Design as code: facilitating collaboration between usability and security engineers using CAIRIS.

FAILY, S. and IACOB, C.

2017

Design as Code: Facilitating Collaboration between Usability and Security Engineers using CAIRIS

Shamal Faily
Bournemouth University
Poole, UK
sfaily@bournemouth.ac.uk

Claudia Iacob
University of Portsmouth
Portsmouth, UK
iacob@port.ac.uk

***Abstract*—Designing usable and secure software is hard without tool-support. Given the importance of requirements, CAIRIS was designed to illustrate the form tool-support for specifying usable and secure systems might take. While CAIRIS supports a broad range of security and usability engineering activities, its architecture needs to evolve to meet the workflows of these stakeholders. To this end, this paper illustrates how CAIRIS and its models act as a vehicle for collaboration between usability and security engineers. We describe how the modified architecture of CAIRIS facilitates this collaboration, and illustrate the tool using three usage scenarios.**

***Index Terms*—CAIRIS; SaaS; Security; Usability; Personas; Threat Modeling; KAOS; Risks; DevOps.**

I. Introduction

Security by Design entails designing security into software up-front. However, designing for use is important when designing for security; unusable software is likely to be insecure as errors or violations may expose exploitable vulnerabilities. While simple in theory, Security by Design is difficult to achieve as functionality and productivity always takes precedence. Moreover, due to the *bounded rationality* bias, decision making around design is limited by incomplete or inaccurate information, and limits in time and resources for considering all possibilities associated with a given design option [1].

Tool-support can deal with the bounded rationality by helping automate the elicitation, analyses, and management of information used to create models for design and engineering. Requirements are focal point when designing software for security and usability [2], but to evolve Security & Privacy Requirements Engineering, software tools need to support not only requirement specifications, but other forms of associated data [3]. Without this information, stakeholders may find it difficult to validate a design, or the analysis upon which design decisions have been made.

Because no one tool facilitates the specification of usable and secure systems, we developed CAIRIS (Computer Aided Integration of Requirements and Information Security) to illustrate the form tool support for requirements, security, and usability engineering might take [4]. One of the main features of CAIRIS is its ability to not only specify elements of a system, but also the environments within which systems might be situated. This allows users to compare and contrast the impact that changes to a context of use might have on a system's usability and security.

In addition to supporting requirements engineers, we have found that CAIRIS has the potential for broader use supporting interaction design and security engineering. For example, in the study described in [5], CAIRIS was used to specify context of use descriptions, persona and attacker persona development, requirements sense-making activities, and architectural risk analysis. The study also illustrated how subjecting design models to version control and build processes, and making them publicly available via github increased project developers' engagement with regards to maintaining requirements, security, and architectural design models. Such behaviours complement *DevOps* practices that emphasise collaboration between design, development, and operations teams via automated processes that consider "infrastructure-as-code" [6]. However, in this study, CAIRIS was used by designers who each had a working knowledge of requirements engineering, security engineering, and user-centred design. In the wider world, many project teams have practitioners with individual expertise in only one, and occasionally two, of these areas. While practitioners with expertise in one area might appreciate the capabilities offered by CAIRIS, they are unlikely to expend effort learning a tool that might impact their current processes, tools, and workflows. There is, therefore, a need to evolve the design of CAIRIS to better facilitate collaboration by encompassing not only security requirements engineering, but also tool and process interoperability with security and usability engineering.

In recent years, one of the drivers for interoperability in software design and engineering has been the growth in *Software-as-a-Service* (SaaS) platforms. SaaS is a cloud-based delivery model where a software application is offered as a managed service [7]. Such platforms allow applications to be accessible via modern web browsers running on any platform. Porting CAIRIS to a SaaS platform helps achieve our goals for process and tool interoperability in three ways. First, CAIRIS would no longer require its users to be co-located, so security, software, and usability engineers could independently elicit their design elements based on the input from others. Second, CAIRIS users can use not only any operating system, desktop environment, and tool chain they wish, they can potentially extend CAIRIS or other tools using web service APIs. Finally, as an open-source project, the design and evolution of CAIRIS would benefit from a growing community of developers with the knowledge and expertise to leverage web and cloud platforms.

To explore these possibilities, this paper presents CAIRIS as a vehicle for collaboration between usability and security engineers. We present the revised CAIRIS architecture in Section II, before describing how it facilitates workflows in usability and security engineering in Section III. After walking through a number of usage scenarios in Section IV, we conclude by discussing related tools, and some implications of our work on CAIRIS in Sections V and VI respectively.

II. Architecture

CAIRIS was developed in Python, MySQL stored procedures, and JavaScript. The source code has been released under an Apache Software License, and is freely available from GitHub [8]. Unlike the original iteration of CAIRIS, which is a single-user Linux-based desktop application, CAIRIS was re-designed to be used by multiple users simultaneously, using a modern web browser on any mobile or desktop operating system.

CAIRIS services run on a single machine, virtual machine, or Docker container. Like the desktop version of CAIRIS, the data for a single CAIRIS project is stored in a MySQL database; its database schema is based on the IRIS meta model [2]. When exported, each database is serialised as an XML model file based on a public Document Type Definition (DTD). However, different aspects of a CAIRIS model, such as goals, usability, and architecture, can be exported to individual XML models conforming to specific DTDs.

Although the architecture of CAIRIS builds on the design of the original desktop application, it differs in a number of ways described below.

A. Multi-user support

The CAIRIS desktop application dealt with multiple users via database locking. With very limited state information held in the desktop application at a given time, this is an effective solution when a small number of desktop clients interact with a single CAIRIS database.

In the new architecture, a single CAIRIS server might support a larger number of users. CAIRIS services interact with CAIRIS databases via a proxy object associated with each user session. These services are provided to clients via Flask [9]. Flask-Security [10] is used to provide authentication services, and Apache mod_wsgi-express [11] hosts the web interfaces that client applications communicate with. These open-source software components were selected as they have a usage track record in multi-user SaaS applications, and benefit from an active community of maintainers in the event of unforeseen bugs and compatibility issues in the future.

B. User Interface Design

Although based on the desktop version of CAIRIS, the interface design of CAIRIS was modified to reflect the different stakeholders that might use it. The interface design of the CAIRIS desktop tool assumed a user primarily interested in requirements management. As such, the user interface was based on a spreadsheet metaphor, where requirements were entered into a table, but additional design elements were entered into modal dialog boxes. In contrast, although tables are still used to edit requirements, requirements no longer dominate the CAIRIS user interface. The home screen of the *CAIRIS GUI* acts as a data dashboard, and the menu options have been re-organised to group functionality into the areas that would be primarily used by requirements engineers (*Requirements*), security engineers (*Risk*), and interaction designers (*UX*).

C. Model management and design build processes

CAIRIS model files might contain all or part of not only the system being specified, but also elements of its context of use, or potential threat models. Such model files allow for different configurations of a system or its environment to be incrementally added by different design stakeholders.

Although the desktop application supports multiple versions of a single requirement, it can also be useful to version other CAIRIS model elements as well. Supporting versioning for each model element would not only be disruptive to the current CAIRIS architecture, but could have a negative impact on performance given the database changes required. However, based on our experiences using CAIRIS in [5], the easiest and most effective way of providing highly granular version control to a CAIRIS model file is to add it to a version control system such as *git*. Because the model files are text based, modern version control systems are adept at keeping track of model changes and their impact.

This form of data interchange is illustrated by the specification of the *webinos* platform in GitHub [12]. This repository is a collection of CAIRIS models of software requirements, persona specifications, architectural models, and attack patterns; it also contains model data in other text file and spreadsheet formats. A build script is used to convert non-CAIRIS model data into CAIRIS models, incrementally import the different models into CAIRIS, and export model data from CAIRIS into different representation formats. The failure of CAIRIS model "build" processes might occur because of some design model inconsistency. Identifying inconsistencies at this stage reduces the risk of errors or ambiguity being introduced into downstream design and development activities.

D. CAIRIS APIs

The CAIRIS desktop tool facilitates data interchange via a collection of import and export scripts; these convert data from various sources into CAIRIS model files, and export CAIRIS data back to other formats. Although these scripts are still available to the CAIRIS server, the server exposes a collection of web services, which are accessible via HTTP-based RESTful APIs. Once authenticated, these APIs are used both by the CAIRIS GUI, and other web application clients. The APIs are used to create, read, update, and delete CAIRIS model elements, retrieve visual models, import models into CAIRIS, and export models and documentation from CAIRIS.

E. Visual Model Generation

One of the features of CAIRIS is the ability to automatically generate visual models based on data within a CAIRIS database. Rather than specification documents being the sole means of information exchange about a system design, the impact of a design change can be observed in real time, e.g. the impact of changing an asset property to related risks, or the impact a risk countermeasure might have on the security and usability of an emerging design.

CAIRIS currently supports 12 different visual models, and each model view displays a sub-set of elements associated with different aspects of the IRIS meta-model [2], e.g. risk views, task views, etc. Visual models are based on Graphviz [13] models of nodes and edges. The declarative data that feeds into this model is based on the results of a CAIRIS database query on the elements associated with a particular model view based on the IRIS meta-model. Based on this, a representation of this model is built up with information about how the model should be laid out on screen. Using components from the Pycairo and XDot open source packages, this data is rendered in SVG, which can be displayed in most modern web browsers running the CAIRIS GUI.

F. Documentation Generation

CAIRIS can automatically generate readable specification documentation based on the contents of a CAIRIS database. Documentation generation is a two-step process. The first step involves rendering the CAIRIS model as XML conforming to the DocBook standard [14]; the chapters of this document conform to the Volere Requirements Specification Template [15]. The second step involves taking the generated, machine-readable DocBook file, together with the various graphic files for the visual models as input, and using DocBook to LaTeX conversion tools to create a human readable output format; the formats currently supported by CAIRIS are RTF and PDF.

III. Facilitating Usability and Security Engineering Workflows

To support collaboration, CAIRIS needs to facilitate rather than inhibit good practice for usability and security engineering. To this end, we describe how CAIRIS supports typical early stage usability and security engineering activities.

A. Persona Creation

Because potential vulnerabilities and threats can be identified during empirical data collection, persona development is useful when designing for security [16]. The data collection and analysis carried out also identifies tensions faced by users interacting with security mechanisms [17].

We devised a tool-supported workflow for the elicitation and creation of personas and their grounding; this is illustrated by the video in [18]. The workflow is driven by the *Persona Helper* [19] tool: a Chrome extension developed to illustrate the use of the CAIRIS web services API. The Persona Helper allows users to create factoids based on highlighted text on a web page.

Once factoids have been elicited, affinity diagramming is typically carried out by designers to group factoids to discover persona characteristics. Previous work described how structuring these groups as argumentation models provides assurances about the processes and data used to build personas [20]. To tool-support the processes for grouping factoids, and deriving persona characteristics using argumentation models, we created the ability to export factoid data to and from Trello: an online tool for creating online cards that users can then group [21]; this was made possible via the use of the Trello API. Factoids are exported from CAIRIS as cards, which can be grouped in the same way as factoids in affinity diagrams, and annotated to indicate the contribution made to each persona characteristic. We also added functionality to automatically generate persona characteristics in CAIRIS from imported Trello boards.

This workflow makes it possible for a scribe to either capture affinity diagrams as they are created, or for the Trello board to be used for online affinity diagramming if designers are not co-located.

B. Threat Modelling

1) Data Flow Diagrams: Many approaches to threat modelling begin with some form of system decomposition activity. Several approaches to security and privacy threat modelling [22][23] prescribe the use of *Data Flow Diagrams* (DFDs); these are graphical models that model the flow of information (data flows) between external human or system actors external to the system (entities), activities that manipulate data (processes), and persistent data storage (data stores).

As part of our work, we introduced the concept of data flows to the CAIRIS meta-model. To capture the security implications of data flows, each data flow in CAIRIS carries one or more information assets. We also aligned pre-existing elements of the meta-model to correspond with DFD elements. System, hardware, or people assets in CAIRIS were aligned with the entities, use cases were aligned with processes, and information assets were aligned with data stores.

2) Attack Trees: Attack trees are a formal, methodical way of describing the security of systems [24]. As one of the most commonly used threat modelling techniques, they are used by security engineers as a lightweight ideation technique for modelling attacks to find their root causes; they are typically created using a paper and pen, or a whiteboard. However, once the insights have been drawn from the attack trees and incorporated into other design models, they are largely forgotten.

CAIRIS supports KAOS goal and obstacle models [25], where goals model requirements the system needs to specify, and obstacles are conditions that prevent goals from being achieved. Because obstacle models are represented using the same top-down approach notation as attack trees, they are a good candidate for representing the attacks.

We developed functionality for importing attack trees marked up in DOT [26] into CAIRIS. When these are imported into a specific environment, generated obstacles can be associated with risk analysis elements; this makes it possible to see how goals are obstructed, and how obstacles are resolved as risks are mitigated.

IV. Usage Scenarios

We now present three usage scenarios to demonstrate how CAIRIS supports security and usability engineering. The context for these scenarios is the design of a new control system for a hypothetical water company. Collaborating on the design and engineering of this system is an interaction designer (Alice), and a security engineer (Bob).

Although not explicitly mentioned in the scenarios, both Alice and Bob interact with the same version controlled CAIRIS model; once they have made changes to their model, they commit their changes to make them available to the wider project team. Even though Alice and Bob work with the same model, they are not using the same instance of CAIRIS. Alice relies on a CAIRIS instance running on a Docker container hosted by her laptop, whereas Bob uses an instance of CAIRIS hosted by a cloud-based service provider.

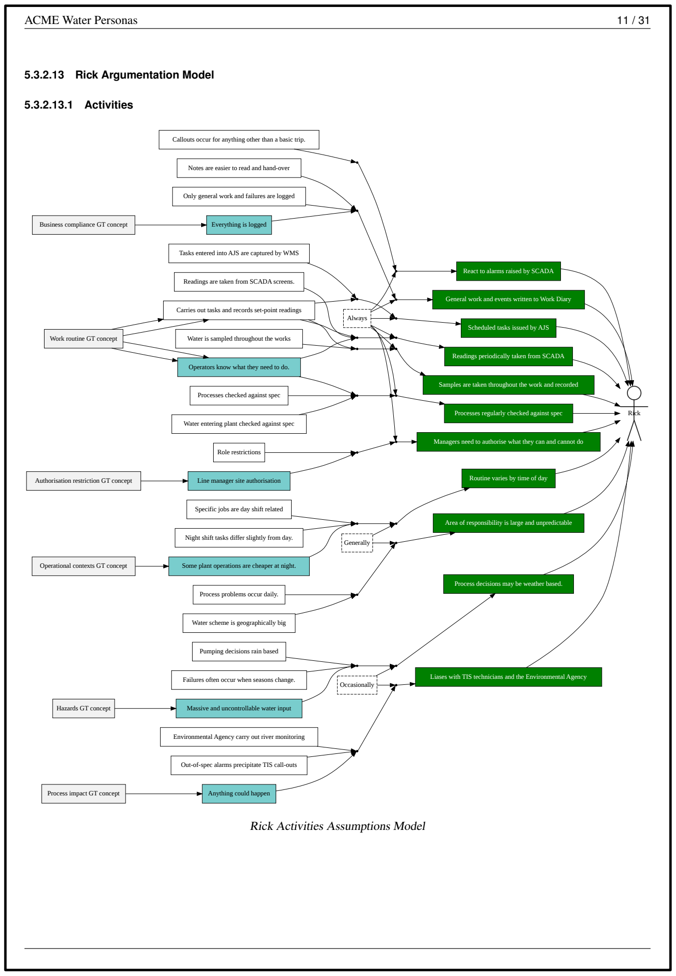
A. Creating Assured Personas

To understand the user goals of people using the new control system, Alice interviewed several plant operators. Once anonymised, Alice rendered her interview transcripts as HTML pages, before opening them up in Google Chrome. She then connected her Persona Helper extension to CAIRIS, and coded her transcripts for factoids. She used CAIRIS to export the factoids into Trello in readiness for affinity diagramming with two of her colleagues. Unfortunately, her colleagues were not in the office that day, but they agreed to use Trello to affinity diagram online while talking on Skype. Once affinity diagramming had concluded, and all factoids were annotated, she exported the Trello board to CAIRIS to generate persona characteristics; she then entered narrative text describing the persona into CAIRIS based on these characteristics. Finally, to quickly share both the persona and the latest system design with colleagues and the security team, she generated a specification document based on the latest version of the CAIRIS model (Figure 1).

B. Discovering Risks from Attackers and Threat Models

A team of penetration testers were scheduled to evaluate the new control system by probing its network infrastructure. Bob wanted the environment to be resistant to intrusive probes of the system for vulnerabilities before this testing took place. To help others understand more about these testers, he decided to create a sketch of a potential penetration tester to illustrate his goals and motivations, and carried out threat modelling to understand how – based on the penetration tester's capabilities and motivations – such probing might be made possible.

Bob and his colleagues in the security team drew an attack tree to model this threat. Once complete, Bob marked up the attack tree in DOT; the Graphviz rendered tree is shown in Figure 2 (left). Bob noted that detecting application scans is a new requirement the infrastructure needs to support, but


ACME Water Personas 11 / 31

5.3.2.13 Rick Argumentation Model

5.3.2.13.1 Activities

Callouts occur for anything other than a basic trip.
Notes are easier to read and hand-over
Only general work and failures are logged
Business compliance GT concept
Everything is logged
Tasks entered into AJS are captured by WMS
Readings are taken from SCADA screens.
Carries out tasks and records set-point readings
Always
Work routine GT concept
Water is sampled throughout the works
Operators know what they need to do.
Processes checked against spec
Water entering plant checked against spec
Role restrictions
Authorisation restriction GT concept
Line manager site authorisation
Specific jobs are day shift related
Night shift tasks differ slightly from day.
Generally
Operational contexts GT concept
Some plant operations are cheaper at night.
Process problems occur daily.
Water scheme is geographically big
Pumping decisions rain based
Failures often occur when seasons change.
Occasionally
Hazards GT concept
Massive and uncontrollable water input
Environmental Agency carry out river monitoring
Out-of-spec alarms precipitate TIS call-outs
Process impact GT concept
Anything could happen
React to alarms raised by SCADA
General work and events written to Work Diary
Scheduled tasks issued by AJS
Readings periodically taken from SCADA
Samples are taken throughout the work and recorded
Processes regularly checked against spec
Managers need to authorise what they can and cannot do
Routine varies by time of day
Area of responsibility is large and unpredictable
Process decisions may be weather based.
Liases with TIS technicians and the Environmental Agency
Rick

Rick Activities Assumptions Model

Fig. 1. Generated requirements specification document, which includes persona definitions and underpinning argumentation models

he would need to consider the implications that incomplete Intrusion Detection System (IDS) rules might have.

Bob imported the attack tree into CAIRIS to generate a new KAOS obstacle model. He defined a threat in CAIRIS to summarise how the penetration tester might enumerate the system (Enumeration), and a new vulnerability based on incomplete IDS rules (Incomplete Intrusion Detection rules), which he linked to the appropriate leaf node of the obstacle model (Incomplete IDS ruleset). As the threat could exploit the vulnerability in the same context of use, he captured this as a risk (System enumeration). CAIRIS automatically rated the risk based on the likelihood of the threat (Probable), severity of the vulnerability (Critical), and security properties the penetration tester aims to compromise; the algorithm for rating risk in CAIRIS is described in more detail in [4]. Before the risk could be confirmed, Bob was required to write a misuse case scenario contextualising the risk and its supporting elements; this helped Bob confirm the analysis contributing to the risk was sound, and provided additional rationale for others wishing examine his reasoning.

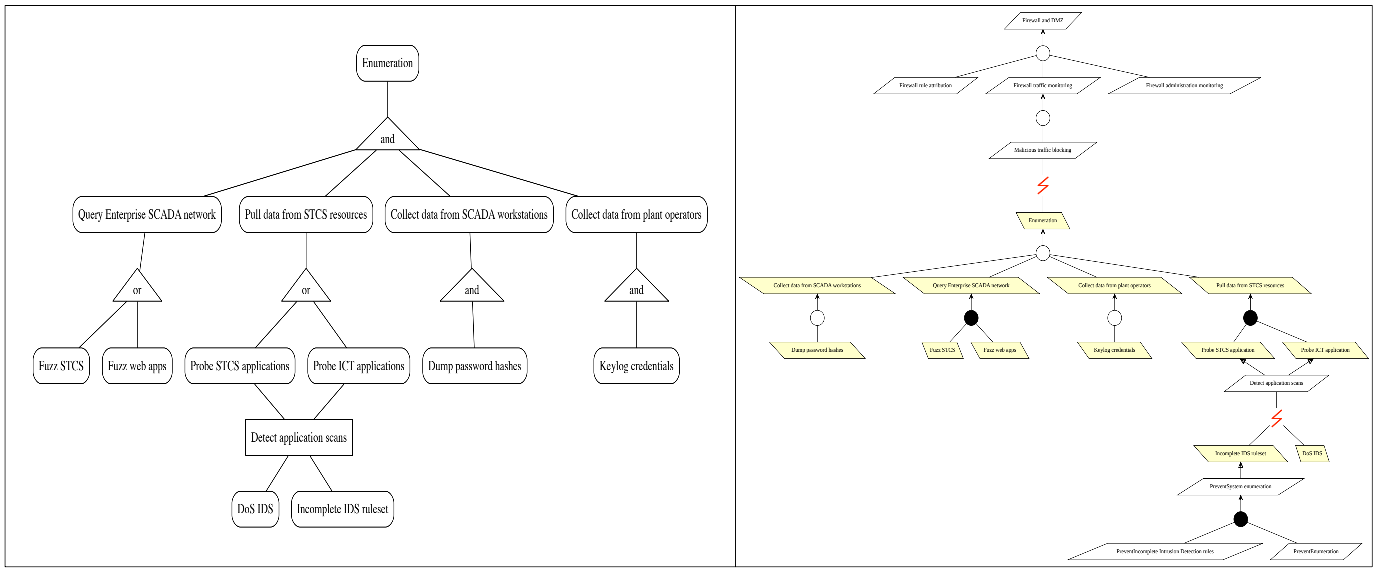
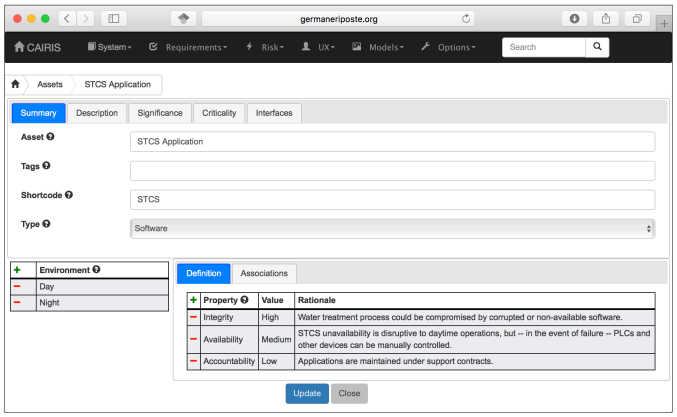

Enumeration
and
Query Enterprise SCADA network
Pull data from STCS resources
Collect data from SCADA workstations
Collect data from plant operators
or
or
and
and
Fuzz STCS
Fuzz web apps
Probe STCS applications
Probe ICT applications
Dump password hashes
Keylog credentials
Detect application scans
DoS IDS
Incomplete IDS ruleset
Firewall and DMZ
Firewall rule attribution
Firewall traffic monitoring
Firewall administration monitoring
Malicious traffic blocking
Enumeration
Collect data from SCADA workstations
Query Enterprise SCADA network
Collect data from plant operators
Pull data from STCS resources
Dump password hashes
Fuzz STCS
Fuzz web apps
Keylog credentials
Probe STCS application
Probe ICT application
Detect application scans
Incomplete IDS ruleset
DoS IDS
PreventSystem enumeration
PreventIncomplete Intrusion Detection rules
PreventEnumeration

Fig. 2. Enumeration threat modelled as an attack tree (left), and KAOS goal model integrating threat model into the network security requirements (right)

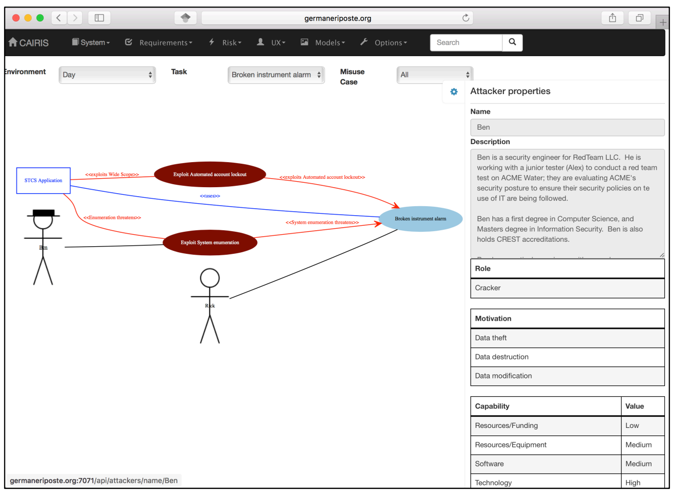


Asset	STCS Application
Tags	
Shortcode	STCS
Type	Software

+	Environment
−	Day
−	Night

+	Property	Value	Rationale
−	Integrity	High	Water treatment process could be compromised by corrupted or non-available software.
−	Availability	Medium	STCS unavailability is disruptive to daytime operations, but -- in the event of failure -- PLCs and other devices can be manually controlled.
−	Accountability	Low	Applications are maintained under support contracts.

Fig. 3. Editing assets


germaneriposte.org
CAIRIS
System
Requirements
Risk
UX
Models
Options
Search
STCS Application
<<exploits Wide Scope>>
Exploit Automated account lockout
<<exploits Automated account lockout>>
<<uses>>
<<Enumeration threatens>>
<<System enumeration threatens>>
Broken instrument alarm
Exploit System enumeration
Ben
Rick
germaneriposte.org:7071/api/attackers/name/Ben

Environment: Day
Task: Broken instrument alarm
Misuse Case: All

Attacker properties

Name

Ben

Description

Ben is a security engineer for RedTeam LLC. He is working with a junior tester (Alex) to conduct a red team test on ACME Water; they are evaluating ACME's security posture to ensure their security policies on te use of IT are being followed.

Ben has a first degree in Computer Science, and Masters degree in Information Security. Ben is also holds CREST accreditations.

Role
Cracker

Motivation
Data theft
Data destruction
Data modification

Capability	Value
Resources/Funding	Low
Resources/Equipment	Medium
Software	Medium
Technology	High

Fig. 4. Visualising a task's security impact

Bob devised a solution for mitigating this risk. When details of this risk mitigation were added to CAIRIS, the goal model was automatically updated to indicate the leaf obstacle was resolved, as illustrated in Figure 2 (right).

C. Contextualising the Implication of Security and Usability Design Decisions

While creating and analysing personas, Alice and her team created two environments in CAIRIS based on the two main contexts of use associated with the new control system: these relate to operations during normal working hours (Day), and out-of-hours operations (Night). Bob used CAIRIS to elicit assets of value to the water company associated with the new control system (Figure 3). Because threats and vulnerabilities might differ based on these contexts, he indicated the security properties needing to be protected in each context. Bob used this information to help elicit risks.

In parallel to this, Alice elicited tasks that the personas created would carry out. For example, she used CAIRIS to provide details of how one of the personas deals with a broken instrument alarm; this included a narrative scenario contextualising how the persona carries out the task, and a list of assets associated with the task.

When Alice visualised the *Broken Instrument Alarm* task in context (Figure 4), she noted risks (red ellipses) that may impact the task (the blue ellipse). By clicking on nodes in the model, Alice obtained details of these risks, including details of potential attackers. This allowed her to review the rationale behind the models created by Bob.

Together Bob and Alice sat down to model data flows between elements associated with this task (Figure 5). The task description guided Alice in sketching the DFD, while Bob

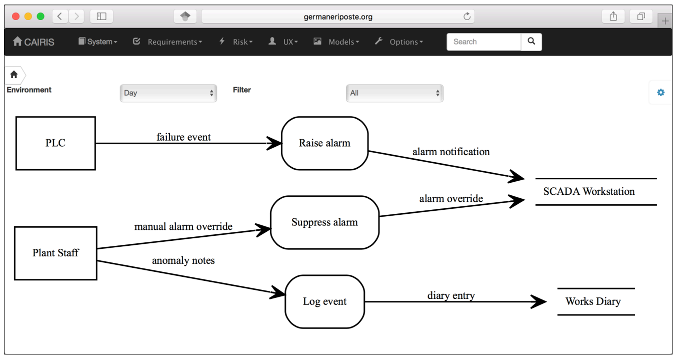


Fig. 5. DFD illustrating the data flow associated with fixing a broken instrument.

contributed by clarifying the information carried in each data flow, and checking that the entities/data stores and processes corresponded with assets and use cases in the existing design model. Once complete, Bob identified obstacles associated with the obstruction of security properties in each DFD element, to form the basis of additional attack tree modelling by his team.

V. Related Tools

CAIRIS is unique in its support for modelling from a security, usability, and software engineering perspective. There are, however, several other tools that support modelling within each area.

A range of commercial tools support the elicitation and management of risk, but comparatively few support the modelling of threats that feed into a risk analysis. Of the tools which are available, the two most notable are the CORAS tool [27] and IriusRisk [28]. CORAS is an open source tool supporting the creation of risk related models associated with the CORAS method; these models support participative risk analysis workshops. IriusRisk is a commercial SaaS platform for capturing and visualising the impact of threats on software architectures.

Several tools allow designers to draw data flow diagrams to support threat modelling, e.g. [29][30]. Given the importance of threat modelling, future work will explore concept alignment between CAIRIS and such tools, and prototype import/export tools to evaluate the potential of threat modelling tool interoperability.

While there are various tools for supporting task analysis and modelling, tool support for personas is currently limited to SaaS tools or templates the facilitate their specification rather than elicitation, e.g. [31].

The dominant software engineering tool for supporting KAOS goal and obstacle modelling is Objectiver [32]. This is a commercial desktop product, but is currently being extended to support web services [33].

CAIRIS was not designed to compete with tools like those mentioned above. It does, however, illustrate how complementary "best of breed" tools might be used together, and the form that process and data interoperability might take.

VI. Conclusion

This paper presented CAIRIS as a vehicle for collaboration between usability and security engineers. We presented the main features of CAIRIS, before presenting usage scenarios that illustrate how security and usability engineers might use the tool. In doing so, we also show how CAIRIS models afford opportunities for tool-supported automation, where processes for managing and provisioning of machine-readable early-stage design models – *Design as Code* – can lead to collaboration between usability and security engineers.

The usage scenarios show how CAIRIS facilitates typical security and usability engineering activities, and helps interaction designers understand the design rationale of security engineers, and vice versa. Implicit in these scenarios is the need for precision. Alice and Bob need to be precise when directly and indirectly specifying security model elements in CAIRIS, e.g. the rationale behind asset security properties, and the information carried in each data flow. Such precision is needed to facilitate traceability, and automation of the analysis and generation of CAIRIS models. Such precision should be welcomed rather than considered a distraction, given the potential of ambiguity for undermining requirements and other design models. We accept that ambiguity can be a resource for design [34], but the scenarios also show that precision need not interfere with ideation when designers from different backgrounds work together.

Although explicitly designed to support the IRIS framework [35], the usage scenarios suggest CAIRIS is evolving to become methodology agnostic. For example, the activities carried out by Bob in Section IV-C are not incompatible with STRIDE [22], or even LINDDUN [23] if the goals pertained to privacy rather than security. Moreover, CAIRIS was recently used by students to help complete coursework assignments for a unit on 'Security by Design' at Bournemouth University. The assignment brief did not mandate the use of any particular methodology, but CAIRIS was found to support students using SQUARE [36], and elements of STRIDE. A more detailed evaluation of the use of CAIRIS to support different Security & Privacy Requirements Engineering approaches will be the subject of future work.

References

[1] H. A. Simon, "Rational decision making in business organizations," *The American Economic Review*, vol. 69, no. 4, pp. 493–513, sep 1979.

[2] S. Faily and I. Fléchais, "A Meta-Model for Usable Secure Requirements Engineering," in *Proceedings of the 6th International Workshop on Software Engineering for Secure Systems*. IEEE Computer Society, 2010, pp. 126–135.

[3] R. Darimont and C. Ponsard, "Supporting quantitative assessment of requirements in goal orientation," in *Proceedings of the 23rd IEEE International Requirements Engineering Conference*, 2015, pp. 290–291.

[4] S. Faily and I. Fléchais, "Towards tool-support for Usable Secure Requirements Engineering with CAIRIS," *International Journal of Secure Software Engineering*, vol. 1, no. 3, pp. 56–70, July-September 2010.

[5] S. Faily, J. Lyle, I. Fléchais, and A. Simpson, "Usability and Security by Design: A Case Study in Research and Development," in *Proceedings of the NDSS Workshop on Usable Security*. Internet Society, 2015.

[6] G. Kim, K. Behr, and G. Spafford, *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*. IT Revolution Press, 2014.

[7] B. Wilder, *Cloud Architecture Patterns*. O'Reilly, 2012.

[8] S. Faily, "CAIRIS GitHub site," http://github.com/failys/CAIRIS, July 2017.

[9] A. Ronacher, "Flask website," http://flask.pocoo.org, December 2016.

[10] M. Wright, "Flask-Security website," https://pythonhosted.org/Flask-Security, June 2017.

[11] G. Dumpleton, "mod_wsgi-express PyPI site," https://pypi.python.org/pypi/mod_wsgi, March 2017.

[12] webinos Consortium, "webinos design data repository," https://github.com/webinos/webinos-design-data, March 2013.

[13] AT&T, "Graphviz web site," http://www.graphviz.org, June 2012.

[14] DocBook Technical Committee, "DocBook Schema Specification," http://www.docbook.org/schemas/5x, 2008.

[15] J. Robertson and S. Robertson, "Volere Requirements Specification Template: Edition 14 - January 2009," http://www.volere.co.uk/template.htm, 2009.

[16] S. Faily and I. Fléchais, "Barry is not the weakest link: eliciting secure system requirements with personas," in *Proceedings of the 24th BCS Interaction Specialist Group Conference*. British Computer Society, 2010, pp. 124–132.

[17] S. Faily and I. Fléchais, "User-centered information security policy development in a post-stuxnet world," in *Proceedings of the 6th International Conference on Availability, Reliability and Security*, 2011, pp. 716–721.

[18] S. Faily, "Creating Personas using the Persona Helper, CAIRIS, and Trello," http://dx.doi.org/10.13140/RG.2.2.29180.54403, March 2017.

[19] ——, "Persona Helper Chrome extension," https://github.com/failys/persona_helper, April 2017.

[20] S. Faily and I. Fléchais, "The secret lives of assumptions: Developing and refining assumption personas for secure system design," in *Proceedings of the 3rd Conference on Human-Centered Software Engineering*, vol. LNCS 6409. Springer, 2010, pp. 111–118.

[21] Fog Creek Software, "Trello website," https://trello.com, April 2017.

[22] A. Shostack, *Threat Modeling: Designing for Security*. John Wiley & Sons, 2014.

[23] K. Wuyts, "Privacy Threats in Software Architecture," Ph.D. dissertation, KU Leuven, 2015.

[24] B. Schneier, "Attack Trees," *Dr. Dobb's Journal*, 1999.

[25] A. van Lamsweerde, *Requirements Engineering: from system goals to UML models to software specifications*. John Wiley & Sons, 2009.

[26] E. R. Gansner, "The DOT Language," http://www.graphviz.org/doc/Dot.ref, February 2002.

[27] M. S. Lund, B. Solhaug, and K. Stølen, *Model-Driven Risk Analysis: The CORAS Approach*. Springer, 2010.

[28] Continuum Security, "IriusRisk - Threat Modeling Tool," https://www.continuumsecurity.net/threat-modeling-tool/, April 2017.

[29] Microsoft Corporation, "Microsoft Threat Modeling Tool 2016," https://www.microsoft.com/en-us/download/details.aspx?id=49168, October 2015.

[30] Mozilla, "SeaSponge website," http://mozilla.github.io/seasponge, April 2015.

[31] Fake Crow, "User Persona Creator," https://xtensio.com/user-persona/, March 2017.

[32] Respect-IT, "Objectiver web page," http://www.objectiver.com, June 2013.

[33] R. Darimont, W. Zhao, C. Ponsard, and A. Michot, "A Modular Requirements Engineering Framework for Web-based Toolchain Integration," in *Proceedings of the 24th IEEE Requirements Engineering Conference*, 2016.

[34] W. W. Gaver, J. Beaver, and S. Benford, "Ambiguity as a resource for design," in *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2003, pp. 233–240.

[35] S. Faily, "A framework for usable and secure system design," Ph.D. dissertation, University of Oxford, 2011.

[36] N. R. Mead, E. D. Hough, and T. R. S. II, "Security Quality Requirements Engineering (SQUARE) Methodology," Carnegie Mellon Software Engineering Institute, Tech. Rep. CMU/SEI-2005-TR-009, 2005.