Mission Assurance Proof-of-Concept:

Mapping Dependencies among Cyber Assets, Missions, and Users

Laurin Buchanan, Mark Larkin, Anita D'Amico
Secure Decisions Division
Applied Visions, Inc.
Northport, NY 11768 USA
{laurin.buchanan, mark.larkin, anita.damico}@securedecisions.com

Abstract—**Decision makers must know if their cyber assets are ready to execute critical missions and business processes. Network operators need to know who relies on a failed network asset (e.g. IP address, network service, application) and what critical operations are impacted. This requires a mapping between network assets and the critical operations that depend on them, currently a manual and tedious task. In addition, because of the dynamic nature of networks and missions, manual mappings of network assets to operational missions rapidly become outdated. This paper describes one approach to modeling the complex relationships between cyber assets and the missions and users that depend on them, using an ontology developed in conjunction with practitioners and cyber mission assurance researchers. We describe the "Camus" (cyber assets, missions and users) proof of concept, which uses this ontology and automatically populates that model from data already on the network. We discuss the technical approach and provide examples of query results returned by the model. We conclude by describing ongoing work to enhance this proof of concept and its potential applicability to support mission assurance and mission impact solutions.**

Keywords-business continuity; computer security; cyber mission assurance; data mining; network defense; network management; ontology

I. Introduction

Understanding the operational environment is a critical element of mission assurance, but how does an organization identify what cyber infrastructure is truly critical to their business operation or mission? What assets should receive those limited dollars for security investment? When faced with responding to a network intrusion alert or an event affecting network service, an analyst must prioritize the importance of the potential degradation, but who relies on the attacked/failed cyber asset? What critical operation/mission is impacted by the asset's loss? What hidden or downstream assets or capabilities depend on an unavailable or vulnerable network resource?

To answer these questions, organizations need to model complex relationships between network assets and critical operations, and to automatically populate and regularly refresh that model to reflect actual usage of assets. Situation awareness systems cannot determine the criticality of a cyber asset without information about the mission and task dependencies of the asset, other assets or network services dependent on that asset, and the people and user dependencies of the cyber asset. Automated methods exist to simply map network assets, but they do not automatically incorporate contextual information such as what critical organizational missions an asset supports. There are no systematic methods for mapping these relationships; existing manual methods for mapping organizational missions to cyber assets are tedious, labor-intensive, and therefore updated infrequently. Business continuity planning systems contain the intended use of cyber assets, but do not reflect actual use of those assets to support organizational missions.

The goal of this project was to model the complex relationships between cyber assets and the missions and users that depend on them, and implement a system for automatically populating that model from commonly available network data sources. The model forms the foundation of what we call Camus (for **c**yber **a**ssets, **m**issions and **u**sers), with which users can query their own populated model to determine:

- What users and operations are affected by the failure of a specific asset?
- What network assets (e.g. devices, IP addresses, applications, and data) are needed to perform specific mission tasks or processes?
- Does an alternative capability or resource already exist that can be substituted for the lost asset?
- What other assets depend on this specific asset?

Much of the grounding for Camus came from Salerno's Air Force Situation Awareness Model (AFSAM) [1],[2], which describes the path *data* takes to become *information* that can be consumed by analysts for improved situation management. Camus aligns with the portion of AFSAM labeled "knowledge of us," which provides contextual information about the operational environment. The general AFSAM was refined [3] and applied directly to the cyber domain, resulting in the Cyber Situation Awareness (SA) Model. Within the Cyber SA Model, the "knowledge of us" relevant to mission assurance is an accurate understanding of how operations are impacted by degradations and compromises in cyber infrastructure.

II. Technical Approach

The technical approach for Camus centers on the following three foundations:

- Automatically mine existing data to create contextual information to better understand attacks;

This work was supported by the Office of the Secretary of Defense and Air Force Research Laboratory through a Small Business Innovative Research grant, under contract no. FA8750-08-C-0166.

- Structure contextual information to show relationships between cyber assets, missions, users, and cyber capabilities;
- Provide this contextual information to automated systems and human analysts.

For Camus, we defined an ontology based on the concepts of User, Mission, Capability, and Asset. The ontology uses Resources (entities) and Properties (relationships) to build a model of the domain. The Resources define specific concepts in the domain, while the Properties define the relationships between the Resources.

At the center of Camus is the Application Core subsystem, which serves as the central subsystem to the Camus system and provides user and system interfaces, and communicates with the Semantic Repository to store and retrieve data. The Semantic Repository, which instantiates the ontology, stores the Camus data and provides the Semantic Rule Engine. The Data Import Service (DIS) provides importation of enterprise data, inference rule processing, and semantic repository population.

The Camus architecture was designed to demonstrate a proof-of-concept of the solution, rather than an operational prototype. To limit the cost and development time, we used open source technologies for the semantic repository, and a Wicket Web application front end. We developed a basic engineering interface to verify what relationships Camus discovered in the data. A future operational prototype would require more costly technologies and software development to add robustness, scalability and enhanced usability.

III. Ontology modeling mission-user-asset relationships

Camus relies on an ontology-based semantic approach to data integration and fusion, similar to the concepts discussed in Yoakum-Stover and Malyuta [4]. Our approach to developing the Camus ontology was informed by two workshops we held on mapping relationships between cyber assets, missions and users [5]. Attendees included people with operational responsibility to assure the availability of cyber assets for critical missions, researchers in areas related to the mapping of cyber assets to missions, and developers of technology that can be used in this mapping.

The workshops revealed the concept of "capability" in mapping missions and tasks to cyber assets, which provides a way to link these three groups of resources. Assets such as applications, data, and network services all provide capabilities (communication, printing, internet services, etc.) to the users. In turn, these capabilities support the completion of various tasks for missions: personnel may use a *communication* capability, such as email, to perform tasks in support of a mission. This idea of "capability" is also important to mission assurance: if a specific device that provides a communication capability, such as an email server, is not available due to an attack, another communication capability provided by a *different* cyber asset may be substituted with minimal mission impact. Mission planners and continuity planners often seek to optimize their plans in terms of these general capabilities, rather than in terms of individual, specific assets.

A. *Camus Ontology*

For Camus, we defined an ontology based on the four main types of resources identified during our workshops: User, Mission, Capability, and Asset. Each type represents a class of Resources. The ontology uses Resources (entities) and Properties (relationships) to build a model of the domain. The Resources define specific concepts (nouns) in the domain, while the Properties define relationships between the Resources.

The Foundation Ontology is the base level ontology for Camus and defines the core Resources and Properties to be used either directly or derived when using the Camus framework. Resources are associated with one of the four base types described above. The Foundation Resource and Property types are shown in Table I and Table II, respectively. An Extension Ontology can extend the Foundation Ontology for a specific implementation.

When mapping dependency relationships for mission assurance, not all relationships are equal. Some relationships are casual, others are critical. The criticality of a dependency can change over time, based on many factors. In the Foundation Ontology Properties, we created three Properties to express the criticality of a relationship: Uses, dependsOn and Requires, in increasing level of dependency.

TABLE I. FOUNDATION ONTOLOGY: RESOURCES

Resource	Type	Description
OrganizationalUnit	User	A collection of User related resources. An OrganizationalUnit can contain other OrganizationalUnits
Person	User	A single human resource
Account	User	A single identity on a cyber resource
MissionElement	Mission	A single tasking element
CapabilityInstance	Capability	A single instance of the ability to execute a specific action
CapabilityType	Capability	A classification of abilities to perform an action
CyberAsset	Asset	A non-human resource accessible from the network
Hardware	Asset	A physical computing device, element of a computing device, or peripheral of a computing device
Software	Asset	A program that performs a specific function directly for a user or system
Data	Asset	Distinct pieces of digital information that have been formatted a specific way
HostName	Asset	A label assigned to a computing device on a network
IPAddress	Asset	An Internet Protocol address
ConnectionPoint	Asset	A pairing of a specific IP address and Port for the purposes of communication
Port	Asset	A port number associated with a communication endpoint used by the Internet Protocol suite
Service	Asset	A software capability or process typically associated with a Port

TABLE II. FOUNDATION ONTOLOGY: PROPERTIES

Property	Relates
HasSubOrganizationalUnit	OrganizationalUnit->OrganizationalUnit
isSubOrganizationalUnitOf	OrganizationalUnit->OrganizationalUnit
hasMember	OrganizationalUnit->Person
isMemberOf	Person-> OrganizationalUnit
has Account	Person->Account
accountBelongsTo	Account->Person
hasSubTask	MissionElement->MissionElement
isSubTaskOf	MissionElement->MissionElement
precedesTask	MissionElement->MissionElement
postcedesTask	MissionElement->MissionElement
performsTask	Person->MissionElement
isPerformedBy	MissionElement->Person
taskUsesCapability	MissionElement->CapabilityInstance
capabilityUsedByTask	CapabilityInstance->MissionElement
accountUsesCapability	Account-> CapabilityInstance
capabilityUsedByAccount	CapabilityInstance->Account
usesCyberAsset	CapabilityInstance->CyberAsset
cyberAssetUsedBy	CyberAsset-> CapabilityInstance
hostsIPAddress	Hardware->IPAddress
ipaddressHostedBy	IPAddress->Hardware
hostsSoftware	Hardware->Software
softwareHostedBy	Software->Hardware
hostsData	Hardware->Data
dataHostedBy	Data->Hardware
managesData	Software->Data
dataManagedBy	Data->Software
mapsToIPAddress	HostName->IPAddress
mapsToHostName	IPAddress->HostName
usedByConnectionPoint	IPAddress->ConnectionPoint
usesIPAddress	ConnectionPoint->IPAddress
servesConnectionpoint	Hardware->ConnectionPoint
servedByHardware	ConnectionPoint->Hardware
servesPort	ConnectionPoint->Port
servedByConnectionPoint	Port->ConnectionPoint
mapsToService	Port->Service
mapsToPort	Service->Port
supportsCapabilityType	Service->CapabilityType
supportedByService	CapabilityType->Service
hasCapabilityType	CapabilityInstance->CapabilityType
capabilityTypeOf	CapabilityType-> CapabilityInstance
is a	Various – shows is type of
Uses	Various – shows minimal dependency
dependsOn	Various – show moderate dependency
Requires	Various – shows maximum dependency

IV. TECHNOLOGIES USED

Camus was designed using a Web-based architecture using open source components. It was developed on a Windows platform using the Eclipse version 3.4 (Ganymede) Integrated Development Environment (IDE) and Java Development Kit (JDK) 1.6.0_18. The Camus application has been run on the Windows 7, Windows Vista, and Ubuntu version 9.10 platforms. Camus consists of three main subsystems: Semantic Repository, Application Core, and Data Import Service. Each of these subsystems can be hosted on separate platforms; they communicate using HTTP.

A. *Semantic Repository*

The semantic repository is implemented using Sesame and hosted on an Apache Tomcat servlet container. Sesame is a framework for storage, inferencing, and querying Resource Description Framework (RDF) data. It supports RDF, RDF Schema (RDFS), and OWL Web Ontology Language, which are specifications developed by the World Wide Web Consortium (W3C) that define the central concepts used for semantic web applications. OWLIM is an enhancement to Sesame specifically for supporting RDFS and OWL and improving query and reasoning performance. We used the free SwiftOWLIM version (now known as OWLIM-Lite from Onotext) in the final Camus proof-of-concept.

1) Semantic Web

The basic data construct in the semantic model is the *statement*. A statement defines a relationship between two resources. A statement is made up of three elements:

Subject-Predicate-Object

The Subject and Object elements are both Resources (entities), and the Predicate defines a relationship between these Resources. A statement is also called a triple indicating the three elements that make up the statement. A collection of statements represents the data model. All three elements are represented as URI's (Universal Resource Identifier).

Statements are added to a repository either by assertion or inferencing. Asserted statements are those statements added to a repository by a user or external application. Inferred statements are those generated by an inferencing engine or reasoner and added to a repository. A Reasoner (or Inferencing Engine) examines all the statements in the repository, and using the rules defined by the various specification levels, creates new statements which are added to the repository. The act of deriving new statements is called inferring and the resulting statements are inferred statements. These are also used by the reasoner to infer even more statements.

B. *Application Core*

The Camus Application Core subsystem provides the central control for the Camus application. The Application Core subsystem is hosted by Apache Tomcat, an open source servlet container. There are three main components of the application core: Camus App core, the RESTful API, and a Wicket Web application front end.

1) Camus App Core

The App Core provides an API to interact with the Camus repository. All data items in the repository are stored as RDF statements (Subject-Predicate-Object) while methods support ensuring, getting, and removing these statements. Ensuring a statement will add the statement to the repository if it does not already exist, and update it otherwise. The CamusRepository class provides methods to execute queries against the repository. Both SPARQL and SeRQL query syntaxes are supported.

Attributes provide the ability to define additional data about Resources or Properties. Attributes for Resource and Properties are implemented via *reification*, which provides the ability to make statements about other statements. Typically, a statement is of the format Subject-Predicate-Object, where the Subjects and Objects are simple Resources; however, with reification the Subject and Object may be other statements. This provides the ability to define attributes for Resources and Properties, such as the *CreatedDateTime* attribute which indicates when a Resource was first discovered by the data import process and inserted into the ontology. Reification is very resource intensive to implement, because it increases the overall statement count in the repository: the first attribute of a Resource or Property has an overhead of five additional statements. Each additional Attribute for that particular Resource or Property is one additional statement. Additional attributes that would indicate confidence values for relationships mapped by Camus were not implemented due to this resource consumption.

2) RESTful API

We implemented a RESTful interface to provide the ability to interact with the Camus repository. A RESTful interface uses the Representational State Transfer (REST) model for communications. It is used by the Data Import Service to populate the Camus repository with Resources, Properties, and Attributes. The RESTful interface is implemented using the open source RESTlet Framework.

C. *Data Import Service*

The Data Import Service (DIS) provides importation of enterprise data, inference rule processing, and semantic repository population. These capabilities were implemented in a separate process in order to maximize scalability and provide flexible deployment scenarios. The DIS is implemented in Java using Java SE 6, enabling access to the widest available set of open source and commercial software products for integration, and the largest possible set of deployment platforms. By leveraging Java-compatible scripting engines for execution of custom logic, both importation logic and the inference rules used by the DIS can be modified without requiring a Java development environment or a formal software development cycle.

1) High-level Workflows

The DIS is deployed as a Java application, with an accompanying data processing pipeline configuration file, data import scripts, and rule statements. The DIS process can be visualized as a recurring pipeline with concurrent actions, designed to process one dataset to completion, populate the semantic repository with the results, process the next dataset, and so on. The pipeline is an XML file that describes a tree of work times and the dependencies between them, and represents a tree of dependent work items. Each work item describes the sources of data it consumes, the scripts that will be applied to these sources, and the rule set that will be applied to the facts generated by the import scripts. In addition, each work item describes which other work items' outputs it needs to function, thus defining a dependency tree. Each work item performs a single analysis data import and/or inference rule execution (e.g. correlate DHCP log with NetFlow log).

2) Types of Data Processed

We identified two broad categories of enterprise data: Reference Data and Event Data. We categorize anything that is "asserted" (a statement of fact) as "Reference Data." These data usually describe enterprise resources such networks, servers, domain and host name information, as well as organizational information like personnel, titles, teams, groups and departments, as detailed in org charts or directory listings. Reference Data are used by the DIS to build a list of entities (with related descriptive information) that is then published to the semantic repository with no inference rules applied.

To infer the relationships between entities, the DIS consumes activity or usage data from enterprise systems. We categorize this kind of information as "Event Data." Examples of this kind of data are NetFlow, server and application logs, and DHCP Logs. To process Event Data, it is first imported and normalized, then an appropriate set of inference rules are applied to generate relationship data between observed entities.

3) Import Engine

The first step in a data import pipeline is importation of raw data by running import scripts (written in JavaScript) to generate a set of normalized "facts", the in-memory Java objects that the DIS conveys through the pipeline. The DIS leverages the JavaScript engine that ships as part of Java 6 (via JSR 223). This script engine is derived from the "Rhino Engine." Once fact objects have been created, the DIS submits these to the rule engine and invokes the inference rules which produce a final set of objects that represent Resources and Properties compatible with the Camus semantic repository. The DIS uses the JBoss Rules Engine (codenamed "Drools"), an open source product licensed under the Apache License v2.0. JBoss Rules is a Forward-Chaining Rule Engine which starts with a set of facts that are asserted into the engine's working memory, and iteratively applies if-then statements until all the facts have been processed. If-then statements may create additional facts that are in turn asserted back into the rule engine's working memory, possibly causing more conditions to trigger. This process can provide a very sophisticated and complex inference processing system. The performance of JBoss Rules was less than ideal, however; other rules engines should be evaluated to see if rule inference performance can be improved.

4) Scalability

Camus was designed for deployment to an enterprise, although no quantitative measures of scalability are currently available. The data import process, from raw input to semantic repository, is "forward-only", meaning the DIS never reads from the Semantic Repository. Only inserts and updates are performed on the repository, which reduces the complexity of the overall system and improves the ability to distribute data importation workload across many computing resources without the semantic repository itself becoming a bottleneck. By breaking down importation and inference workloads into a number of independent processes, rapid turnaround of enterprise data can be achieved. Data processing can be distributed across different servers, reducing the impact on network performance. This also alleviates the need to move sensitive information across large parts of the enterprise network by keeping raw data local to its source; only the output data from a data processing pipeline must be sent to the Semantic Repository. Multiple DIS instances installed in an enterprise will contribute results to a single Semantic Repository, providing a single view of the enterprise.

The DIS also implements a lightweight historical cache, which maintains historical aggregations of activity and greatly improves the performance of rule inferencing by allowing the rule engine to work across event aggregations, rather than each individual event; e.g. the count of packets that are sent from a workstation to a server is stored, rather than the individual NetFlow records. This also greatly reduces the data storage and processing requirements. Aggregation, however, means that changes to rules that affect what information is needed, or how aggregations are calculated, will require time to build up sufficient aggregate periods for the new rules to detect meaningful relationships, and that Camus cannot be used for historical behavioral analysis.

D. *Synthetic Data Set*

We were unable to obtain real-world sample data that included network activity in support of an identified mission; most available security data sets contained attack data and random activity data, but lacked any identified missions or mission activity. In an early version of the Camus proof-of-concept [6] we used such a security data set, but without known ground truth of mission-related activity we were unable to verify the validity of relationships that Camus discovered and inferred.

To overcome this challenge in later versions of the proof-of-concept, we modeled business operations typical of a commercial organization and asserted our own mission structure, using typical back-office departments such as Finance, IT, etc. as generic missions, and using functional teams such as Accounts Receivable, Accounts Payable, Benefits, Payroll, etc. as representations of tasks. Leveraging team members' prior corporate IT and business continuity planning experiences, we then designed a network environment that would support the missions. We developed a synthetic test data set, proscribing various users, systems, and realistic log data and traffic flows by type, quantity, and periodicity; some specified activity supported various missions, other activity was more generic. The resulting synthetic network data (flow, logs, etc.) was used to test and demonstrate Camus.

To reflect this organization, we extended the User and Mission resource types in the Foundation Ontology to include resources such as Company, Department, Group and Role, resulting in the combined Foundation and Extended Ontology seen in Fig. 3. In this figure, the User type resources are shown in the upper left and consist of Organizational Unit, Person, and Account (Foundation elements) and the extended elements Company, Department, Group, and Role. The Mission resources in the upper right of the diagram consist of Mission Element (Foundation element) and extended elements Mission, Task and Subtask. Capabilities are represented by the Capability Instance and Capability Type elements. Lastly, the three main types of Asset resources are shown in the bottom half of the diagram: Hardware, Software and Data. Cyber Assets combine to provide Capabilities, which are used by Users to perform Missions.

V. Mapping Results

Readers familiar with entity relationship diagrams will grasp the ontology and relationships maps produced by Camus. In these graphs, nodes are the Resources while edges represent the Properties, as seen in the mapping of relationships between user (Case_Donaldson) and mission (Human_Resources) in Fig. 1. The user shown in the Person node has relationships to the Payroll Subtask and its parent Task, Benefits. The lines from Person to Task and Person to SubTask indicate that evidence for each relationship was discovered by Camus. Reference data provided to Camus asserted that the Benefits task is part of the Human Resources mission; this enabled Camus to infer the mapping of Person to Mission.

Fig. 2 shows the relationship of the capabilities Database, Internet_Services, and Cryptography to a Hardware resource, i.e. an Asset, with the designation 192.168.10.20. The data provided to Camus did not indicate whether a computing device was a workstation or server, so we used the reasoner to infer "what is a server?" and "what is a workstation?" based on whether the individual device had connections below port 1024, a behavior commonly associated with servers. In addition, MAC addresses of workstations do not change as often as IP addresses assigned via DHCP, so this enables Camus to more accurately map users to their workstation assets. Camus indicates a server as a Hardware resource with its IP address as its node name; workstations are identified by Media Access Control (MAC) addresses as the node name. In Fig. 2, the Hardware resource node name is an IP address, indicating Camus inferred that this asset is a server.

Camus answers mission assurance questions, from either top-down or bottom-up perspectives. The top-down approach is often related to mission readiness, e.g., when a senior officer in

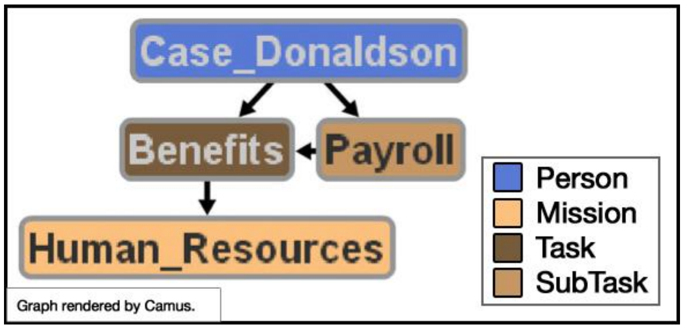

Figure 1. Camus mapping users to mission

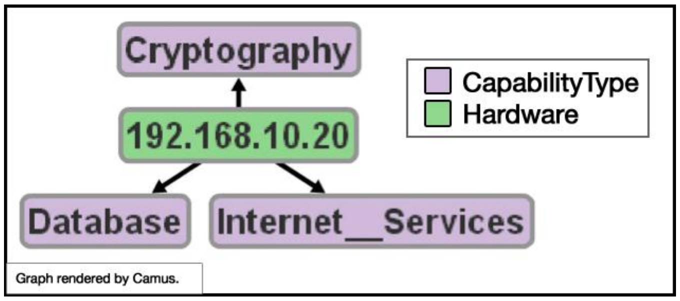

Figure 2. Camus mapping capabilities to assets

the military or industry asks, “What cyber assets do I need to execute my mission?” The question focuses at the top of the organization, at the mission level. Bottom-up questions, such as, “What missions or users are impacted by the loss of this device?” may be asked during planning stages prior to mission execution, or by incident responders seeking to determine how to ensure mission success after an incident occurs

Fig. 4 shows Camus answering the mission readiness question, “What is needed for the Invoicing SubTask?” by mapping relationships from the Invoicing SubTask to connection points (IP address and port number). Personnel associated with Invoicing are linked to their user accounts and workstations (shown by MAC addresses), then a dense matrix of connections to other assets from those accounts and workstations. The graph also reveals hidden dependencies from the Hardware asset 192.168.10.20 to the connection point of 192.168.10.21:3369.

Camus answers the question, “What missions does the IP address 102.168.10.30 support?” in Fig 5. This graph shows the selected IP address at the center, with initial mappings to Capability Instances, which are then mapped to SubTasks (Invoicing), then Tasks (Accounts_Receivable, Credit, Payroll, Accounts_Payable, Recruiting, Human_Resources, Benefits), and Missions (Human Resources and Finance).

VI. Conclusions

Technologies for mapping cyber assets to missions and users provide an accurate operational picture of an organization: who uses what asset, and what they are using it for. Technologies like Camus are one component in a situation awareness solution that supports mission assurance. The dependency mappings provided by Camus can provide a scientific basis for mission-based risk assessments, revealing which assets actually support critical or multiple missions, and are thereby prime targets for a limited cyber security budget. Camus could likewise enable mission-based vulnerability analysis and mitigation, or assessment of actual impacts of cyber attacks on missions [7]. For these approaches to be effectively realized, they all require mapping missions to cyber assets based on actual, not planned, usage. Camus could support business continuity planners and systems, as well as capacity planning systems, providing automatic updates of system dependency mappings and usage information based on recent, actual data.

There are several current operational applications for Camus. If integrated with security event and incident management systems such as ArcSight, or intrusion detection or prevention devices, Camus could provide context to help prioritize tickets based on affected missions, and help incident responders determine who to notify. Camus could be both a consumer of, and provider to, configuration management systems from vendors such as IBM (Tivoli), Novell (ZenWorks) and

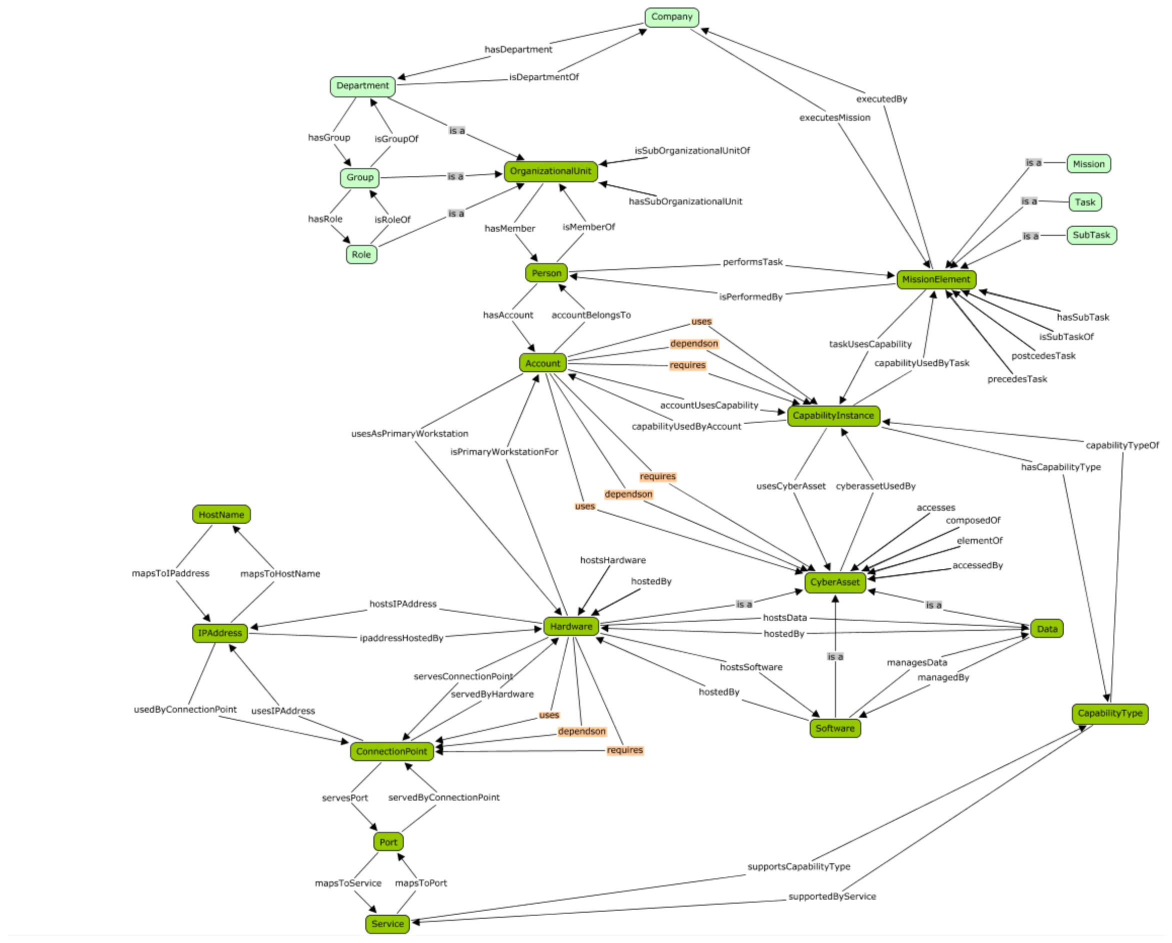

Figure 3. Foundation and Extension ontology used in the Camus prototype

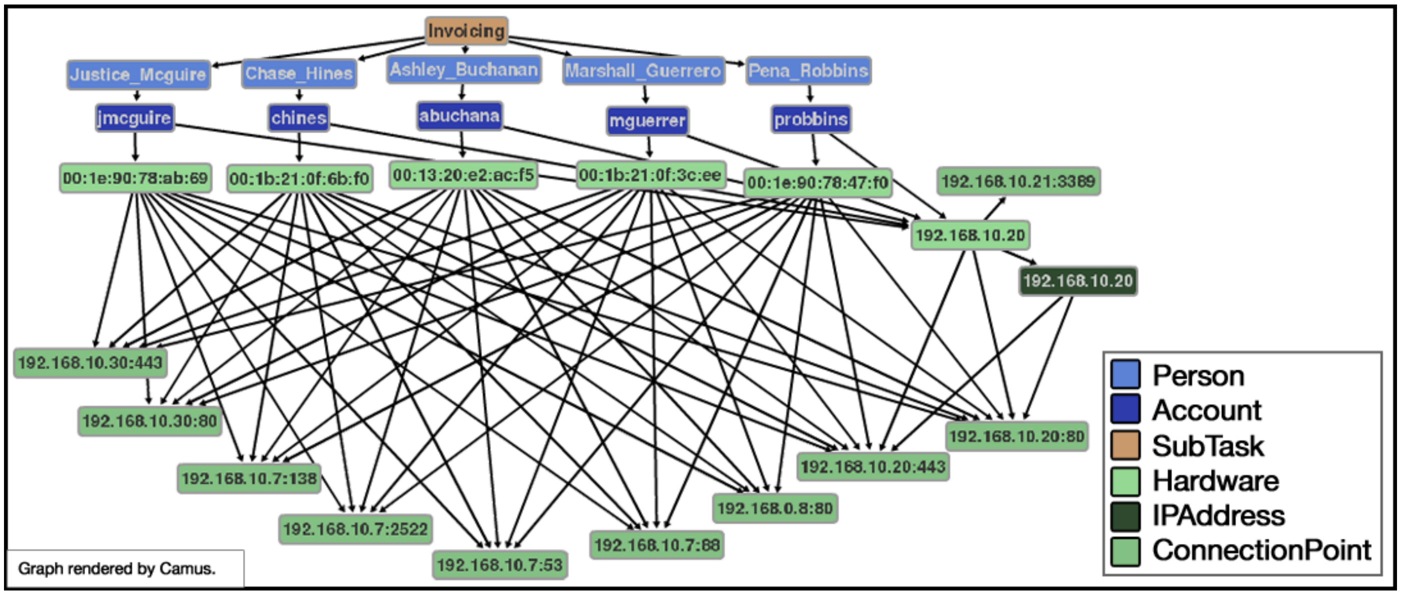

Figure 4. Camus mapping relationships between tasks and assets

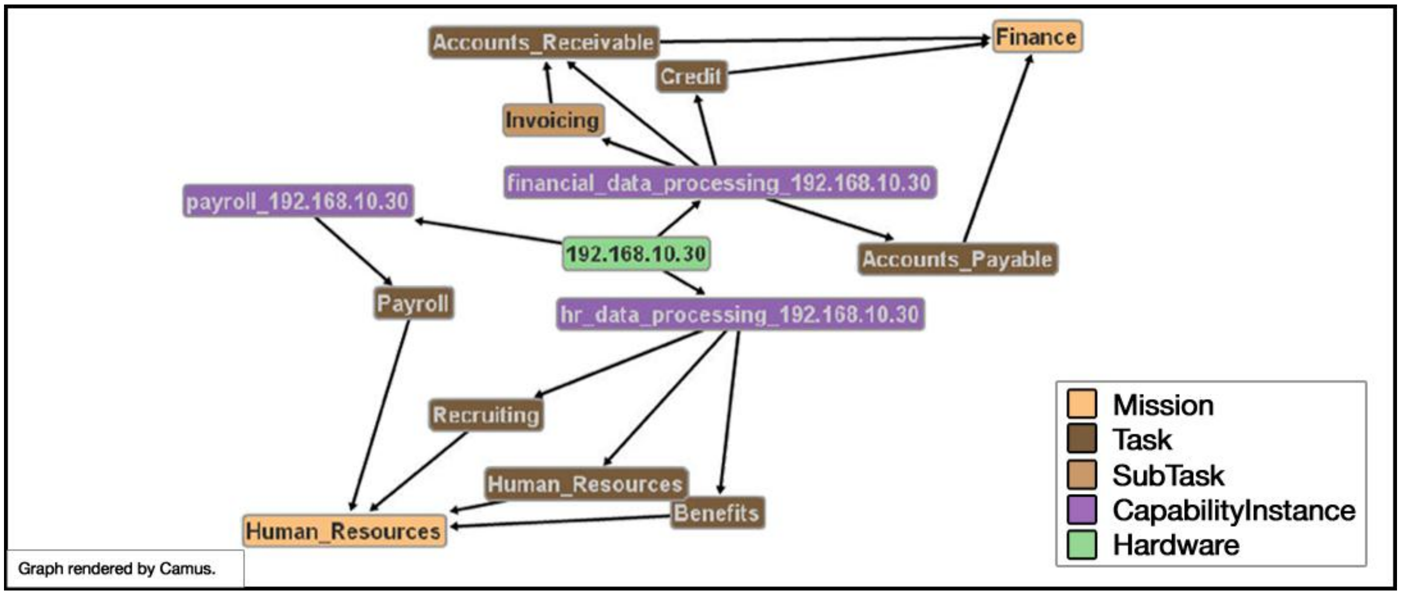

Figure 5. Camus mapping cyber asset to missions

BMC (Atrium), taking in asset inventory information and providing context for outage notifications and downstream infrastructure impacts. These technologies could benefit from the automated dependency mappings between cyber assets and missions that the Camus proof-of-concept has demonstrated.

References

[1] J. Salerno, M. Hinman, and D. Boulware, "A situation awareness model applied to multiple domains," Proc. of Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2005, vol. 5813, B. Dasarathy, Eds., pp.65-444.

[2] J. Salerno, "Measuring Situation Assessment Performance through the Activities of Interest Score," Proc. of 11th International Conf. on Information Fusion, IEEE Press, 2008.

[3] G. Tadda, J. Salerno, D. Boulware, M. Hinman, and S. Gorton, "Realizing Situation Awareness within a Cyber Environment", Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2006, vol. 6242, B. Dasarathy, Eds., Apr 2006.

[4] S. Yoakum-Stover and T. Malyuta, Unified Data Integration for Situation Management. Situation Management (SIMA), Proc.of the Military Communications Conf. (MILCOM), IEEE Press, 2008.

[5] A. D'Amico, L. Buchanan, J. Goodall, and P. Walczak, "Mission impact of cyber events: scenarios and ontology to express the relationships between cyber assets, missions and users," 1 December 2009, handle.dtic.mil/100.2/ADA517410.

[6] J. Goodall, A. D'Amico, and J. Kopylec, "Camus: automatically mapping cyber assets to missions and users." Proc. of the Military Communications Conf. (MILCOM), IEEE Press, 2009.

[7] M. R. Grimaila, L. W. Fortson, and J. L. Sutton, "Design Considerations for a Cyber Incident Mission Impact Assessment (CIMIA) Process," Proceedings of the 2009 International Conf. on Security and Management (SAM09), Las Vegas, Nevada, July 13-16, 2009.