *Threat modeling: a practical guide for development teams*. O'Reilly.
[128] Google Cloud Security Team. 2023. Sec-PaLM: Supercharging security with generative AI. https://cloud.google.com/blog/products/identity-security/rsa-google-cloud-security-ai-workbench-generative-ai.
[129] Tutamantic. 2024. Tutamen Threat Model Automator. https://www.tutamantic.com/.
[130] Tony UcedaVelez and Marco M Morana. 2015. *Risk Centric Threat Modeling: process for attack simulation and threat analysis*. John Wiley & Sons.
[131] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2018. Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 178–195.
[132] William E Vesely, Francine F Goldberg, Norman H Roberts, and David F Haasl. 1981. Fault tree handbook.
[133] Maximilian von Tschirschnitz, Ludwig Peuckert, Fabian Franzen, and Jens Grossklags. 2021. Method confusion attack on Bluetooth pairing. In *2021 IEEE symposium on security and privacy (SP)*. IEEE, 1332–1347.
[134] W3C. 2021. Web Authentication: An API for accessing Public Key Credentials - Level 2. https://www.w3.org/TR/webauthn/.
[135] Cynthia Wagner, Alexandre Dulaunoy, Gérard Wagener, and Andras Iklody. 2016. 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*. ACM, 49–56.

[136] Wojciech Wideł, Maxime Audinot, Barbara Fila, and Sophie Pinchinat. 2019. Beyond 2014: Formal Methods for Attack Tree–based Security Modeling. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 1–36.
[137] Jake Williams. 2020. 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/.
[138] TMM working group. 2020. Threat Modeling Manifesto. https://www.threatmodelingmanifesto.org.
[139] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave (Jing) Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. 2020. BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy. In *WOOT, USENIX Security Symposium*.
[140] Kim Wuyts, Riccardo Scandariato, and Wouter Joosen. 2014. LINDDUN privacy threat tree catalog.
[141] Kim Wuyts, Laurens Sion, and Wouter Joosen. 2020. Linddun GO: A lightweight approach to privacy threat modeling. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 302–309.
[142] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 640–656. https://doi.org/10.1109/SP.2015.45
[143] yaml developers. 2024. YAML: YAML Ain't Markup Language™. https://yaml.org/.
[144] yamllint developers. 2024. YAMLlint: the YAML validator. https://www.yamllint.com/.
[145] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, Kevin Fu and Jaeyeon Jung (Eds.). USENIX Association, 719–732.
[146] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. 2020. Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks. In *USENIX Security Symposium*. 37–54.

A Tables

Table 2. The 11 taxonomies supported by ADF's `get_map`

The 11 taxonomies currently supported by ADF's `get_map`. UIT refers to a privacy taxonomy from International Association of Privacy Professionals (IAPP) [51], PMD to a privacy taxonomy from NISTIR 8062 [92], OTT to the OWASP Top Ten Web Application Security Risks [99], CKC to the Cyber Kill Chain by Lockheed Martin [71], CWETH21 to the top ten hardware CWE from 2021 [75], and CWETS22 to the top twenty-five software CWE from 2022 [76] and 2023 [77]

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
CWETS23	787, 79, 89, 416, 78, 20, 125, 22, 352, 434, 862, 476, 287, 190, 502, 77, 119, 798, 918, 306, 362, 269, 94, 863, 276

Table 3. List of 18 BLE ADs from our catalog of 46 Bluetooth ADs

ad_name	a	tag
sco_ble	Downgrade attacks on BLE SCO [146]	Impl
sweyntooth_ble_1	Link Layer Length Overflow [47]	Impl
sweyntooth_ble_2	Link Layer LLID Deadlock [47]	Impl
sweyntooth_ble_3	BLE Crafted packet buffer overflow [47]	Impl
sweyntooth_ble_4	Key Size Overflow [47]	Impl
sweyntooth_ble_5	Zero LTK Installation [47]	Impl
blesa_ble	BLE reconnection spoofing [139]	Impl
bleedingbit_ble_1	Malformed packet BoF in BLE beacons parsing [117]	Impl
frankenstein_ble_1	Heap overflow in BLE PDUs parsing [109]	Impl
knob_ble	Key Negotiation of Bluetooth (KNOB) [5]	Proto
blur_ble	BLUR Cross-Transport Key Derivation attacks [6]	Proto
nino_ble	MitM on BLE SSP [54]	Proto
bluemirror_ble	Reflection attack on passkey entry [21]	Proto
invcurve_ble	Invalid Curve Attack [11]	Proto
pairing_meth_conf_ble	Method confusion attack [133]	Proto
crackle_ble	BLE Key Derivation [110]	Proto
injectable	PHY packet injection [16]	Proto
gatt_fp_ble	GATT Fingerprinting and Tracking [17]	Proto

Received 15 February 2024; revised 15 February 2024; accepted 7 September 2024