JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

11100 Johns Hopkins Road • Laurel, Maryland 20723-6099

AOS-19-0107

March 2019

AUTOMATIC GENERATION OF NETWORK ELEMENT SOFTWARE (AGNES)

Final Report

Prepared for: Office of Naval Research

Prepared by:
Christopher Rouff, Ph.D.: Johns Hopkins University Applied Physics Laboratory
M. Douglas Williams, Ph.D.: Leidos, Inc. (currently at Seed Innovations)
Daniel Bennett: Shavano Systems, LLC

Task No.: FGXA9
Contract No.: N00014-15-1-2509

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01/10/2019	Final	01/01/2015-03/31/2019

4. TITLE AND SUBTITLE

Automatic Generation of Network Element Software (AGNES)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

N00014-15-1-2509

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Christopher Rouff, Ph.D.

5d. PROJECT NUMBER

FGXA9

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Johns Hopkins University
Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723-6099

8. PERFORMING ORGANIZATION REPORT NUMBER

AOS-19-0107

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
875 N. Randolph Street Suite 1425
Arlington VA 22203-1995

10. SPONSOR/MONITOR'S ACRONYM(S)

ONR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The goal of the Automatic Generation of Network Element Software (AGNES) project was to automatically generate network element software that is free from known weaknesses, reducing the cyber-attack surface area of military networks. Using today's development technologies and processes, it is effectively impossible for humans to write software that accounts for the hundreds of known weaknesses that can lead to vulnerabilities. Additionally, the encouraged practice of code reuse serves to perpetuate and propagate weaknesses; when vulnerability patches are applied to the original source code, they are unlikely to be propagated to reused code. AGNES generates weakness-free software by using a knowledge base of coding solutions for known weaknesses which will eliminate or drastically reduce known software vulnerabilities.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Christopher Rouff
U	U	U	UU	57	19b. TELEPHONE NUMBER *(Include area code)* 240-228-3603

Reset

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

CONTACT INFORMATION

Christopher Rouff, Ph.D.

Johns Hopkins University
Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723-6099

Phone: 240-228-3603
Christopher.Rouff@jhuapl.edu

CONTENTS

FIGURES

Page

TABLES

Page

EXECUTIVE SUMMARY

The goal of the Automatic Generation of Network Element Software (AGNES) project was to automatically generate network element software that is free from known weaknesses, reducing the cyber-attack surface area of military networks. Using today's development technologies and processes, it is effectively impossible for humans to write software that accounts for the hundreds of known weaknesses that can lead to vulnerabilities. Additionally, the encouraged practice of code reuse serves to perpetuate and propagate weaknesses; when vulnerability patches are applied to the original source code, they are unlikely to be propagated to reused code. AGNES generates weakness-free software by using a knowledge base of coding solutions for known weaknesses which will eliminate or drastically reduce known software vulnerabilities.

AGNES was funded as a grant from the Office of Naval Research (ONR) under contract N00014-15-1-2509/120937 with planned period of performance from July 2015 through March 2019. The AGNES team consisted of Johns Hopkins University Applied Physics Laboratory (JHU/APL) in Laurel Maryland, Leidos Corporation in Littleton Colorado, and Shavano Systems, LLC in Centennial Colorado. Due to funding constraints, only two of the three phases of the project were funded. During the two funded phases, the AGNES code generator successfully generated code for the Router Information Protocol (RIP), Version 2. The generated code successfully executed in a network simulation environment with other simulated routers. To fully validate the generated code, it was successfully tested in hardware and run in a standalone network with other routers. The code was run through rigorous static analysis that validated the absence of the weaknesses selected from the Common Weakness Enumeration (CWE) database. This validation demonstrated that weakness-free software can be automatically generated.

The focus of Phase 1 of the project was to develop a framework which would support demonstration of the ability to develop coding rules that would generate weakness-free code. The project selected known weaknesses related to network element software and developed the core framework, auto-generation engine, and coding rules for network elements. During Phase 2, the project completed the auto code generator framework which supported automatic generation of RIPv2 code. Though Phase 3 was not funded, residual funding from Phase 2 was used to complete the auto code generator and successfully generate RIPv2 code. This capability was demonstrated to the ONR program manager, Dr. Waleed Barnawi, in a network simulator in January 2018.

If follow-on work were to be pursued, JHU/APL recommends that AGNES be extended to include RIP authentication and cryptography (RFC 4822). This would require research into ontological representations for authentication and encryption specifications for code generation. Additionally, research into static analysis using code contracts would enhance the accuracy of static code analysis and further reduce software weaknesses and associated vulnerabilities. Addition of these capabilities would support code generation for secure protocols.

The AGNES project validated that it is possible to automatically generate network element software which is free from known weaknesses, thereby reducing the cyber-attack surface of networks. Though funding only supported demonstration of code generation for RIPv2, software for other network protocols could also be readily generated, thereby reducing software weaknesses. Further development of AGNES could lead to generation of secure software critical to military and civilian networks and other mission critical systems.

1 INTRODUCTION

Our nation's civilian and military information networks are under constant attack at a time when these networks are increasingly critical to our nation's infrastructure and in particular our military's command and control capabilities. Network element software / firmware is extremely vulnerable, both to direct attack and as the vector for attacks against other infrastructure elements. Mobile networks are particularly problematic: they have an over-the-air attack vector with no physical connectivity required; they require constant messaging to maintain the changing network topology; and many of the capabilities – especially in military systems – are embedded in firmware, which makes frequent updates more difficult. While millions of new malware attacks are being released each day, they rely on a much smaller number (thousands) of existing vulnerabilities in the software or firmware that is being attacked. These vulnerabilities in turn exist because human software engineers have written code that falls prey to just a few hundred common weaknesses in programming practice.

The goal of the Automatic Generation of Network Element Software (AGNES) project is to automatically generate network element software that is free from known weaknesses, reducing the cyber-attack surface area. Using today's development technologies and processes, it is effectively impossible for humans to write software that accounts for the hundreds of known weaknesses that can lead to vulnerabilities. Additionally, the encouraged practice of code reuse serves to perpetuate and propagate weaknesses; when vulnerability patches are applied to the original source code, they are unlikely to be propagated to reused code. AGNES generates weakness-free software by using a knowledge base of coding solutions for known weaknesses which will reduce software vulnerabilities.

AGNES is based on the following premises:

- Computers can generate software that is more secure than what is written by human programmers.
- A developer using a high-level protocol description language can generate more code in less time than a programmer using conventional programming tools.
- A developer using auto-generation techniques can generate security updates to existing code faster and with fewer errors than human programmers using conventional tools.
- Auto-generation enables tracing of the provenance of every module of code, preventing attacks that attempt to insert compromised versions of existing modules.

Figure 1 shows the overall goal of AGNES. Currently, engineers developing network element software have to read and interpret the network standard documents, called Requests for Comments (RFCs), to determine what code to write. In AGNES, these documents are stored in a machine-readable ontology along with coding rules based on the Common Weakness Enumeration (CWE) database maintained by the MITRE Corporation [1, 2]. Using the standards and coding rules in the ontology, AGNES generates code for network elements that is free of known software weaknesses and their associated vulnerabilities.

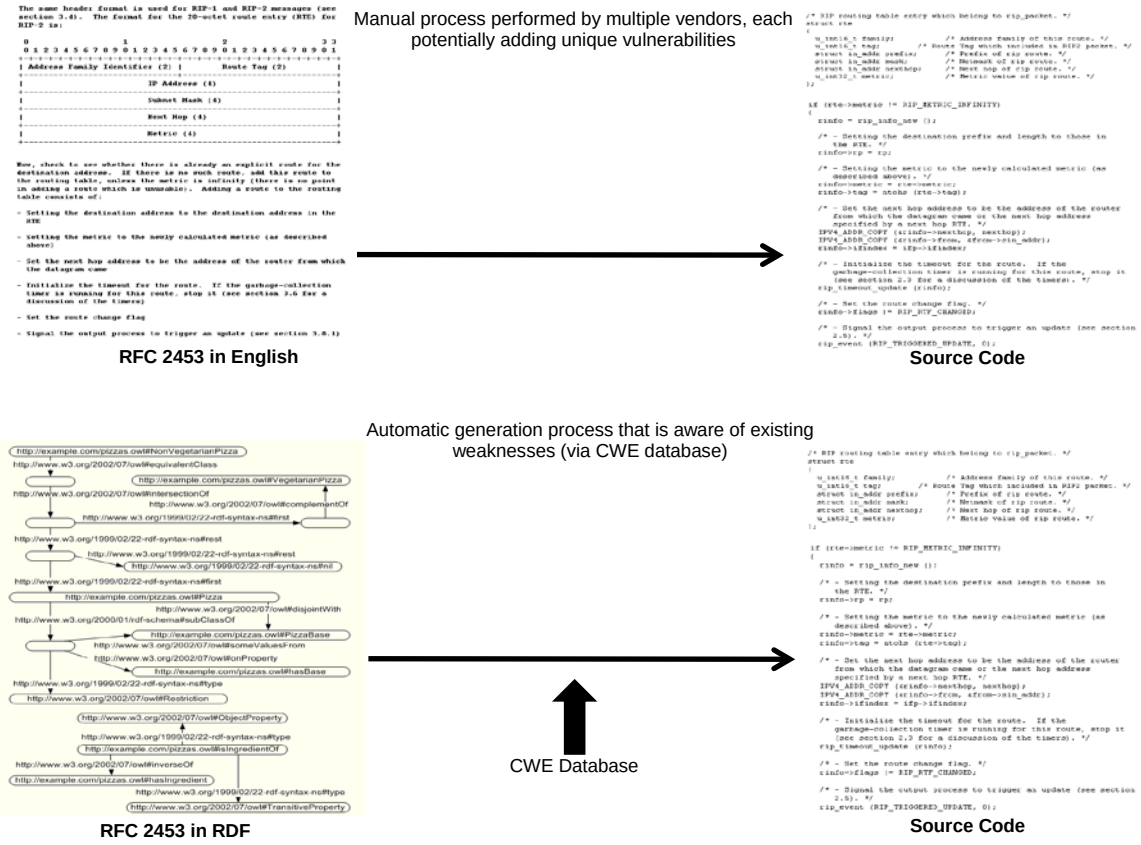

Figure 1: Manual versus AGNES automatic generation of network element software.

Since AGNES-generated code is free from known weaknesses, it will not provide openings for attacks by adversaries' malware. AGNES will also reduce the time required to develop network software modules and reduce the effort required to update software in response to new threats. The result is more secure software that is developed in less time and can be automatically updated to counter new threats.

2 AGNES OVERVIEW

The AGNES approach to building secure network element software is based on three fundamental capabilities:

1. High-level representation of building blocks of network protocols.
2. Machine-readable representations of rules for generating software that is free of known weaknesses.
3. Auto-generation of executable software.

To measure the utility and usability of AGNES, a quantitative evaluation process was defined to provide a head-to-head comparison of manually written open source network software against AGNES auto-generated software.

Our knowledge source of software weaknesses is the CWE database, an open database maintained by the MITRE Corporation [5] which is based on input from the larger software development community. There are approximately 1,000 weaknesses documented in the CWE. No human can keep in mind 1,000 constraints while writing software. AGNES builds secure software by storing a subset of these weaknesses and coding rules to generate network element software that is free from the vulnerabilities caused by these weaknesses.

Figure 2 shows a high-level overview of the AGNES concept of operations (CONOPS) and its main components. The main components include:

1. Software weaknesses that are formally specified in a machine-readable format and maintained in a knowledge base (ontology).
2. For each weakness in the ontology, an expert in secure programming develops a set of coding rules that will ensure the code avoids the corresponding weakness. For example, buffer overflow is a common weakness found in code; to prevent a buffer overflow attack, the code should perform bounds checking.[1] The coding rules are also specified in a machine-readable format and stored in the knowledge base.
3. Based on an ontology of network elements and software patterns, a developer specifies the design of network element software in a formal representation language.
4. The AGNES Auto-Code Generator (ACG) interprets the design representation and applies the coding rules to generate executable code.

The AGNES CONOPS starts with a coding expert writing the coding rules based on the CWE descriptions of coding errors that are found in network element software; those coding rules are stored in an ontology. A network expert also takes network element specifications and rewrites these so that they can be stored in an ontology. To develop new network elements, a developer specifies a network element by referencing the element components that are stored in the ontology by the network expert. The software for the new elements is generated based on the rules stated in the coding rules ontology. The resulting software is secure and free from known security weaknesses (as defined by the CWE).

The AGNES Network Element Ontology defines the concepts and relationships used to describe and represent network element components and software, such as classes and subclasses related to network element components and their characteristics, and relations and sub-relations among concepts and their constraints. Because of the level of formality required, AGNES uses the Resource Description Framework (RDF) to define the Common Network Element Ontology and store it in an Eclipse RDF4J database [53].

[1] Though a static code checker can find some buffer overflow errors, it may not find them all and so cannot guarantee that the code is free from such errors. Buffer overflow is just one example of the many weaknesses in the CWE, many of which are difficult to detect with static code checkers.

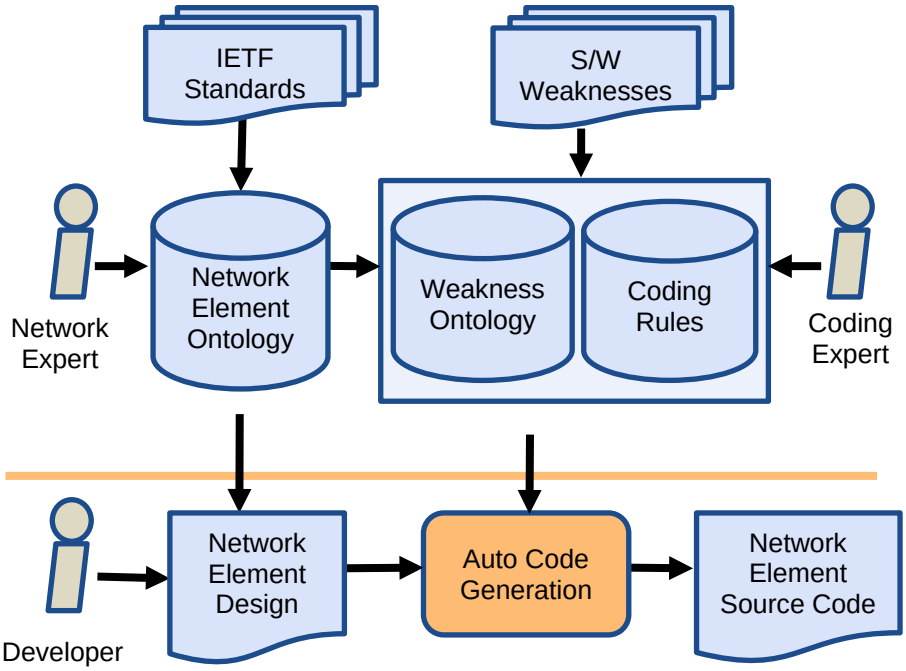

Figure 2: AGNES concept of operation.

The fundamental concept of the AGNES approach is that the representation of domain components (i.e., network element software/firmware) should be descriptive rather than imperative. This is the same fundamental idea that the Internet Engineering Task Force (IETF) uses in publishing Request for Comment (RFC) documents that describe a network component rather than just publishing source code that implements it. AGNES is using these RFCs as the source of the network element descriptions, encoding the English language description in RDF.

RIP [3] was used as a demonstration case for AGNES. RIP (currently RIPv2) is in widespread use in existing commercial and military networks, including wireless networks. A complete implementation of RIP, including cryptographic protocols and IPv6, requires understanding ten (10) separate RFCs. Automatic regeneration of this software whenever a new or updated RFC is released will significantly reduce the time to deployment and reduce the number of potential security vulnerabilities.

An RFC does not always completely describe the behavior required to implement a network component. We estimate, from experience working with RFCs, that they are typically only 80% specified.[2] For example, RIP does not specify how to store router tables or how to efficiently search them, it just describes the message formats and required behaviors of conformant implementations. This is because RIP is applicable to routers that range from small home routers to large Internet backbone routers and, while the functionality is the same, the details may be very different. On the other extreme are some cryptographic standards that are strictly mathematical algorithms and are completely specified. We did not change any RFC; instead, we

[2] Estimate is based on professional judgement and anecdotal evidence from years of experience.

produced ancillary descriptions that specified the required additional content, which is exactly what a vendor would have to do.

3 AGNES DESIGN AND IMPLEMENTATION

The following describes the design of the AGNES proof of concept in more detail.

3.1 Core Representation Framework

The AGNES infrastructure consisted of an Amazon Web Services (AWS) instance for project collaboration; Ububtu 14.04 LTS as its base operating system; GitLab for repository management, code reviews, issue tracking, activity feeds, and project wiki; Eclipse RDF4J (previously known as Sesame) for processing and handling RDF data; and other development tools.

The version numbers given for the software described below are the preliminary versions that were used. Over the course of the AGNES effort, later versions of the software were adopted.

3.1.1 Amazon Web Services (AWS) Instance for Collaboration

The AGNES project used an Amazon Elastic Compute Cloud (Amazon EC2) that provided convenient, secure, resizable compute capacity for the geographically distributed AGNES development team [54]:

> Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios.

The AWS EC2 instance hosted 64-bit Ubuntu 14.04 base operating system (see Section 3.1.2) and GitLab (see Section 3.10.1). The AWS EC2 instance also hosted a shared instance of the AGNES development tools (see Section 3.10.3), which the AGNES developers replicated as needed.

3.1.2 Ubuntu 14.04

The AGNES project used Ubuntu 14.04 LTS (Long Term Support) as its base operating system for development and, initially, its target operating system. UNIX-like operating systems form the base of many router operating systems, including the OpenBSD RIPv2 implementation which was targeted for comparison to the AGNES generated code.

Due to the low-level kernel interface required to implement the chosen RIPv2 protocol, the generated code was specific to the Linux kernel selected.

3.2 Core Auto-Generation Engine

The auto code generation engine, called Content Generation from Templates (Cogent),[3] was used on a number of previous projects before AGNES and is a stable piece of software. The original Cogent code generator used XML as input and was modified for AGNES to use RDF as input. Both the legacy XML and the AGNES RDF versions of Cogent are written in the Racket programming language. Both versions use the following packages:

- The open-source Racket Inference Collection implements an inference engine that supports both forward-chaining (data-driven) and backward chaining (goal-driven) for developing rule-based systems
- The `scribble/text` language provided with Racket acts as "preprocessor" language for generating text. This language uses the same @ syntax as the main Racket Scribble tool, but instead of working in terms of a document abstraction that can be rendered to text and HTML (and other formats), the preprocessor language works in a way that is more specific to the target formats – C source code in the case of AGNES.

The major difference between the two versions of Cogent is the external data abstraction used – XML versus RDF. The fundamental differences between these representations – XML uses a tree structured representation while RDF uses a graph structured representation – require substantially different processing algorithms.

For its external (and internal) data abstraction, Cogent-XML uses the following packages:

- The `xml` library provided with Racket provides functions for parsing and generating XML. XML can be represented as an instance of the document structure type, or as a kind of S-expression that is called an X-expression.
- The `html` library provided with Racket provides functions to read conformant HTML4 documents and structures to represent them. This library is used to read HTML documents as XML.

For its external (and internal) data abstraction, Cogent-RDF used the following package:

- The `sesame` library provides an interface to Racket, which is an open-source framework for storage, inferencing, and querying of RDF data. This library was developed as part of the AGNES program and is intended to be open sourced.

While most of the AGNES code generation is done by Cogent-RDF, some code generation is done by Cogent-XML where the source data is in XML. For example, we developed an application using Cogent-XML to create RDF representations of RFC documents published by the IETF to define Internet protocols.

[3] Code generation is no different from generating any other textual content – which is why the term "content generation" is used instead of "code generation" in the name.

3.2.1 Cogent-XML – Legacy

The AGNES code generator was bootstrapped with the legacy Cogent-XML generator. This generation process was done by the following as illustrated in Figure 3:

- The Cogent Framework parses the XML document, feeding successive pieces to the inference engine, which processes them according to the ruleset. It then generates content from the results using the templates.
- A Cogent Application consists of the Cogent framework along with a ruleset, templates, and any other code the application may require.
- Cogent takes an XML schema document (example), which describes an XML document structure and how to process it, and generates a ruleset that processes those XML documents within the Cogent framework.

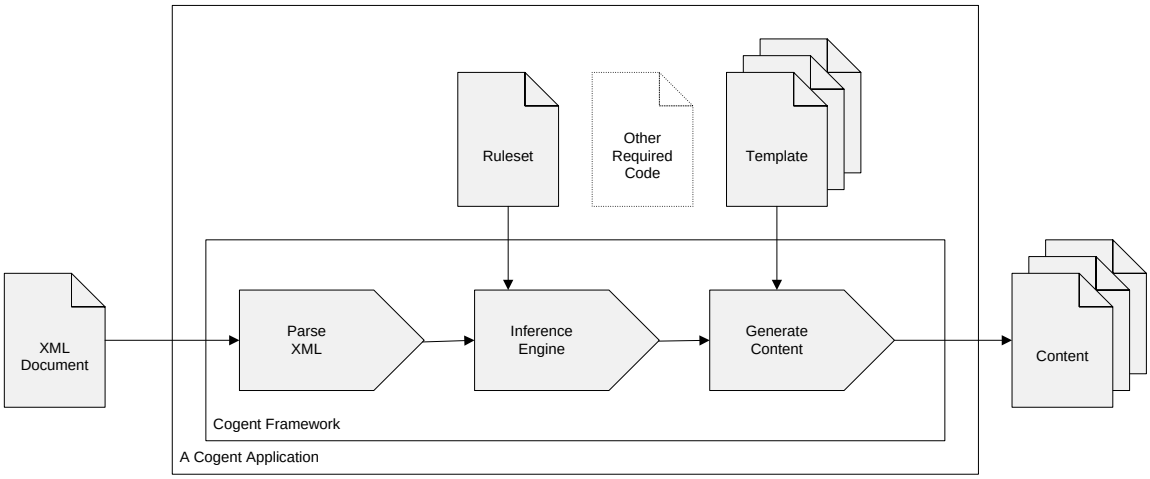

Figure 3. A Cogent application accepts XML documents as input, processes them using a ruleset, and generates contents per templates.

Cogent is itself a Cogent application, that is, Cogent is used to generate Cogent code. The Cogent schema document describes itself and the Cogent framework generates the Cogent ruleset. The ruleset template generates rulesets (Figure 4). An initial hand-coded Cogent ruleset was used to bootstrap the process.

3.2.2 Cogent-RDF Framework

The AGNES project developed the Cogent-RDF application using Cogent-XML as the starting point. Processing arbitrary RDF graph structured data is more complex than processing XML tree structured data. Specifically, walking a tree structure is algorithmically simpler than walking an arbitrary graph structure.

The Cogent-RDF framework works by parsing the command line and connecting to the RDF4J (Sesame) server. It then traverses the specified graph, asserts facts for each node, runs the inference engine for each node, and initiates code generation.

The Cogent-RDF framework itself was stable and did not require any extensive changes throughout the AGNES program. However, the rule set to process and infer RDF graphs and the code templates to generate C code were developed over the course of the AGNES project. The following describes these needed developments.

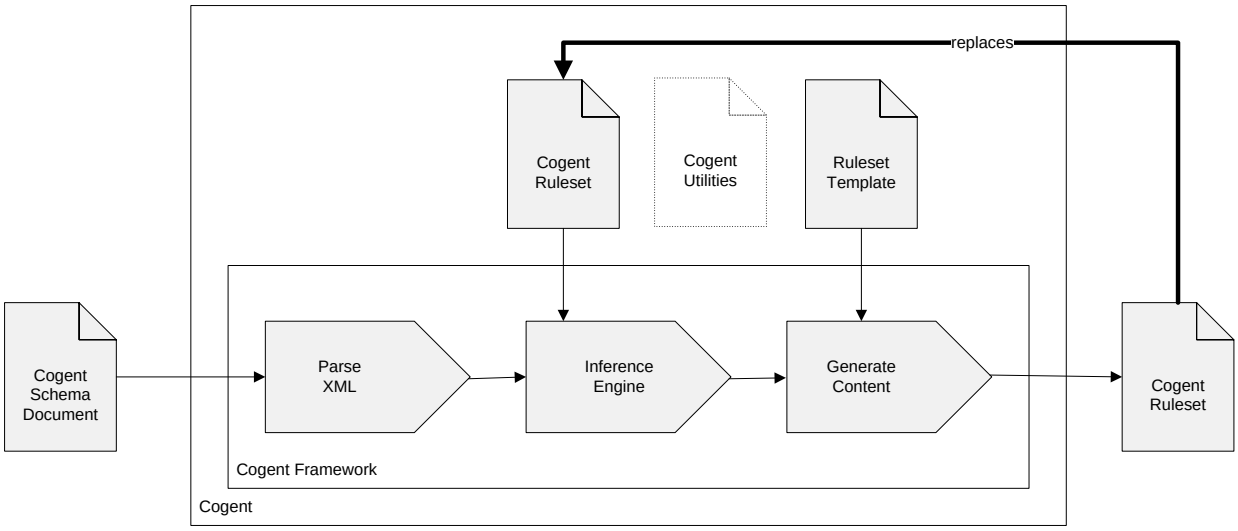

Figure 4. Cogent is self-generating. That is, it is generated by Cogent.

3.2.2.1 *Cogent-RDF Rule Set Development*

The following components of Cogent-RDF required further development to take into account the network element domain and use of the coding rules that support the CWEs:

- Recognition rules for program elements
- Rules that infer structural relationships among program elements
- Recognition rules for potential weaknesses in program elements.

3.2.2.2 *Code Template Development*

The following code template modifications were made:

- Templates were modified to use Scribble syntax for text generation
- The internal knowledge base populated with the Cogent-RDF rule set was used instead of the Sesame knowledge base
- Templates for program elements were developed for code generation
- Templates for code that mitigate code weaknesses were developed for code generation.

3.2.3 AGNES Ontology

An initial step in the AGNES project was the development of an ontology for describing internet protocols to a level required for the automatic generation of network element software. This was broken down into several independent ontologies that cover different aspects of the problem:

- Code Generation Ontology
- Network Protocol Ontology
- Weakness (i.e., CWE) Ontology
- RFC Ontology.

3.2.3.1 *Code Generation Ontology*

The code generation ontology covers basic data structure and algorithm descriptions to the level required to describe structures and processing in network protocols. Some of these elements are inherent in the structure of RDF and RDF Schema. This includes classes / subclasses, relations / sub-relations, collections, and high-level descriptive elements. Also, the data types from XML schema definitions (XSD) are also inherent in RDF.

We defined elements to describe data structures.

- Collections (sets, sequences, and bags) are inherent in RDF and are used to describe many data structures
    - Graphs are based on sets – G = (V, E), where V is a set of vertexes and E is a set of edges between vertexes
    - Stacks, queues, and deques are based on sequences
- Structures
    - Sequences of Fields
    - With Subfields
    - Offsets
- Arrays

We defined elements to describe algorithms at the level of pseudo-code.

- Blocks are sequences of statements
- Statements include
    - Assignments
    - Alternation (if / then / else) – a sequence of alternatives
    - Selection (case) – sequence of selectors
    - Iteration – a sequence of iterators
        - While / Until / Forever
        - Indexed – For
        - Data structure iterators
    - Calls
        - Primitives (e.g. UDPReceiveFrom) are implemented as calls
- Expressions – expression trees (graphs).

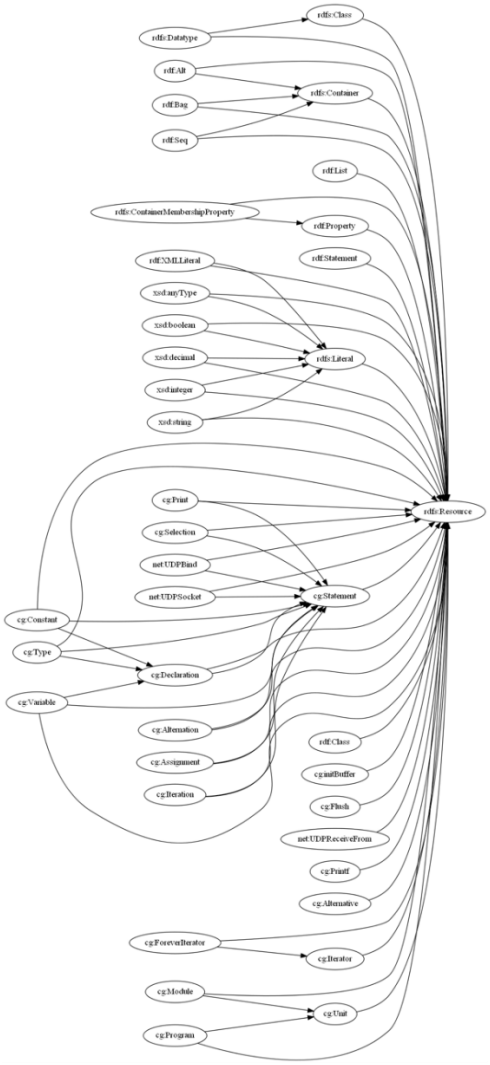

Figure 5. An early graph of the AGNES ontology.

Finally, there are high-level structure and control constructs:

- Units – the basic unit for which code can be generated
  - Programs – generated as main programs (i.e., executables)
  - Modules – generated as libraries (header files, etc.) [Not currently implemented]
- High-Level Control Strategies
  - Reactive Control Strategy – events are represented by file descriptors and the control loop implemented by select / poll
    - Timers
    - Kernel interface (netlink / rtnetlink)
    - Sockets
    - Files

3.2.3.2 *Network Ontology*

The network ontology includes the primitive elements required for high-level network protocols like RIP, such as IP, TCP, and UDP. In an ideal world, these protocols would themselves already have been described in RDF and would have their code automatically generated as well.

For AGNES, there are two specific network protocols that are required to implement RIP:

- User Datagram Protocol (UDP) – RIP is a UDP-based protocol
- Multicast – RIPv2 uses multicast to reduce its bandwidth requirements.

These are explicitly known by the code generators and are implemented as primitives. That is, they are not described in the RDF.

We also include the Routing Table Netlink (RTNetlink) as primitives in the network ontology. RTNetlink is a Linux protocol that allows user space programs, like the RIP server, to exchange routing table information with the kernel. This is a required functionality to implement RIP.

Over the course of the AGNES project the network ontology was expanded to describe the RIPv2 protocol.

3.2.3.3 *Weaknesses Ontology*

The Weaknesses Ontology that describes the CWEs was the least developed of the ontologies and was expanded during Phase 2. Previously, the weaknesses were recognized and mitigated in the primitives. Each primitive element had been manually evaluated for possible weaknesses through the CWE. Based on the results of the evaluation, the rule set was updated to recognize the potential weaknesses and the code templates were encoded with the mitigations.

For example, here is a summary of the analysis of the UDP primitives.

```
--------------------------------------------------------------------------------
Declaration net:UDPSocket

Declares a UDP socket. Uses the node label as the name of the socket.

Includes:
<sys/types.h>
<sys/socket.h>

Generated code:

int <node.label>;

if ((<node.label> = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
{
  perror("cg:Socket");
  exit(1);
}

Weaknesses:

None
--------------------------------------------------------------------------------
Statement net:UDPBind (<sockfd>, <addr>, <port>)
  <sockfd> : net:UDPSocket
  <addr> : net:SocketAddress_In
  <port> : int -- I'll use the xsd name instead of int
```

```
Includes:
<sys/types.h>
<sys/socket.h>

Generated code:

memset ((char *) &<addr>, 0, sizeof(<addr>));

<addr>.sin_family = AF_INET;
<addr>.sin_port = htos(<port>);
<addr>.sin_addr.s_addr = htol(INADDR_ANY);

if ((bind(<sockfd>, (struct sockaddr *) &<addr>, sizeof(<addr>))) == -1)
{
  perror("cg:Bind");
  exit(1);
}

Weaknesses:

None

Note that memset is safe because it explicitly uses the size of its buffer.
Note that bind is safe because it explicitly uses the size of its buffer.
--------------------------------------------------------------------------------
Statement net:UDPReceivefrom (<recv_len>, <sockfd>, <buffer>, <len>, <src_addr>)

Includes:
<sys/types.h>
<sys/socket.h>

Generated code:

{
    int slen = sizeof(src_addr);
    if ((<recv_len> = recvfrom (<sockfd>, *<buffer>, <len>, 0,
                                (struct sockaddr *) &<src_addr>, &slen>)) == -1)
    {
      perror("cg:Bind");
      exit(1);
    }
}

Weaknesses:

<len> may be larger than the capacity of <buffer>
<len> may be smaller than the length of <buffer>

Note that <src_addr> and slen are okay.
--------------------------------------------------------------------------------
Statement net:UDPSendTo (<sockfd>, <buffer>, <len>, <addr>)

Includes:
<sys/types.h>
<sys/socket.h>

Generated code:
if (sendto (<sockfd>, <buffer>, <len>, 0,
            (struct sockaddr *) &<addr>, (sizeof <addr>)) == -1)
{
  perror("cg:Bind");
  exit(1);
}

Weaknesses:

<len> may be larger than the capacity of <buffer>
<len> may be smaller than the length of <buffer>

Note that <addr> and sizeof(slen) are okay.
```

The implementation of the UDP primitives in the code generator was based on the above analysis so that weaknesses were recognized and mitigated.

3.2.3.4 *RFC Ontology*

As part of the `xmlrfc2rdf` task, we developed an ontology to represent IETF RFCs that were published in XML format. This is described in Section 3.5.

3.3 Selection of Weaknesses from the CWE Database

For the selection of the CWEs to be used in the proof of concept, the most frequently occurring software weaknesses, represented by the "Top 25" and "On the Cusp" subsets of the CWE database, were examined. The weaknesses chosen from these subsets had the following properties:

- Were relevant to network element software, especially RIP
- Arose during the implementation phase of a project (in contrast to weaknesses that arose during phases such as requirements or design)
- Were sufficiently concrete to evaluate whether software contains the weakness

We identified 26 weaknesses that met these criteria. From the 26, we selected 10 that were the most relevant to the code that was generated in the first year. Table 1 shows the 26 relevant weaknesses; bolded entries indicate the 10 weaknesses that became the focus of Phase 1.

Table 1. CWE Weaknesses identified as relevant to the AGNES project

Common Weakness Enumerations (CWE) Relevant to Router Information Protocol (RIP) Bold items were selected for use in Phase 1.
CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CWE-124 Buffer Underwrite ('Buffer Underflow')
CWE-127 Buffer Under-read
CWE-129 Improper Validation of Array Index
CWE-131 Incorrect Calculation of Buffer Size
CWE-134 Use of Externally-Controlled Format String
CWE-190 Integer Overflow or Wraparound
CWE-209 Information Exposure Through an Error Message
CWE-306 Missing Authentication for Critical Function

Common Weakness Enumerations (CWE) Relevant to Router Information Protocol (RIP) Bold items were selected for use in Phase 1.
CWE-307 Improper Restriction of Excessive Authentication Attempts
CWE-311 Missing Encryption of Sensitive Data
CWE-327 Use of a Broken or Risky Cryptographic Algorithm
CWE-330 Use of Insufficiently Random Values
CWE-456 Missing Initialization of a Variable
CWE-476 NULL Pointer Dereference
CWE-676 Use of Potentially Dangerous Function
CWE-681 Incorrect Conversion between Numeric Types
CWE-732 Incorrect Permission Assignment for Critical Resource
CWE-754 Improper Check for Unusual or Exceptional Conditions
CWE-759 Use of a One-Way Hash without a Salt
CWE-770 Allocation of Resources without Limits or Throttling
CWE-772 Missing Release of Resource after Effective Lifetime
CWE-798 Use of Hard-coded Credentials
CWE-805 Buffer Access with Incorrect Length Value
CWE-822 Untrusted Pointer Dereference
CWE-825 Expired Pointer Dereference

For each of the weaknesses listed in Table 1, rules in structured English were developed to detect when source code was subject to the weakness, with additional rules describing how to avoid the weakness. For example, CWE-805 arises when a buffer access goes outside the bounds of the buffer. To avoid this weakness, it was ensured that the program tracked the size of every buffer and checked that every access was within bounds. Figure 6 shows the structured English rules for the CWE-805 weakness.

IF the program has a statement that accesses a buffer location
AND that statement is not within the scope of a condition guaranteeing that the location(s) accessed are within the buffer start and end bounds
THEN the program is subject to CWE-805 *Buffer Access with Incorrect Length Value*

(a) Detection rule

IF the program uses pointers (including arrays and strings)
THEN keep every pointer as part of a triple that also contains a pointer to the beginning of the buffer and the buffer size (in bytes)

IF the program has a statement that accesses a buffer location
AND that statement is not within the scope of a condition guaranteeing that the location(s) accessed are within the buffer start and end bounds (as specified by the triple)
THEN replace that statement with a conditional statement that checks whether the location(s) accessed are within the buffer start and end bounds and if so, performs the access, but if not, returns from the current function with an error-indicating return value

(b) Mitigation rules

Figure 6: Structured English rules for CWE-805

For the 10 weaknesses selected for Phase 1, we also implemented static analysis tools that checked the generated source code to verify that it complied with the mitigation rules for those weaknesses. Section 3.6 discusses these tools further.

3.4 Coding Rules Development

The project developed coding rules for basic network element functions, and coding rules for avoiding the selected weaknesses. The following discusses how the code generator uses the coding rules and gives examples of generated code. All code is correct through inspection and execution.

3.4.1 Example agnes:Null

The example program `agnes:Null` is a trivial program that does nothing. It demonstrated end-to-end operation of the code generation system with a program with no content.

3.4.1.1 *RDF Triples*

```
agnes:Null rdf:type cg:Program .
agnes:Null rdfs:label "null" .
agnes:Null rdfs:comment "Null program" .
```

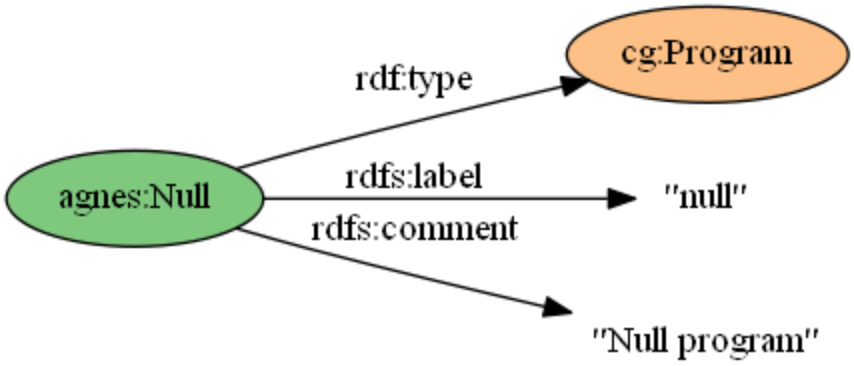

Figure 7. The RDF graph for the agnes:Null program.

3.4.1.2 *Generated Code*

```
// Content Generation from Templates - RDF (Cogent-RDF)
// Autogenerated C code null.c for program null
// Generated Tuesday, February 16th, 2016 3:29:33am
// Null program

int main ()
{
  return 0;
}
```

3.4.2 Example agnes:Hello

The example program `agnes:Hello` is the standard Hello, World! program. It demonstrated end-to-end operation of the code generation system with a program with minimal content.

3.4.2.1 *RDF Triples*

```
agnes:Hello rdf:type cg:Program .
agnes:Hello rdfs:label "hello" .
agnes:Hello rdfs:comment "\"Hello, World!\" program" .
agnes:Hello cg:body agnes:Hello_Statements_1 .

agnes:Hello_Statements_1 rdf:type rdf:List .
agnes:Hello_Statements_1 rdf:first agnes:Hello_Print .
agnes:Hello_Statements_1 rdf:rest rdf:nil .

agnes:Hello_Print rdf:type cg:Statement .
agnes:Hello_Print cg:type cg:Print .
agnes:Hello_Print cg:arguments agnes:Hello_Print_Arguments .

agnes:Hello_Print_Arguments rdf:type rdf:Seq .
agnes:Hello_Print_Arguments rdf:_1 "Hello, World!\n" .
```

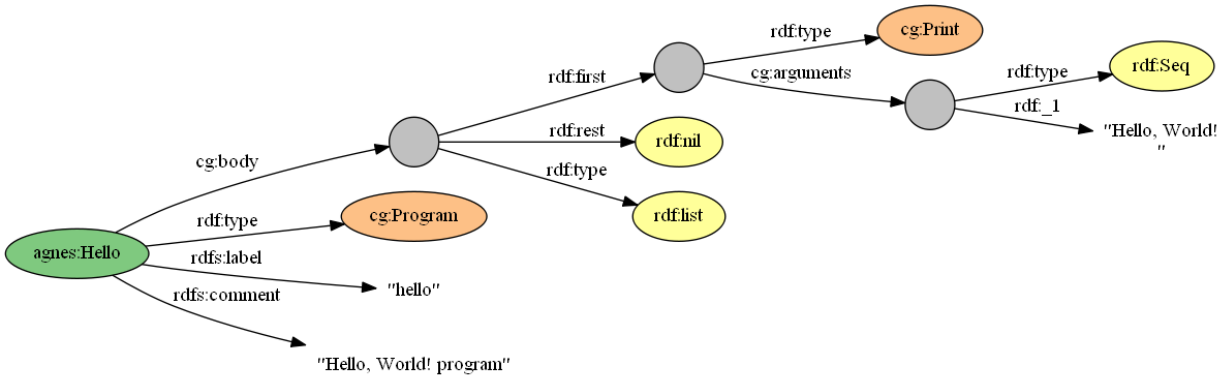

Figure 8. The RDF graph for the agnes:Hello program.

3.4.2.2 *Generated Code (some blank lines removed)*

```
// Content Generation from Templates - RDF (Cogent-RDF)
// Autogenerated C code hello.c for program hello
// Generated Tuesday, February 16th, 2016 3:41:28am
// "Hello, World!" program
#include <stdio.h>

int main ()
{

//
printf("Hello, World!\n");

  return 0;
}
```

3.4.3 Example agnes:Echo

The example program agnes:Echo implemented a simple messaging protocol with (potential) weaknesses that demonstrated UDP protocol primitives needed for RIP. This also demonstrated mitigation of weaknesses within (generated) primitives.

- CWE-456 Missing Initialization of a Variable
- CWE-754 Improper Check for Exceptional Conditional Conditions
- CWE-805 Buffer Access with Incorrect Length Value.

This example is larger than is practical to include in its entirety; however, the following figure shows the RDF graph for the example.

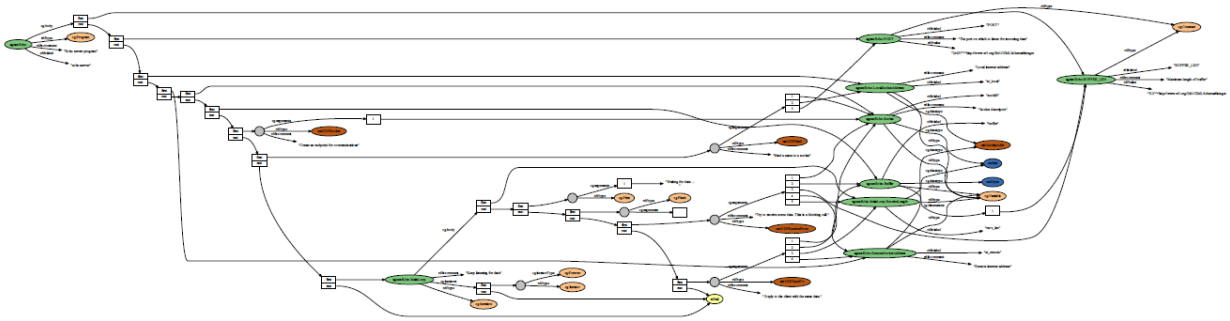

Figure 9. The RDF graph for the agnes:Echo program.

We used the subset of the RDF and generated code for udf:UDPReceiveFrom as an example showing weakness generation.

3.4.3.1 *Net:UDPReceiveFrom Call RDF Triples*

```
agnes:Echo_MainLoop_Receive rdf:type cg:Statement .
agnes:Echo_MainLoop_Receive cg:type net:UDPReceiveFrom .
agnes:Echo_MainLoop_Receive rdfs:comment "Try to receive some data. This is a blocking call." .
agnes:Echo_MainLoop_Receive cg:arguments . agnes:Echo_MainLoop_Receive_Arguments .

agnes:Echo_MainLoop_Receive_Arguments rdf:type rdf:Seq .
agnes:Echo_MainLoop_Receive_Arguments rdf:_1 agnes:Echo_Socket .
agnes:Echo_MainLoop_Receive_Arguments rdf:_2 agnes:Echo_Buffer .
agnes:Echo_MainLoop_Receive_Arguments rdf:_3 agnes:Echo_BUFFER_LEN .
agnes:Echo_MainLoop_Receive_Arguments rdf:_4 . agnes:Echo_RemoteSocketAddress .
agnes:Echo_MainLoop_Receive_Arguments rdf:_5 . agnes:Echo_MainLoop_Receive_Length .
```

This call has a potential weakness *CWE-805 Buffer Access with Incorrect Length Value*. The buffer length argument may be greater than the capacity of the buffer. If both the length and the capacity arguments are constants, the code generator statically checks it. Otherwise, mitigation is added to the generated code.

The other applicable CWEs, CWE-456 and CWE-754, are always mitigated in the generated code.

3.4.3.2 *Buffer Length Known Good*

The below shows the generated code when both the buffer length and capacity are known – that is, are constants.

```
// Try to receive some data. This is a blocking call.
  {
    int slen = sizeof(si_remote);

    // Mitigate CWE-456 Missing Initialization of a Variable
    memset(buf, '\0', BUFFER_LEN)
    // Mitigate CWE-754 Improper check for Exceptional Conditions
    if ((recv_len=recvfrom(sockfd, buffer, BUFFER_LEN, 0, (struct sockaddr *) &si_remote, &slen))
== -1)
    {
      perror("net:UDPReceiveFrom");
      exit(1);
    }
```

No generated CWE-805 mitigation – not needed

CWE-456 mitigated

CWE-754 mitigated

3.4.3.3 *Buffer Length Not Known Good*

The below shows the generated code when either the buffer length or capacity are not known. In this case, the code generator must generate mitigation for CWE-805.

```
// Try to receive some data. This is a blocking call.
  {
    int slen = sizeof(si_remote);

    // Mitigate CWE-805 Buffer Access with Incorrect Length Value
    if (1024 > BUFFER_LEN)
    {
      exit(1);
    }
    // Mitigate CWE-456 Missing Initialization of a Variable
    memset(buf, '\0', BUFFER_LEN)
    // Mitigate CWE-754 Improper check for Exceptional Conditions
    if ((recv_len=recvfrom(sockfd, buffer, 1024, 0, (struct sockaddr *) &si_remote, &slen)) == -
1)
    {
      perror("net:UDPReceiveFrom");
      exit(1);
    }
  }
```

CWE-805 mitigation against known capacity

Buffer length was set to a constant 1024, which is larger than the known buffer capacity of 512.

3.4.3.4 *Generated Echo Code*

Below is the code that was generated for the Echo program from the RDF graph. Note that the code includes comments, etc. from the graph.

```
// Content Generation from Templates - RDF (Cogent-RDF)
// Autogenerated C code echo-server.c for program echo-server
// Generated Tuesday, February 16th, 2016 2:50:11am
// Echo server program
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h

#define BUFFER_LEN 512 // Maximum length of buffer
#define PORT 24637 // The port on which to listen for incoming data

int main ()
{
  struct sockaddr_in si_local; // Local internet address
  struct sockaddr_in si_remote; // Remote internet address
  int sockfd; // Socket descriptor
  char buffer[BUFFER_LEN];

// Create an endpoint for communications
if ((sockfd=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
{
  perror("net:UDPSocket");
  exit(1);
}

// Bind a name to a socket
// Zero out the structure
memset((char *) &si_local, 0, sizeof(si_local));

// Initialize the internet socket address
si_local.sin_family = AF_INET;
si_local.sin_port = htons(PORT);
si_local.sin_addr.s_addr = htonl(INADDR_ANY);

// Bind socket to port
```

Includes inferred from generated statements

Constants generated as preprocessor #define directives

Generated variable declarations

Body generated from RDF graph

```
if (bind(s, (struct sockaddr*) &si_local, sizeof(si_local)) == -1)
{
  perror("net:UDPBind");
  exit(1);
}

// Keep listening for data
while (1)
  {
    int recv_len;
  //
  printf("Waiting for data ...");
  //
  fflush(stdout);
  // Try to receive some data. This is a blocking call.
  {
    int slen = sizeof(si_remote);
    if ((recv_len=recvfrom(sockfd, buffer, BUFFER_LEN, 0, (struct sockaddr *) &si_remote, &slen))
== -1)
    {
      perror("net:UDPReceiveFrom");
      exit(1);
    }
  }

  // Reply to the client with the same data.
  if (sendto(sockfd, buffer, recv_len, 0, (struct sockaddr *) &si_remote, sizeof(si_remote)) == -
1)
  {
    perror("net:UDPSendTo");
    exit(1);
  }

  }

  return 0;
}
```

While loop generated for iteration

3.5 XML RFC to RDF (xmlrfc2rdf)

RFC 7749 The "xml2rfc" Version 2 Vocabulary describes the XML vocabulary used to document internet protocols in XML. This is an initial step by the IETF to produce machine readable protocol specifications.

The IETF has published many RFCs in XML – including RFC 2453 Router Information Protocol Version 2 (RIPv2), which we used as our target protocol for AGNES. For now, these XML documents are just the text forms of the RFC with some high-level information – title, author, etc. – encoded. The rest is mostly English text inside of XML document, section, subsection, and paragraph tags.

We developed a tool using Cogent-XML, called `xmlrfc2rdf`, that accepted RFC documents in XML format and generated RDF documents (in Terse RDF Triple Language (Turtle) format [55]) that can be loaded into the AGNES RDF triple store. This allowed us to directly reference RFC elements in our protocol descriptions.

A long-term goal would be to extract descriptive elements from the XML which, in general, would require natural language processing. However, this was not a goal of AGNES.

3.5.1 XML Tags Processed by xmlrfc2rdf

The following is a hierarchical breakdown of the XML tags recognized by the `xmlrfc2rdf` application. It includes all of the XML tags defined in the current version of RFC 7749.

```
element rfc
+-- attribute number
+-- attribute obsoletes
+-- attribute updates
+-- attribute category
+-- attribute consensus
+-- attribute seriesNo
+-- attribute ipr
+-- attribute iprExtract
+-- attribute submissionType
+-- attribute docName
+-- attribute xml:lang
+-- element front ...
+-- element middle
    +-- element section ...
+-- element back
    +-- element references [this is a list]
        +-- attribute title
        +-- element reference
            +-- attribute anchor
            +-- attribute target
            +-- element front ...
            +-- element seriesInfo
                +-- attribute name
                +-- attribute value
            +-- element format
                +-- attribute target
                +-- attribute type
                +-- attribute octets
            +-- element annotation
                +-- element xref ...
                +-- element eref ...
                +-- element iref ...
                +-- element cref ...
                +-- element spanx ...
                +-- text
    +-- element section ...

element rfc
+-- ...
    +-- element reference|rfc
        +-- element front
            +-- element title
                +-- attribute abbrev
                +-- text
            +-- element author
                +-- attribute initials
                +-- attribute surname
                +-- attribute fullname
                +-- attribute role
                +-- element organization
                    +-- attribute abbrev
                    +-- text
                +-- element address
                    +-- element postal
                        +-- street
                            +-- text
                        +-- city
                            +-- text
                        +-- region
                            +-- text
                        +-- code
                            +-- text
```

```
                        +-- country
                            +-- text
                    +-- element phone
                        +-- text
                    +-- element facsimile
                        +-- text
                    +-- element email
                        +-- text
                    +-- element uri
                        +-- text
            +-- element date
                +-- attribute day
                +-- attribute month
                +-- attribute year
            +-- element area
                +-- text
            +-- element workgroup
                +-- text
            +-- element keyword
                +-- text
            +-- element abstract
                +-- element t ...
            +-- element note
                +-- attribute title
                +-- element t ...

element rfc
+-- ...
    +-- element back|middle|section
        +-- element section
            +-- attribute anchor
            +-- attribute title
            +-- attribute toc
            +-- element t ...
            +-- element figure ...
            +-- element texttable
                +-- attribute anchor
                +-- attribute title
                +-- attribute suppress-title
                +-- attribute align
                +-- attribute style
                +-- preamble ...
                +-- ttcol
                    +-- attribute width
                    +-- attribute align
                    +-- text
                +-- c
                    +-- element xref ...
                    +-- element eref ...
                    +-- element iref ...
                    +-- element cref ...
                    +-- element spanx ...
                    +-- text
                +-- postamble ...
            +-- element iref ...
            +-- element section ...

element rfc
+-- ...
    +-- element abstract|list|note|section
        +-- element t
            +-- attribute anchor
            +-- attribute hangText
            +-- element list
                +-- attribute style
                +-- attribute hangIndent
                +-- attribute counter
                +-- element t ...
            +-- element figure ...
            +-- element xref ...
            +-- element eref ...
```

```
            +-- element iref ...
            +-- element cref ...
            +-- element spanx ...
            +-- element vspace
                +-- attribute blankLines
            +-- text

element rfc
+-- ...
    +-- element annotation|c|postamble|preamble|t
        +-- element xref
            +-- attribute target
            +-- attribute pageno
            +-- attribute format
            +-- text

element rfc
+-- ...
    +-- element annotation|c|postamble|preamble|t
        +-- element eref
            +-- attribute target
            +-- text

element rfc
+-- ...
    +-- element annotation|c|figure|postamble|preamble|section|t
        +-- element iref
            +-- attribute item
            +-- attribute subitem
            +-- attribute primary

element rfc
+-- ...
    +-- element annotation|c|postamble|preamble|t
        +-- element cref
            +-- attribute anchor
            +-- attribute source
            +-- text

element rfc
+-- ...
    +-- element annotation|c|postamble|preamble|t
        +-- element spanx
            +-- attribute xml:space
            +-- attribute style
            +-- text

element rfc
+-- ...
    +-- element section|t
        +-- element figure
            +-- attribute anchor
            +-- attribute title
            +-- attribute suppress-title
            +-- attribute src
            +-- attribute align
            +-- attribute alt
            +-- attribute width
            +-- attribute height
            +-- element iref ...
            +-- element preamble ...
            +-- element artwork
                +-- attribute xml:space
                +-- attribute name
                +-- attribute type
                +-- attribute src
                +-- attribute align
                +-- attribute alt
                +-- attribute width
                +-- attribute height
                +-- text
```

```
            +-- element postamble ...

element rfc
+-- ...
    +-- element figure|texttable
        +-- element preamble
            +-- element xref ...
            +-- element eref ...
            +-- element iref ...
            +-- element cref ...
            +-- element spanx ...
            +-- text

element rfc
+-- ...
    +-- element figure|texttable
        +-- element postamble
            +-- element xref ...
            +-- element eref ...
            +-- element iref ...
            +-- element cref ...
            +-- element spanx ...
            +-- text
```

3.5.2 The xmlrfc2rdf Application

The `xmlrfc2rdf` application consists of:

- The rule set that defines the structure of the RFC XML file and the rules to infer the structural elements from it
- The template to generate the Turtle RDF file.

In this case, the rule set completely infers the structure of the RDF graph from the XML format of the data. This allows an essentially trivial template that just formats the inferred triples. The following is a complete template:

```
@"@"prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@"@"prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@"@"prefix xsd: <http://www.w3.org/2001/XMLSchema/> .
@"@"prefix dc: <http://purl.org/dc/elements/1.1/> .
@"@"prefix rfc: <http://www.example.com/rfc/> .

@in[(subject predicate object) (in-query '(triple ?subject ?predicate ?object))]{
@subject @predicate @(if (string? object) (list "\"" object "\"") (list object)) .

}
```

3.5.3 Snippets of Generated Turtle RDF for RFC 2453

The entire generated Turtle RDF file is 1258 individual triples, each of which corresponds to a <subject> <predicate> <object>.

The following is a snippet of the front material:

```
        rfc:RFC2453 rdf:type rfc:RFC .
        rfc:RFC2453 rfc:number "2453" .
        rfc:RFC2453 rfc:obsoletes "1723, 1388" .
        rfc:RFC2453 rfc:updates "" .
        rfc:RFC2453 rfc:category "std" .
        rfc:RFC2453 rfc:seriesNo "56" .
        rfc:RFC2453 rfc:submissionType "IETF" .
        rfc:RFC2453 xml:lang "en" .
```

```
rfc:RFC2453 rfc:front _:front710 .
rfc:RFC2453 dc:title "RIP Version 2" .
_:title711 rdf:title "RIP Version 2" .
rfc:RFC2453 dc:creator "Gary Scott Malkin" .
_:front710 rfc:author _:author712 .
_:author712 rfc:initials "G.S." .
_:author712 rfc:surname "Malkin" .
_:author712 rfc:fullname "Gary Scott Malkin" .
_:node713 rdf:type rdf:List .
_:node713 rdf:first _:author712 .
_:front710 rfc:authors _:node713 .
_:organization714 rdf:organization "Bay Networks" .
_:author712 rfc:address _:address715 .
_:address715 rfc:postal _:postal716 .
_:postal716 rfc:street "8 Federal Street" .
_:postal716 rfc:street "Billerica" .
_:postal716 rfc:street "MA 01821" .
_:address715 rfc:phone "(978) 916-4237" .
_:address715 rfc:email "gmalkin@baynetworks.com" .
_:front710 rfc:date _:date717 .
_:date717 rfc:month "November" .
_:date717 rfc:year "1998" .
_:front710 rfc:area "Routing" .
_:node719 rdf:type rdf:List .
_:node719 rdf:first _:area718 .
_:front710 rfc:areas _:node719 .
_:front710 rfc:keyword "routing" .
_:front710 rfc:keyword "routing information protocol" .
_:front710 rfc:keyword "security" .
_:front710 rfc:abstract _:abstract720 .
_:abstract720 rfc:t _:t721 .
_:t721 rfc:text "This document specifies an extension of the Routing Information Protocol
(RIP), as defined in" .
_:t721 rfc:xref _:xref722 .
_:xref722 rfc:target "RFC1058" .
_:xref722 rfc:pageno "false" .
_:xref722 rfc:format "default" .
_:t721 rfc:text ", to expand the amount of useful information carried in RIP messages and
to add a measure of security." .
_:abstract720 rfc:t _:t723 .
_:t723 rfc:text "A companion document will define the SNMP MIB objects for RIP-2" .
_:t723 rfc:xref _:xref724 .
_:xref724 rfc:target "RFC1389" .
_:xref724 rfc:pageno "false" .
_:xref724 rfc:format "default" .
_:t723 rfc:text ". An additional document will define cryptographic security improvements
for RIP-2" .
_:t723 rfc:xref _:xref725 .
_:xref725 rfc:target "RFC2082" .
_:xref725 rfc:pageno "false" .
_:xref725 rfc:format "default" .
_:t723 rfc:text "." .
_:node713 rdf:rest rdf:nil .
```

The following is a typical small section of the document describing elements of the RIP protocol:

```
_:section874 rfc:t _:t881 .
_:t881 rfc:text "There are two timers associated with each route, a \"timeout\" and a \"garbage-
collection\" time. Upon expiration of the timeout, the route is no longer valid; however, it is
retained in the routing table for a short time so that neighbors can be notified that the route
has been dropped. Upon expiration of the garbage-collection timer, the route is finally removed
from the routing table." .
_:section874 rfc:t _:t882 .
_:t882 rfc:text "The timeout is initialized when a route is established, and any time an update
message is received for the route. If 180 seconds elapse from the last time the timeout was
initialized, the route is considered to have expired, and the deletion process described below
begins for that route." .
_:section874 rfc:t _:t883 .
_:t883 rfc:text "Deletions can occur for one of two reasons: the timeout expires, or the metric
is set to 16 because of an update received from the current router (see section 3.7.2 for a
```

```
discussion of processing updates from other routers). In either case, the following events
happen:" .
_:t883 rfc:list _:list884 .
_:list884 rfc:t _:t885 .
_:t885 rfc:text "- The garbage-collection timer is set for 120 seconds." .
_:list884 rfc:t _:t886 .
_:t886 rfc:text "- The metric for the route is set to 16 (infinity). This causes the route to be
removed from service." .
_:list884 rfc:t _:t887 .
_:t887 rfc:text "- The route change flag is set to indicate that this entry has been changed." .
_:list884 rfc:t _:t888 .
_:t888 rfc:text "- The output process is signaled to trigger a response." .
_:section874 rfc:t _:t889 .
_:t889 rfc:text "Until the garbage-collection timer expires, the route is included in all updates
sent by this router. When the garbage-collection timer expires, the route is deleted from the
routing table." .
_:section874 rfc:t _:t890 .
_:t890 rfc:text "Should a new route to this network be established while the garbage- collection
timer is running, the new route will replace the one that is about to be deleted. In this case
the garbage-collection timer must be cleared." .
_:section874 rfc:t _:t891 .
_:t891 rfc:text "Triggered updates also use a small timer; however, this is best described in
section 3.9.1." .
```

Finally, the following is an artwork section that shows the structure of a protocol packet:

```
_:figure954 rfc:artwork _:artwork955 .
_:artwork955 xml:space "preserve" .
_:artwork955 rfc:name "" .
_:artwork955 rfc:type "" .
_:artwork955 rfc:align "left" .
_:artwork955 rfc:width "" .
_:artwork955 rfc:height "" .
_:artwork955 rfc:text "" .
_:artwork955 rfc:text "    0                   1                   2                   3 3" .
_:artwork955 rfc:text "    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1" .
_:artwork955 rfc:text "   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+" .
_:artwork955 rfc:text "   | Command (1)   | Version (1)   |            unused             |" .
_:artwork955 rfc:text "   +---------------+---------------+-------------------------------+" .
_:artwork955 rfc:text "   |             0xFFFF            |    Authentication Type (2)    |" .
_:artwork955 rfc:text "   +-------------------------------+-------------------------------+" .
_:artwork955 rfc:text "   ~                       Authentication (16)                     ~" .
_:artwork955 rfc:text "   +---------------------------------------------------------------+" .
```

3.6 Implementation of the RIP Protocols

Early accomplishments during Phase I focused on defining the key data elements necessary to implement RIPv2 routing per RFC 2453. Specifications in the RFC and existing implementations of RIP in the Zebra/Quagga framework were examined. From this examination, specifications of the XML Schema Definition (XSD) of the RIP packet and the RIP router table entry were written. Figure 10 contains the XSD schema developed for these constructs along with the C implementations in Quagga as a comparison. The three elements of particular interest are the RIPPacket, RTE, and RouterTable schema definitions. These schemas were transformed into the appropriate representations used by the auto-code generator.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!--
defines and structs from quagga. ripd.h
-->
<!--
/* Normal RIP packet min and max size. */
```

```
#define RIP_PACKET_MINSIZ               4
#define RIP_PACKET_MAXSIZ             512
#define RIP_HEADER_SIZE                 4
#define RIP_RTE_SIZE                   20
/* Max count of routing table entry in one rip packet. */
#define RIP_MAX_RTE   ((RIP_PACKET_MAXSIZ - RIP_HEADER_SIZE) / RIP_RTE_SIZE)
-->

  <xs:element name="RIPPacket">
    <xs:complexType>
      <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="command" type="xs:byte"/>
    <xs:element name="version" type="xs:byte"/>
    <xs:element name="pad1" type="xs:byte"/>
    <xs:element name="pad2" type="xs:byte"/>
    <xs:element name="rte" type="xs:unsignedByte" minOccurs="1" maxOccurs="25"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

<!--- /* RIP routing table entry which belong to rip_packet. */
struct rte
{
  u_int16_t family;              /* Address family of this route. */
  u_int16_t tag;          /* Route Tag which included in RIP2 packet. */
  struct in_addr prefix;  /* Prefix of rip route. */
  struct in_addr mask;           /* Netmask of rip route. */
  struct in_addr nexthop; /* Next hop of rip route. */
  u_int32_t metric;              /* Metric value of rip route. */
};
-->

<!--
RTE uses unsignedLong as the type for address, per definition from <netinet/in.h> - uint32_t
RTE is 20 bytes
-->
  <xs:element name="RTE">
    <xs:complexType>
      <xs:sequence>
    <xs:element name="family" type="xs:unsignedShort"/>
    <xs:element name="tag" type="xs:unsignedShort"/>
    <xs:element name="prefix" type="xs:unsignedLong"/>
    <xs:element name="mask" type="xs:unsignedLong"/>
    <xs:element name="nexthop" type="xs:unsignedLong"/>
    <xs:element name="metric" type="xs:unsignedLong"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

<!--
As defined in quagga lib/table.h

/* Routing table top structure. */
struct route_table
{
  struct route_node *top;
  /*
   * Delegate that performs certain functions for this table.
   */
  route_table_delegate_t *delegate;
  unsigned long count;
  /*
   * User data.
   */
  void *info;
};
-->

  <xs:element name="RouterTable">
    <xs:complexType name="TableEntry">
      <xs:element name="entry" type="RTE" />
```

```
      <xs:element name="next" type="TableEntry" minOccurs="0" maxOccurs="1" />
    </xs:complexType>
  </xs:element>

<!-- Other useful constructs from include file:
/* RIP event. */
enum rip_event
{
  RIP_READ,
  RIP_UPDATE_EVENT,
  RIP_TRIGGERED_UPDATE,
};
-->
</xs:schema>
```

Figure 10. Specification of RIPv2 Data Elements in XSD Schema.

3.7 Static Analysis Tools for Coding Rules

As discussed in Sections 3.1 and 3.4, we selected 10 weaknesses from MITRE's CWE database as our focus and developed coding rules for the AGNES code generator to avoid these weaknesses. In this section, we describe the construction of static analysis tools used to verify that the AGNES-generated code complied with these coding rules. The rules are strict so that rule compliance can be statically checked.

Potentially Dangerous Function Call Checker – This tool checked that no calls are made to potentially dangerous functions as defined by "Security Development Lifecycle (SDL) Banned Function Calls" (http://msdn.microsoft.com/en-us/library/bb288454.aspx). This also checked compliance with the coding rule for CWE-676 *Use of Potentially Dangerous Function*.

Agnes Buffer Use Checkers – We developed an AGNES buffer implementation to ensure that the generated software would track buffer sizes and check bounds for buffer accesses. These four tools checked that the generated software correctly used the AGNES buffer implementation. Together they checked compliance with the following coding rules:

- CWE-120 *Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')*
- CWE-124 *Buffer Underwrite ('Buffer Underflow')*
- CWE-127 *Buffer Under-read*, CWE-129 *Improper Validation of Array Index*
- CWE-131 *Incorrect Calculation of Buffer Size*
- CWE-805 *Buffer Access with Incorrect Length Value*.

The following is a description of the four tools that checked for the above coding rules:

- **Agnes Buffer Bounds Checker** – This tool checked that each call to an AGNES buffer function either is making a request (buffer read, buffer write, or change to buffer offset) that is guaranteed to be within bounds, or has its return value used (so that the caller becomes aware of the bounds error in order to handle it appropriately).
- **Direct Agnes Buffer Access Checker** – This tool checked that there was no direct access to AGNES buffers, only access using the AGNES buffer functions. This included checking for direct field access, direct initialization of buffers, and casting to

or from buffers. (Casting to/from AGNES buffers would allow circumventing the other checks.)

- **Pointer Arithmetic Checker** – This tool checked that no pointer arithmetic (including array subscripting) was performed.
- **Buffer Size Calculation Checker**- This tool checked that the "sizeof" function was not applied to a pointer.

Integer Overflow Checker – This tool checked that no arithmetic operations were performed that may result in a value that exceeds the range of values of the underlying numerical representation. This checked compliance with the coding rule for CWE-190 *Integer Overflow or Wraparound*.

Exceptional Condition Handling Checker – This tool checked to make sure that each call to an `int`-returning function either was guaranteed to not return zero or had its return value used. The return value may indicate the status of the function call, so it should either be guaranteed to not indicate an error or checked so that errors may be handled. The checker accepts a whitelist; this should contain `int`-returning functions whose return values may be safely ignored. This tool checks compliance with the coding rules for CWE-754 *Improper Check for Unusual or Exceptional Conditions*.

Variable Initialization Checker – This tool checked that no variable was used before it was assigned a value. The checker accepts a whitelist; this should contain functions to which it is safe to pass a pointer that points to uninitialized memory (because the function writes to the memory without reading from it). This tool checked compliance with the coding rule for CWE-456 *Missing Initialization of a Variable*.

3.8 Investigation of RIP Implementation Basis

The team looked at several open-source implementations of RIP to be better familiarized with current RIPv2 implementations. Of several candidates, Quagga appeared to be one of the more popular and readily available implementations. The purpose of examining Quagga source code was twofold: (1) to become familiar with existing implementations, and (2) to understand requirements for replacing elements of the Quagga implementation with AGNES-generated code.

Our initial emphasis was to auto-generate the processing logic of RIP along with the key data elements. As we examined the code it became clear that without a full routing daemon framework, the AGNES code would not be easily tested, and correct routing behavior verified within a representative environment. As such, the team decided that a full implementation would be required by either replacing components of Quagga or as part of a more direct implementation. The team investigated an alternative implementation of RIPv2 within the OpenBSD environment. That implementation was more straightforward and provided the framework for the AGNES-generated implementation.

3.9 Investigation of Simulation/Emulation Environment

Initial Phase I efforts focused on the investigation of EMANE as a simulation / emulation environment for AGNES. EMANE provides a powerful environment for modeling mobile network systems. It focuses on real-time modeling of link and physical layer connectivity so that network protocol and application software can be experimentally subjected to the same conditions that are expected to occur in real-world mobile, wireless network systems.

Upon further investigation, the AGNES team felt that EMANE was more appropriate for later phases of testing as AGNES moved into the mobile network environment. For initial testing, the team chose the GNS3 graphical network simulator environment. GNS3 is a free, open source network simulation tool that includes a graphical interface for visualizing the network topology. Core and contributed appliances are available that implement a large variety of commercial and open-source routers.

Accomplishments during Phase I of this effort included:

- Standing up of a virtual machine that includes an installation of GNS3 with multiple Cisco router emulators.
- Construction of a layered network topology suitable for testing routing across multiple networks (Figure 11).

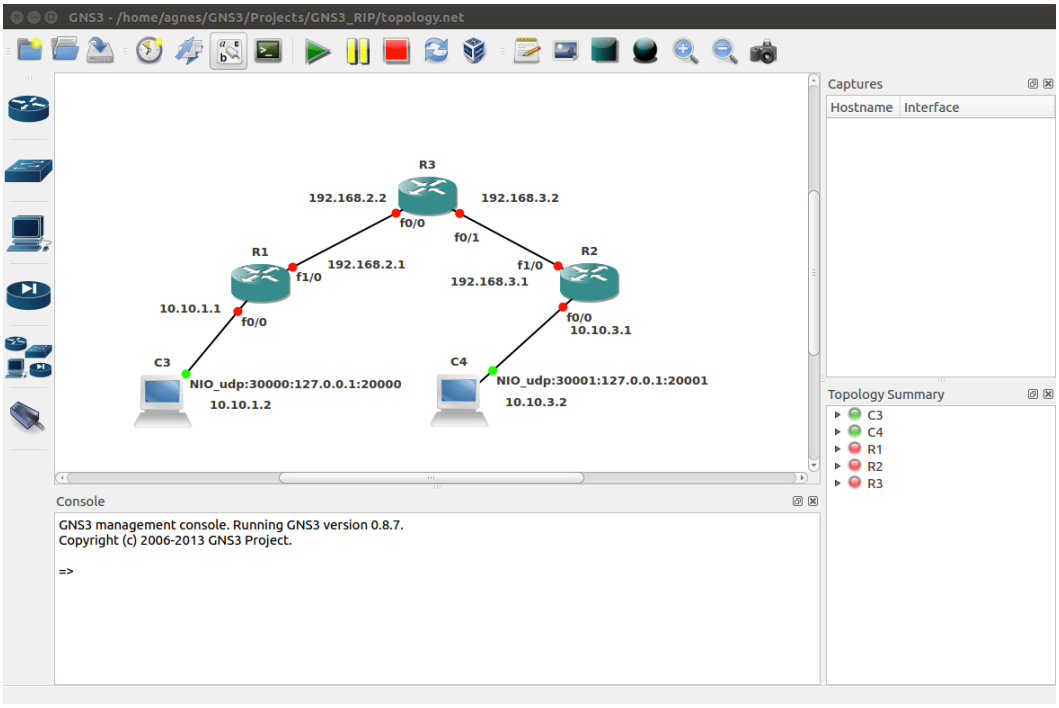

Figure 11. GNS3 Network Topology Suitable for Testing AGNES-generated Routers.

As Ubuntu-based servers running the AGNES-generated Router become available, they could replace one or more of the current emulated Cisco routers (E.g., R1, and/or R3). The project used

this environment during Phase 2 to test the interoperability of the AGNES generated routers with other existing routers running the RIP protocol.

3.10 Development Environment

3.10.1 GitLab

GitLab is a web-based Git repository hosting service (Figure 12). It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. GitLab Enterprise Edition builds on top of Git and includes extra features. It has LDAP group sync, audit logs and multiple roles. It includes deeper authentication and authorization integration, has fine-grained workflow management, has extra server management options and integrates with tool stacks. GitLab EE runs on servers (on premise) behind a firewall.

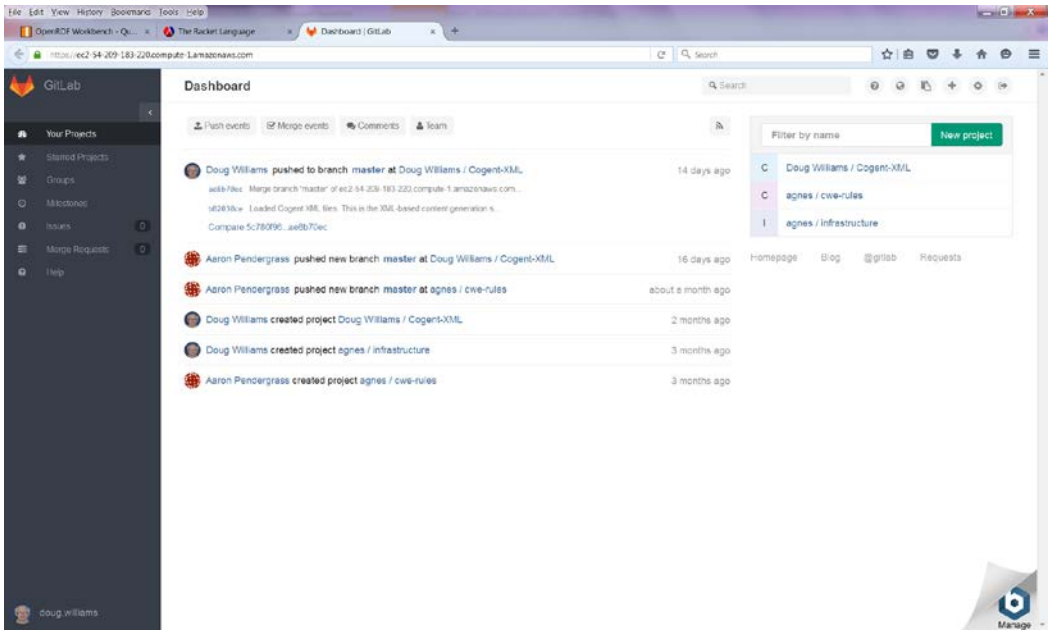

Figure 12. GitLab provides repository management, code reviews, issue tracking, activity feeds, and AGNES wiki.

GitLab provides the following:

- Version control and repository management based on Git
- Issue management and bug tracking
- Code Review functionality
- Continuous Integration tool (GitLab CI)
- ChatOp tool (Mattermost)
- Wiki
- Integration with IDEs
- Rich API

- On-premise or cloud -based installations
- Repository mirroring and high availability (HA)
- Development Analytics

3.10.2 Eclipse RDF4J (Sesame)

Eclipse RDF4J (previously known as Sesame) is a framework for processing and handling of RDF data. It provides tools for creating, parsing, storing, inference, and querying over RDF data. It also offers an easy to use REST interface.

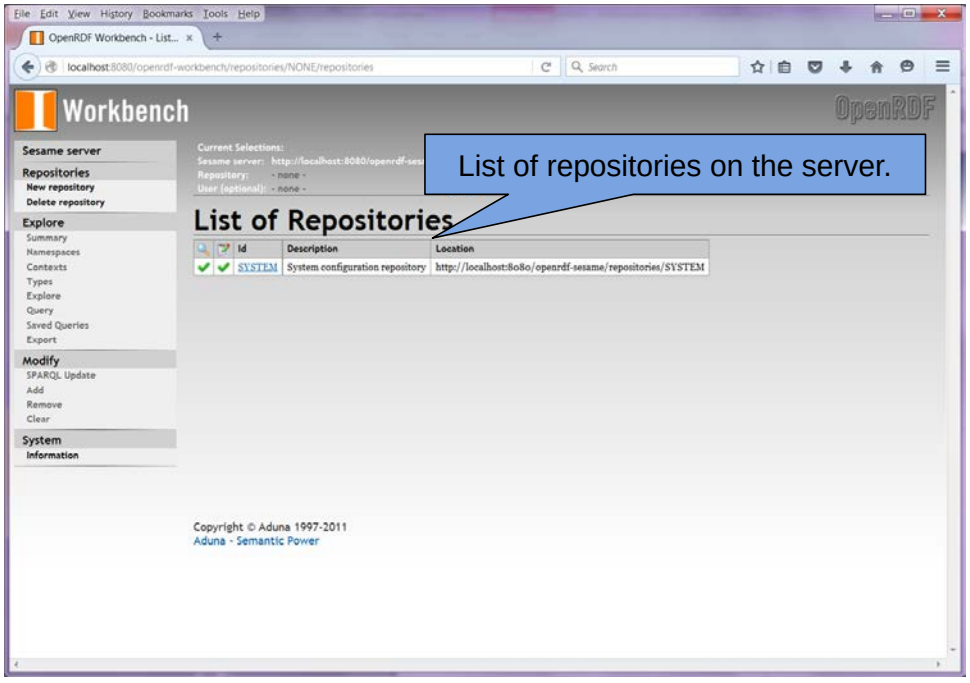

Figure 13. Eclipse RDF4J Workbench provides processing and handling of RDF data.

3.10.3 Development Tools

3.10.3.1 *Racket v6.2.1*

Racket is a full-spectrum programming language, based on Lisp and Scheme, used as the implementation language for AGNES. It provides the following advantages for the AGNES development:

- It has features beyond Lisp and Scheme with dialects that support objects, types, laziness, and more
- It enables programmers to link components written in different dialects, and empowers programmers to create new, project-specific dialects
- Racket's libraries support applications from web servers and databases to GUIs and charts
- Our existing code generation capabilities were in Racket
- Uses existing inference engine
- Same source code across Windows, Mac OS X, and Linux.

The AGNES program uses Racket as the development language for its code generators.

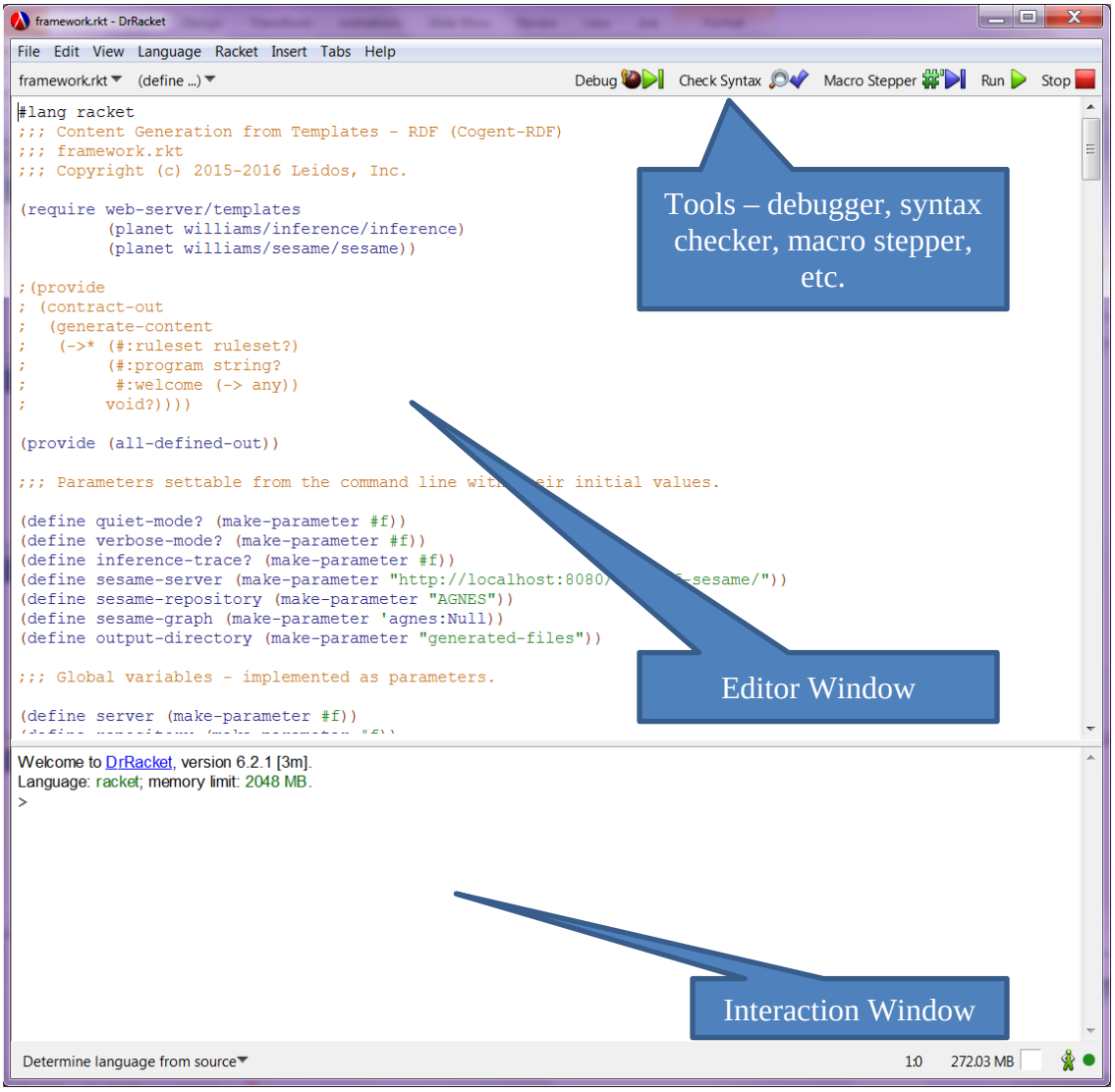

Figure 14. DrRacket is the Racket interactive development environment.

3.10.3.2 *GCC 4.8.2*

The GNU Compiler Collection (GCC) includes front ends for C, C++, Objective-C, Fortran, Ada, and Go, as well as libraries for these languages (libstdc++, ...). GCC is the compiler suite generally used for development under Linux.

The AGNES project used C as its target language and GCC as the target compiler. Some of the reasons these were chosen are:

- There are more 'interesting' weaknesses (e.g., memory management in Racket or Go internally addresses many weaknesses)
- There are several reference implementations of RIP written in C

- It is possible to use the 'way-back machine' to get older, less stable RIP versions for comparison
- Existing test and evaluation tools were available.

4 ACCOMPLISHMENTS

The following are the accomplishments for the three phases of the AGNES project. Phase 3 was not funded.

4.1 Phase 1 Accomplishments

Phase 1 of the AGNES project ran from July 2015 through June 2016. The focus of Phase 1 was to develop a framework to demonstrate the ability to develop coding rules to avoid network element weaknesses. The project selected known weaknesses related to network element software development and developed the core framework, auto-generation engine, and coding rules for network elements. Accomplishments included:

Task 1: Core Representation Framework and Ontologies

This task developed the automatic code generator software framework and ontologies that the code generator reads to produce the code. The knowledge base contains the RDF representation of the RFCs, software weaknesses, coding rules and the specification of the generated network element.

- Started the development of the core representation frameworks for specifying a network ontology, coding rules, and network elements
- Started the development of the framework for network ontology, coding rules, and network elements
- Started the development of the coding and network ontologies
- Started the development of the Networking Ontology based on IETF RFCs
- Started the development of the Software Weakness Ontology
- Started the development of the method to split network protocol descriptions at the RFC level from the description of the programs
- Started the development of the method to augment protocol descriptions to represent omitted details – common in RFCs

Task 2: Auto-Generation Engine

The auto code generation engine was developed prior to AGNES and is a stable piece of software. The original Cogent code generator used XML as input and was modified for AGNES to use RDF as input.

- Started the development of the core of the auto-generation engine
- Started the development of a reactive programming control strategy that supports TCP/UDP transmission, timers, kernel interface, etc.

- Started conversion of the code generator to process RDF, in addition to the existing XML capability
- Started the development of initial coding rules
- Started the development of the capability to generate code based on coding ontology
- Started prototyping coding rules and code templates using the Echo protocol

Task 3: Selection of CWE Database weaknesses

Known weaknesses were selected from the MITRE CWE database relevant to network element software. The most frequently occurring network element software weaknesses were identified and a subset was selected.

- Selected 26 relevant software weaknesses from the CWE database based relevant to network element software. (see Table 1):
  - Are relevant to network element software, especially RIP
  - Arise during the implementation phase of a project (in contrast to weaknesses arising during phases such as requirements or design)
  - Are sufficiently concrete to evaluate whether software contains the weakness
- From the 26 weaknesses selected, 10 were selected that were the most relevant to the code that was generated in Phase 1

Task 4: Coding Rules Development and Static Analysis Tools for Coding Rules

Coding rules for basic network element functions and for avoiding the selected weaknesses were developed.

- Started developing coding rules for basic network element functions, and coding rules to avoid the selected weaknesses, which included the following.
- Potentially Dangerous Function Call Checker (CWE-676)
- AGNES Buffer use Checkers, this included coding rules for:
  - AGNES Buffer Bounds Checker
  - AGNES Buffer Direct Access Checker (CWE-120)
  - CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
  - CWE-124 Buffer Underwrite ('Buffer Underflow')
  - CWE-127 Buffer Under-read
  - CWE-129 Improper Validation of Array Index
- Buffer Size Calculation
- Pointer Arithmetic Checker
- Integer Overflow Checker (CWE-190)
- Exceptional Condition Handling Checker (CWE-754)
- Variable Initialization Checker (CWE-456)
- NULL pointer dereference (CWE 476)
- Untrusted Pointer Dereference Checker (CWE-822)

Task 5: XML RFC to RDF

The team began researching a tool that would accept RFC documents in XML format and generate RDF documents (in Terse RDF Triple Language (Turtle) format) that could be loaded into the AGNES RDF triple store. This allowed for the direct reference of RFC elements in the protocol descriptions.

Task 6: Implementation of the RIP Protocols

Phase 1 accomplishments focused on defining the key data elements necessary to implement RIPv2 routing per RFC 2453:

- Partially implemented RIP using the auto-generation engine
- Began investigation of Zebra/Quagga as the basis of a RIP implementation
- Started developing code templates for control strategy, timers and kernel interface
- Started developing structures and algorithms for network element code generation
- Started developing reactive programming control strategy supporting TCP/UDP transmission, timers, kernel interface, etc.
- Started developing message processing, timer processing and global timer maintenance code
- Started developing router table kernel interface and router table processing code

Task 7: Investigation of Simulation/Emulation Environment

This task investigated the Extendable Mobile Ad-hoc Network Emulator (EMANE) as a simulation / emulation environment for testing generated code:

- Obtained documentation and read papers on EMANE as part of evaluation
- Investigated obtaining a copy of EMANE
- Investigated APL expertise on EMANE
- Investigated the Graphical Network Simulator-3 (GNS3) as an alternative simulation / emulation environment
  - Obtained documentation and read papers on GNS3 as part of evaluation
- Stood up a virtual machine that includes an installation of GNS3 with multiple Cisco routers emulators.
- Constructed a layered network topology in GNS 3 suitable for testing routing across multiple networks

Task 8: Evaluation of AGNES Generated Code

This task investigated testing the generated code against other simulated routers and static analysis tools to verify the generated software is free from the chosen software weaknesses.

- Started development of static analysis tools to verify that the generated software complies with the coding rules that implement the chosen CWEs

- Opted to develop separate static analysis tools for each CWE that was implemented due to the large differences in the tools (the core of the static analysis tools could be reused)
- Researched certification tests for RIPv2 routers

4.2 Phase 2 Results

During Phase 2, Version 1 of AGNES was completed. Between July 2016 and June 2017, the project completed the auto code generator framework and was on schedule to generate RIPv2 code in advance of a demonstration planned for late summer/fall of 2017. At this time, an additional goal was added to integrate the AGNES-generated RIP code into a hardware solution by the end of the 2017 calendar year.

The following lists the major tasks performed during Phase 2 in support of the goal of automatically generating a RIPv2 implementation from known CWE weaknesses, testing the code in a network simulation environment, analyzing the code for weaknesses and vulnerabilities, and running the generated code on a hardware device.

Task 1: Core Representation Framework and Ontologies

- Finished development of the framework for network ontology, coding rules, and network elements
- Finished development of the coding and network ontologies
- Finished development of the Networking Ontology based on IETF RFCs
- Finished development of the Software Weakness Ontology
- Finished development of the method to split network protocol descriptions at the RFC level from the description of the programs
- Finished development of the method to augment protocol descriptions to represent omitted details – common in RFCs

Task 2: Auto-Generation Engine

- Completed conversion of the code generator to process RDF, in addition to the existing XML capability
- Completed partial implementation of RIP using the auto-generation engine. This was enough to start generating code for a subset of RIP
- Finished initial coding rules
- Finished capability to generate code based on coding ontology
- Finished prototyping coding rules and code templates using the Echo protocol

Task 3: Selection of CWE Database weaknesses

- Table 1 captures the 26 relevant weaknesses identified in Phase 1 of the project; bolded entries indicate the 10 weaknesses focused on for Phase 2.

- Rules in structured English were developed for each weakness selected in Phase 1 to detect when source code is subject to the weakness and additional rules describing how to avoid the weakness, which are used to write the coding rules (Task 4).
- A static analysis tool was implemented for each weakness that checks the generated source code to verify it complies with the mitigation rules for those weaknesses.

Task 4: Coding Rules Development and Static Analysis Tools for Coding Rules

The project team created static analysis tools to verify that the generated software complied with the coding rules. The following static analyzers were developed to verify that the generated software complies with the coding rules. These tools were applied to the current AGNES software:

- Potentially Dangerous Function Call Checker (CWE-676) – checks that no calls are made to potentially dangerous functions as defined by "Security Development Lifecycle (SDL) Banned Function Calls" (http://msdn.microsoft.com/en-us/library/bb288454.aspx).
- AGNES Buffer Use Checkers – ensure that the generated software would track buffer sizes and check bounds for buffer accesses. These four tools check that the generated software correctly uses the AGNES buffer implementation. Together they check compliance with the following coding rules:
    - AGNES Buffer Bounds Checker – checks calls to a buffer function is either making a request that is guaranteed to be within bounds, or has its return value used.
    - AGNES Buffer Direct Access Checker (CWE-120) – checks that there is no direct access to buffers, only access using the buffer functions.
    - CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
    - CWE-124 Buffer Underwrite ('Buffer Underflow')
    - CWE-127 Buffer Under-read
    - CWE-129 Improper Validation of Array Index
    - CWE-131 Incorrect Calculation of Buffer Size
    - CWE-805 Buffer Access with Incorrect Length Value.
- Buffer Size Calculation Checker- checks that the "sizeof" function is not applied to a pointer.
- Pointer Arithmetic Checker – checks that no pointer arithmetic (including array subscripting) is performed.
- Integer Overflow Checker (CWE-190) – checks that no arithmetic operations are performed that may result in a value that exceeds the range of allowable values.
- Exceptional Condition Handling Checker (CWE-754) – checks that each call to an integer returning function either is guaranteed to not return zero or has its return value used.
- Variable Initialization Checker (CWE-456) – checks that no variable is used before it has been assigned a value.
- NULL pointer dereference (CWE 476)

- Untrusted Pointer Dereference Checker (CWE-822) – checks that pointer values obtained from outside sources, such as environment variables or user input, are not dereferenced.

Task 5: XML RFC to RDF

The team developed a tool using Cogent-XML, called xmlrfc2rdf, that accepts RFC documents in XML format and generates RDF documents (in Terse RDF Triple Language (Turtle) format) that can be loaded into the AGNES RDF triple store. This allows for the direct reference of RFC elements in the protocol descriptions.

Task 6: Implementation of the RIP Protocols

RFC specifications and existing RIP implementations in the Zebra/Quagga framework were examined enabling the specification of the XML Schema Definition (XSD) of the RIP packet and the RIP router table entry.

- Investigated Zebra/Quagga as the basis of a RIP implementation
- Developed code templates for control strategy, timers and kernel interface
- Updated structures and algorithms for network element code generation
- Developed reactive programming control strategy supporting TCP/UDP transmission, timers, kernel interface, etc.
- Developed message processing, timer processing and global timer maintenance code
- Developed router table kernel interface and router table processing code
- Developed and tested hand coded version of RIP based on ontologies, coding rules and code templates for functional testing and analysis

Task 7: Investigation of Simulation/Emulation Environment

- Opted to use GNS3 for our simulation/emulation test environment over EMANE
- Set up GNS3 virtual test environment
- Started functional testing of hand generated code from above in GNS3

Task 8: Evaluation of AGNES Generated Code

- Finished instrumenting Open Source RIP v2 code for analysis of hand coded RIP implementation
- Performed static analysis of hand generated RIP code

Additional Phase 2 Accomplishments

In addition to the above accomplishments, the following was also performed:

- A detailed progress report was delivered to ONR in February of 2017
- Presented an overview of the project and accomplishments at the ONR Code 30 C4/EW/Cyber Program Review on 20 April 2017

- Presented a paper on AGNES at the 12th Annual Cyber and Information Security Research (CISR) Conference on 6 April 2017

4.3 Phase 3 Results

Due to funding shortfalls, ONR was unable to fund Phase 3. Between July 2017 and February 2018, the project team used the remainder of FY 2017 funding and completed the auto code generator framework, completed the initial capability to auto-generate RIPv2 code, and demonstrated this capability in January 2018 to the ONR program manager, Dr. Waleed Barnawi. The project team also integrated the AGNES-generated RIP code into a hardware solution at the end of the 2017.

The revised objective for Phase 3 was to finish developing the automatic code generator framework in order to demonstrate the capability needed to generate network element code that follows coding rules and avoids CWE weaknesses. This objective was achieved and a demonstration was conducted for the ONR program manager. The code was executed in Graphical Network Simulator-3 (GNS3) and in a hardware router.

The following subsections list the major tasks performed in support of the goal of automatically generating a RIPv2 implementation from known CWE weaknesses, testing the code in a network simulation environment, analyzing the code for weaknesses and vulnerabilities, and running the generated code on a hardware device.

Task 1: Core Representation Framework and Ontologies

This task consisted of development of the automatic code generator software framework and ontologies that the code generator reads to produce code. The knowledge base contains the XML representation of the RFCs, software weaknesses, coding rules and the specification of the generated network element.

The Core Representation Framework and Ontologies task was completed in a prior program year on AGNES. Minor updates and bug fixes to the framework for network ontology, the coding rules and the network elements were completed during this past year.

Testing:

- Performed functional testing of hand generated code in Graphical Network Simulator-3 (GNS3).
- Ran the InterOperability Laboratory RIPv2 Operations Test Suite on the generated code and fixed all errors.
- Successfully ran hand-written code in hardware device.
- Performed weakness analysis of initial generated code.

Task 2: Auto-Generation Engine

The auto code generation engine was developed prior to AGNES and is a stable piece of software. The original Cogent code generator used XML as input. The following updates were performed on the code generator during Phase 3:

- Finished code templates for code generation.
- Completed hand-coded version of RIP for testing and analysis.
- Finished initial version of code generator based on XML.
- Generated RIPv2 code which executed in the GNS3 network simulation environment.

Figure 15 shows a Quad Chart Summary of the AGNES Code Generation effort.

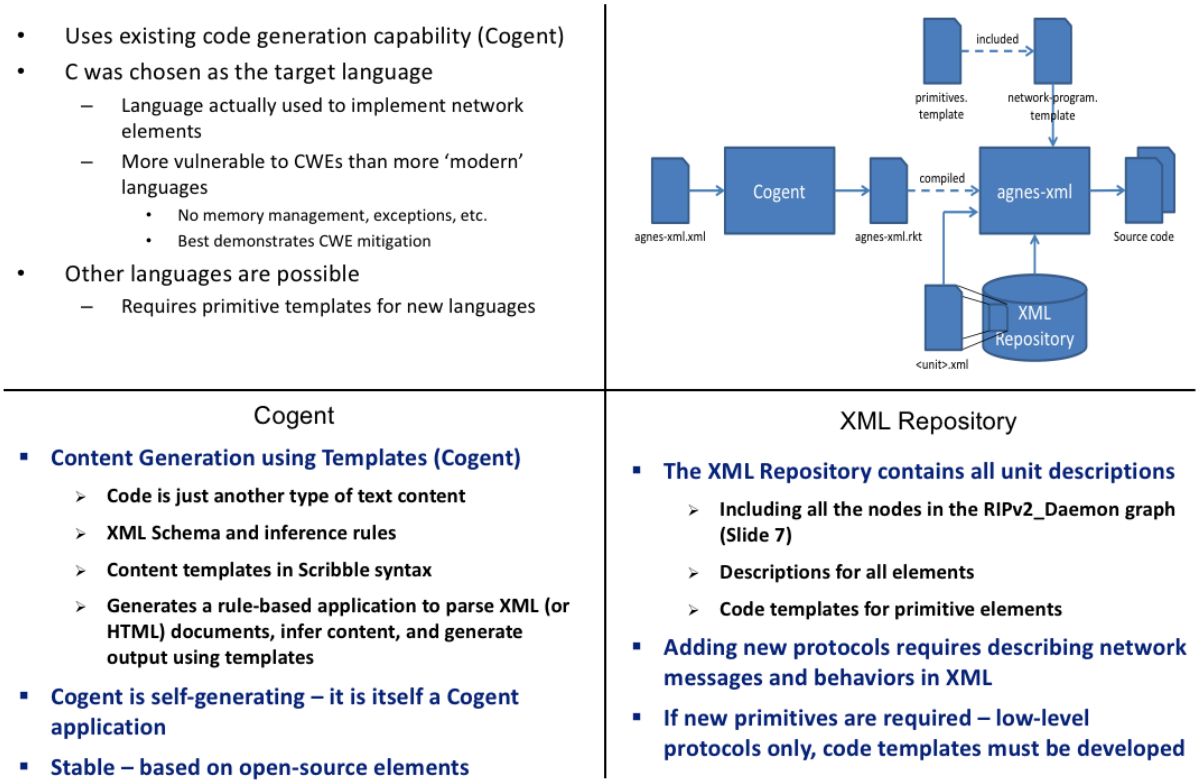

Figure 15: AGNES Code Generation.

Task 3: Selection of CWE Database Weaknesses

Known weaknesses were selected from the MITRE CWE database relevant to network element software in the previous year. The most frequently occurring network element software weaknesses were identified and a subset was selected.

- The Software Weakness Ontology was updated with additional SWEs.

Task 4: Coding Rules Development and Static Analysis Tools for Coding Rules

Coding rules for basic network element functions and for avoiding the selected weaknesses were developed. The project team created static analysis tools to verify that the generated software complies with the coding rules. The following static analyzers were developed to verify that the generated software complies with the coding rules. These tools were applied to the current AGNES generated software:

- Updated the Potentially Dangerous Function Call Checker (CWE-676) – checks that no calls are made to potentially dangerous functions as defined by "Security Development Lifecycle (SDL) Banned Function Calls" (http://msdn.microsoft.com/en-us/library/bb288454.aspx).
- Updated the AGNES Buffer Use Checkers – which ensure that the generated software would track buffer sizes and check bounds for buffer accesses. These four tools check that the generated software correctly uses the AGNES buffer implementation. Together they check compliance with the following coding rules:
    - AGNES Buffer Bounds Checker – checks to see if calls to a buffer function are either making a request that is guaranteed to be within bounds, or has its return value used.
    - AGNES Buffer Direct Access Checker (CWE-120) – checks that there is no direct access to buffers, only access using the buffer functions.
    - CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow').
    - CWE-124 Buffer Underwrite ('Buffer Underflow').
    - CWE-127 Buffer Under-read.
    - CWE-129 Improper Validation of Array Index.
    - CWE-131 Incorrect Calculation of Buffer Size.
    - CWE-805 Buffer Access with Incorrect Length Value.
- Exceptional Condition Handling Checker (CWE-754) – checks that each call to an integer returning function either is guaranteed to not return zero or has its return value used. This was started but not completed due to funding.

Task 5: XML RFC to RDF

This task was completed in Phase 2.

Task 6: Implementation of the RIP Protocols

This task was completed in Phase 1.

Task 7: Investigation of Simulation/Emulation Environment

In the first year, the project team chose GNS3 as the network simulation environment to test the AGNES generated code. In Phase 2 the project team:

- Set up the GNS3 virtual test environment to test RIPv2 generated AGNES code.
- Started functional testing of hand generated code in GNS3.
- Performed bug fixes based on testing.
- Finished testing of AGNES generated code for a RIPv2 router communicating with a CISCO router in GNS3.

Task 8: Evaluation of AGNES Generated Code

- Completed static analysis of hand generated RIP code.
- Performed static code analysis of hand generated code
- The static analyzer generated 72 errors. Code was updated to remove most of the warnings. The remaining warnings were addressed by whitelisting the functions.
- Documented errors found for updating of code generator.
- Found integer overflow analysis of Clang (static analyzer) needs improvement and looked for another checker.
- Performed static code analysis of initial code generator version
- Results of the initial code generated version were similar to the static analysis of the hand-generated code.
- Results were documented for updating of the code generator.
- Documented errors for updates.
- Started symbolic analysis of generated code using Klee
- Reached level of 50% of code to be analyzed by Klee, and was in the process of updating to improve coverage when funding was exhausted.
- Started investigating code contracts to improve static analysis.

Figure 16 illustrates the approach to AGNES Coding Rule Compliance Checking.

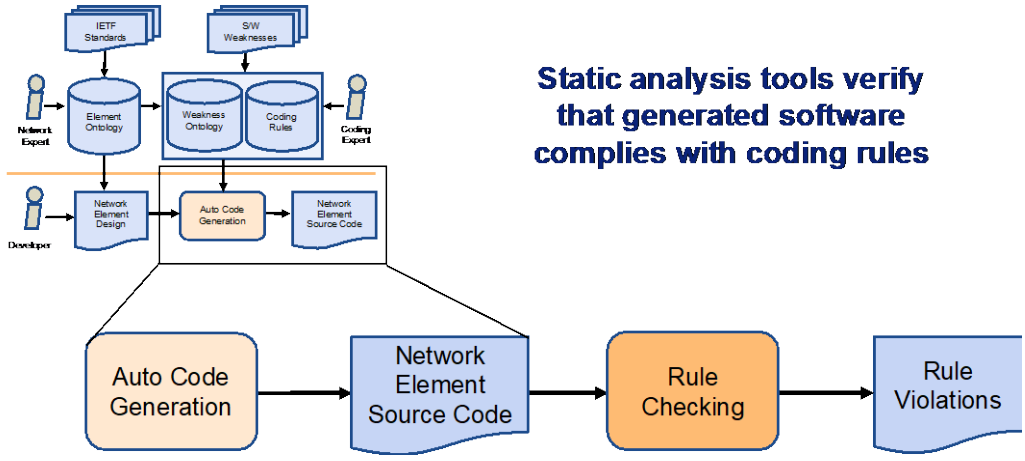

Figure 16: AGNES Coding Rule Compliance Checking

Figure 17 illustrates the approach used on AGNES for Vulnerability Analysis.

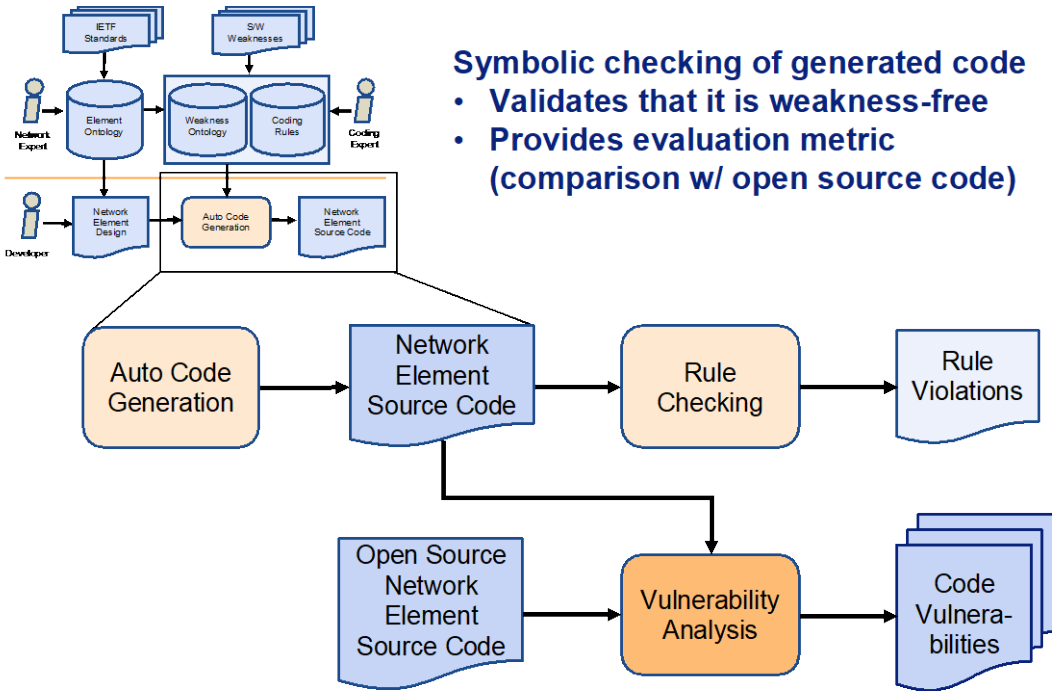

Figure 17: AGNES Vulnerability Analysis.

In addition to the above, a plan for the third and final year was developed and was partially executed using remaining funding. This plan included the following:

- Completed: Perform functional testing and analysis of the RIPv2 generated code (see Figure 18 and above tasks). This was completed before funding expired.
- Completed: Gave Demo of RIVv2 to ONR PM. This was given in January 2018.
- NOT STARTED: Extend current RIPv2
- Researched RFC 4822 to add authentication and cryptography to the RIPv2 implementation.
- Started adding authentication and cryptography but was unable to finish due to funding limitations.
- NOT STARTED: Static analysis research
- Researched code contracts for code generator to increase accuracy of static analysis of code.
- Researched integer overflow static analysis tool.
- Publish Results: Pending.

In addition to the publication at CISRC 2017, results were disseminated through a poster that was presented at the High Confidence Software and Systems Conference in Annapolis. The following is the citation for the paper:

J. Myers, R. McDowell, C. Rouff, D. Williams and D. Bennett. Static Analysis of Programmatically Generated Network Software: Challenges and Synergies. High Confidence Software and Systems Conference. May 7-9, 2018.

Additional dissemination of results:

- ONR Code 30 C4/EW/Cyber Program Review on April 20, 2017.
- Technical report submitted to ONR in February 2017.
- Was contacted by Alwyn Goodloe from NASA Langley at the HCSS conference about the AGNES buffer code. We shared the source code with NASA per ONR direction.

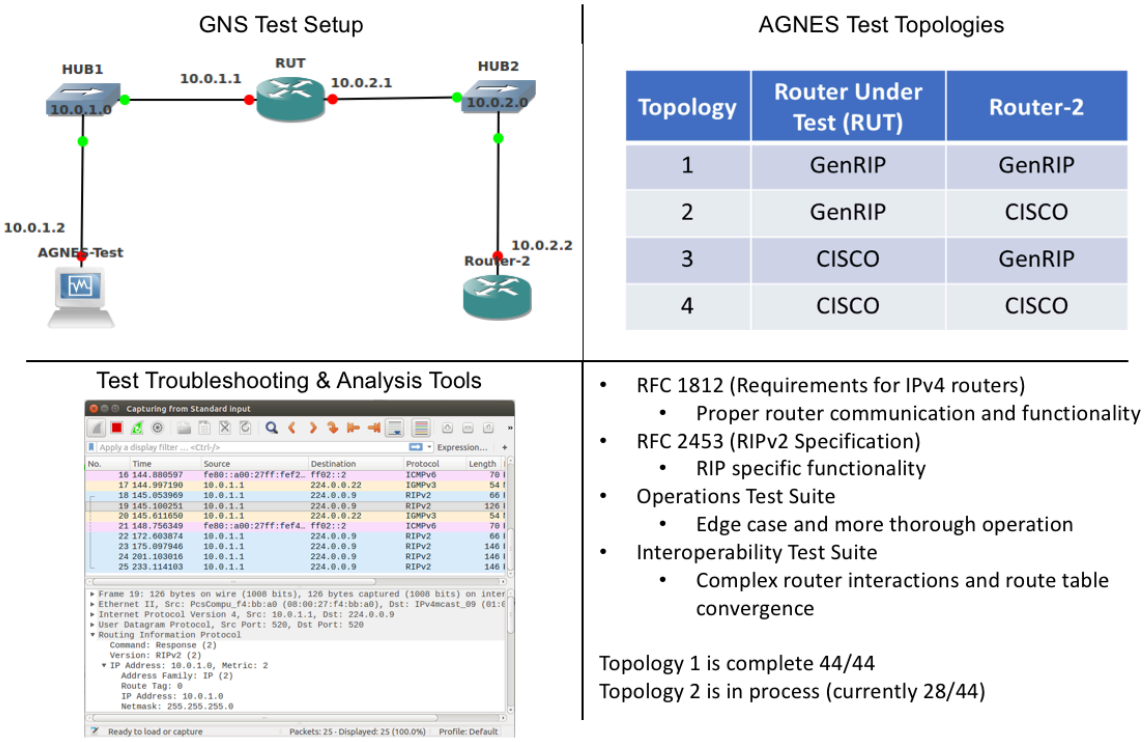

Figure 18: AGNES Testing.

5 CONCLUSION

The Automatic Generation of Network Element Software (AGNES) project made significant progress during Phase 1 towards automatically generating weakness free software for network elements. Phase 1 developed the core elements of AGNES. This included the selection of weaknesses from the CWE database, the development of the core representation frameworks, the development of the core of the auto-generation engine, the development of the coding rules, the partial auto-generation of RIPD, static analysis tools developed, and the selection of a virtual testing environment.

The above accomplishments resulted in the development of the core functionality of AGNES. Phase 2 used the Phase 1 functionality to automatically generate RIPD software for a network

router. A virtual and hardware environment were used to demonstrate that the AGNES generated software ran correctly and performed as well as hand-written software. Phase 3 planned to use results from the previous phases to support a Navy-relevant demonstration of automatically generating RIP elements into a tactical radio using the soldier radio waveform (SRW). The final results of the project demonstrated that weakness-free network element software which runs as fast as hand-coded software can be automatically generated and produced faster than hand-coded software.

6 RECOMMENDATIONS

The AGNES project validated that network element software that is free from known weaknesses can be automatically generated. AGNES generated code for RIPv2; however, software for other network protocols could readily be generated as well.

If follow-on development were pursued, it is recommended that a Navy network which runs RIP be identified as a test platform. Code from AGNES could be generated for a router and inserted into the network; network performance using generated code could then be assessed in an operational environment. After positive results are attained, additional CWEs that are relevant to RIP could be added to make the generated code even more secure and resistant to attacks. Baseline capabilities could then be extrapolated by using AGNES to generate mission critical software.

7 PUBLICATIONS

The following are two publications that were written during the project. One or more additional publications are being written.

> Christopher A. Rouff, Douglas Williams, Qinqing Zhang, Daniel Bennett, Raymond McDowell, Anthony Nowicki, Aaron Pendergrass, Daniel Anderson, Robert Douglass, Bradley T. Dufresne, Jonathan T. Pham. Automatic Generation of Network Element Software (AGNES). Cyber and Information Security Research Conference (CISRC 2017). Oak Ridge, TN. April 4 - 6, 2017. DOI: 10.1145/3064814.306482
>
> Jonathan Myers, Raymond McDowell, Christopher Rouff, Doug Williams and Daniel Bennett. Static Analysis of Programmatically Generated Network Software: Challenges and Synergies. High Confidence Software and Systems Conference. May 7-9, 2018.

8 REFERENCES

1. R.A. Martin and S. Barnum. 2008. Common weakness enumeration (CWE) status update. Ada Letters, XXVIII (1):88–91.

2. MITRE. 2008. Common weakness enumeration (CWE). URL: http://cwe.mitre.org/data/slices/2000.html

3. C.L. Hedrick. 1988. Routing Information Protocol. Request for Comment 1058. URL: https://datatracker.ietf.org/doc/rfc1058/

4. R.A. Martin and S. Barnum. 2008. Common weakness enumeration (CWE) status update. Ada Letters, XXVIII (1):88–91.

5. http://cwe.mitre.org

6. Natalie, I., Rivera, B. and Adamson, B.: Mobile ad hoc network emulation environment, Proc. IEEE MILCOM, pp.1–6 (2009)

7. EMANE website: www.nrl.navy.mil/itd/ncs/products/emane

8. http://www.howtogeek.com/188884/symantec-says-antivirus-software-is-dead-but-what-does-that-mean-for-you/

9. http://online.wsj.com/news/article_email/SB10001424052702303417104579542140235850578-lMyQjAxMTA0MDAwNTEwNDUyWj

10. B. Myers, “User interface software tools”, ACM Transactions on Human-Computer Interaction, 2(1):64-103 (1995).

11. http://www.mathworks.com/

12. http://www.scilab.org/

13. http://www.wolfram.com/mathematica/

14. http://www.maplesoft.com/

15. K. Murray, J. Lowrance, K. Sharpe, D. Williams, K. Grembam, K. Holloman, C. Speed, and R. Tynes, “Toward Culturally Informed Option Awareness for Influence Operations with S-CAT”, 4th International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction (SBP11), Springer, vol. 6589, pp. 2-9, March 2011.

16. K. Murray, J. Lowrance, K. Sharpe, D. Williams, K. Grembam, K. Holloman, and C. Speed, “Capturing culture and effects variables using structured argumentation”, 1st International Conference on Cross-Cultural Decision Making, (2010).

17. http://www.w3.org/RDF/

18. http://www.w3.org/TR/rdf-schema/

19. http://www.w3.org/TR/owl2-overview/

20. http://www.w3.org/TR/rif-overview/

21. N. S. Evans, A. Benameur, M. C. Elder, “Large-Scale evaluation of a vulnerability analysis framework”, Proceedings of CSET’14, San Diego, CA, USA (2014).

22. D. Reifer, Industry software cost, quality, and productivity benchmarks, http://www.compaid.com/caiinternet/ezine/reifer-benchmarks.pdf (2004)

23. Coverity Static Analysis: http://www.coverity.com/products/static-analysis.html

24. Coverity website: https://www.coverity.com/

25. Coverity tool license website: http://www.coverity.com/html/end_user_licenses.html

26. HP Fortify Source Code Analyzer: https://www.fortify.com/products/hpfssc/source-code-analyzer.html and https://www.fortify.com/products/fortify360/source-code-analyzer.html

27. HP Fortify Source Code Analyzer tool license website: https://www.fortify.com/services/fortify360-services.html

28. KlocWork Insight Comprehensive Source Code Analysis: http://www.klocwork.com/products/insight/?source=feature

29. KlocWork website: https://www.klocwork.com/

30. KlocWork tool license website: http://www.klocwork.com/support/

31. Source Forge FindBugs: Improve the quality of your code http://www.ibm.com/developerworks/java/library/j-findbug1/ - author1

32. Source Forge FindBugs: Writing custom detectors http://www.ibm.com/developerworks/java/library/j-findbug2/

33. Source Forge website: http://findbugs.sourceforge.net/

34. Source Forge FindBugs tool license website: http://www.gnu.org/copyleft/lesser.html

35. Veracode. Veracode “2010 CWE/SANS Top 25 Most Dangerous Software Errors” Detection, http://www.veracode.com/images/pdf/VERAFIED/veracode-cwesans25-detection.pdf

36. Veracode Solution for FISMA: http://www.veracode.com/images/pdf/datasheet_fisma.pdf

37. Veracode website: http://www.veracode.com/

38. Veracode tool license website: http://www.veracode.com/about/contact-us.html

39. IBM Purify paper: http://www.ibm.com/developerworks/rational/library/06/0822_satish-giridhar/

40. UNICOM/IBM Purify website: http://unicomsi.com/products/purifyplus/

41. UNICOM/IBM Purify license agreement: http://www-947.ibm.com/support/entry/portal/Overview/Software/rational/Rational_Purify?S_CMP=rnav

42. Binary Analysis Platform tool website: http://security.ece.cmu.edu/index.html

43. Parasoft Insure++: Runtime Analysis and Memory Error Detection for C and C++

44. Parasoft website: http://www.parasoft.com/jsp/products/insure.jsp?itemId=63

45. Parasoft Insure++ license agreement website: http://www.parasoft.com/jsp/products/insure.jsp?itemId=63

46. Nicholas Nethercote, Julian Seward. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation (http://valgrind.org/docs/valgrind2007.pdf)

47. Valgrind research papers: http://valgrind.org/docs/pubs.html

48. Valgrind website: http://valgrind.org/

49. http://www.gotomeeting.com/online/

50. Pascal Hitzler, Markus Krötzch, Bijan Parsia, Peter F. Patel-Schneider, Sebastian Rudolph. OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation, 11 December 2012. The latest version is available at http://www.w3.org/TR/owl-primer.

51. Guus Schreiber, Yves Raimond. RDF 1.1 Primer. W3C Working Group Note, 25, February 2014. The latest version is available at https://www.w3.org/TR/rdf11-primer/.

52. Dan Brickley, R.V. Guha. RDF Schema 1.1. W3C Recommendation, 25 February 2014. The latest version is available at https://www.w3.org/TR/rdf-schema/.

53. RDF4J website: http://rdf4j.org/

54. Amazon Elastic Code Compute (EC2): aws.amazon.com/ec2/

55. RDF 1.1 Turtle - Terse RDF Triple Language. World Wide Web Consortium (W3C). 25 February 2014. www.w3.org/TR/turtle/

56. Amazon Web Services. https://aws.amazon.com

9 ACRONYMS

ACG	Auto-Code Generator
AGNES	Automatic Generation of Network Element Software
AWS	Amazon Web Services
CISR	Cyber and Information Security Research
COGENT	Content Generation from Templates
CONOPS	Concept of Operations
CWE	Common Weakness Enumeration
RDF	Resource Description Framework
EMANE	Extendable Mobile Ad-hoc Network Emulator
GNS3	Graphic Network Simulator-3
IETF	Internet Engineering Task Force
JHU/APL	The Johns Hopkins University Applied Physics Laboratory
LTS	Long Term Support
ONR	Office of Naval Research
RDF	Resource Description Framework
RFC	Request for Comments
RIP	Router Information Protocol
RTNetlink	Routing Table Netlink
SCM	Source Code Management
SDL	Security Development Lifecycle
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XML	Extensible Markup Language

XSD	XML Schema Definition