Research Plan
PROTOS Protocol Genome Project
The Search for Information Technology DNA
DRAFT
$Revision: 1.33$

Aki Helin
Joachim Viide
Timo Hyart
Marko Laakso
Juha Röning

19th November 2004

Abstract

While we depend on computer networks in our daily lives, the programs we use communicate via a plethora of often incorrectly implemented protocols. Our previous experiments suggest, that there are structural similarities - protocol genes - shared between currently used protocols. The theoretical tools for finding these genes are out there - both structure extraction and protocols have gotten their fair share of research. However, they have yet to be combined to find genes from the sea of protocols. The purpose of this project is to develop a methodology to find the genes and present protocols using them. The collected gene pool gives a fresh perspective into many traffic related problems, and a way to look at the 'big picture' giving a radical perspective to the protocol pandemonium.

Contents

1 Research plan

1.1 Introduction

"After a brisk, promising start, the dark ages fell upon the Internet. A number of people were solving the same set of problems, usually with poor results. This caused a growing pile of less than optimal standards and programmes. One particularly troublesome area was that of protocols. While the rest of humanity was starting to think computers might do a pretty good job at doing something useful, programmers still found it fashionable to write their own buggy ad hoc parsers to solve problems that already had been solved. Essentially they were reinventing the same wheels over and over again.

Meanwhile people called 'biologists' had started developing efficient methods for digging out bits of DNA doing similar things. They had decided to call these similar bits 'genes'. The genes they found made it easy (well, at least easier) to compare species, find out DNA sequences causing different weaknesses, and create horrible mutated organisms.

Some programmers, too, had tried to compare protocols, find out their weak spots, and create horrible mutated PDUs (protocol data units). So, easing the tasks by mimicking the biologists - figuring out similarities between protocols and calling them genes - wasn't too far fetched. Still nobody did so. For 40 years, fear, uncertainty and doubt prevailed."[1]

Intuition based on our previous experiments with creating horrible mutated PDUs suggests, that there are structural similarities in currently used protocols.[1] When looking at a large set of protocol definitions, abstracting out the similarities makes the overall definitions simpler without losing information content, not unlike compression. Although possibility of using a scheme similar to lossy compression should not be overlooked. When this process is started from definitions in a concise formal form, and iterated many times there will be a set of primitives capable of describing the syntax of the initial protocols.

We will regard these contained structures as 'genes' of the protocols. This makes the genome of a protocol a highly domain-specific description of the protocol's structural content. From now on protocol descriptions formed from these genes will be called 'g-expressions'.

Basic tools for finding the genes are out there. Both structure extraction and protocols have gotten their fair share of research, but they have yet to be combined to find similarities from several protocols. Previous studies on protocol analysis have concentrated mainly on individual protocols and their state models. Prior work on structural analysis has concentrated on extracting only syntax from datasets, ignoring all the semantics. Moreover, the focus has been on developing general, all-around solutions for all problem domains (eg. extracting structure from natural language or music).[2] It is a noble pursuit, but not a particularily feasible one for our envisioned application scenarios, if we prefer living long enough to reap the results. In our strategy, specialising in protocols helps to keep the system simple enough to be practical. At the same time,

[1] Pers. comm. of the project group in the initial project planning phase.

analysing a number of similar protocols in parallel helps to keep the project going - at each initial step there must be at least one clear structure to handle.

The gene analogy can be taken further once the gene pool is sufficiently well populated. It is important to grasp the limitations of the analogy, but it is not mere wordplay either, and can give fresh perspective to the protocol analysis. Are there lethal genes - gene sequences particularly prone to implementation errors? Can mutant PDUs be separated from valid traffic? Is it possible to found related protocols by examining their genes? And why not use results from the actual gene research field, eg. algorithms for DNA sequencing, matching and classification?

The purpose of this project is to develop a methodology to extract protocol genes and present protocols using them. The collected gene pool gives a way to look at the 'big picture' of the sea of current protocols. As the work proceeds the methodology will be abstracted and applied iteratively, each iteration building on the experiences from previous ones. Some of the work will be carried out manually, by analysing large amounts of standards and specifications, but also by prototyping activity that will apply the theory into practical scenarios, such as:

- Identifying weak genes, i.e. constructs prone to implementation flaws.
- Automatic structural mutation (protocol mutation engine).
- Assessment of structural mutation coverage.
- Identifying mutations (a form of structural IDS).
- Structure-aided 'diff' (comparison of two instances).
- Parser contracts ("I am hereby designed to handle this, protect me from everything else.") and canonisation to a standard form.

This approach helps us verify that the results will have actual practical value, from our slightly information security inclined perspective.

1.2 Aims

The Protocol Genome Project aims to:

- Develop a method to identify protocol genes from protocol specifications and PDUs and collect them and their characteristics into a database.
- Develop a notation for presenting PDUs as sequences of these building blocks.
- Build tools for the tasks mentioned above, eg. genome extractor for both known and unknown protocols.

These aims appear to be along the same lines with Cisco's Critical Infrastructure Assurance Group (CIAG) project wishlist item 'Automated Arbitrary Protocol Reverse Engineering':

“Develop a general-purpose tool for analyzing and describing arbitrary protocols based on a packet capture file. This tool would provide a probable description of any protocol with a particular benefit in describing previously unknown protocols.”[3]

1.3 Risks

- The worst case scenario is that extraction of a useful gene pool fails. If there are not enough similarities in protocols, the gene pool might grow too much and the benefits would be lost. Should this be the case, the focus of the project would probably shift towards heuristic structure inference techniques. However, at least one of us believes that the human kind has only been able to create a limited number of protocol genes.
- A team without proper background could easily spend a few years reinventing parts of formal language theory and basic parsing techniques. The classic and contemporary works will be constantly reviewed during the project.
- Even is succesful, the project might remain as a curiosity if the g-expressions and their underlying formalism become too complex. We will try to keep the g-expressions and underlying system simple.
- There might not be enough resources (CPU, personnel etc.) to create g-expressions for sufficiently many protocols. By keeping the down complexity of g-expressions and underlying system the risk can be mitigated.

1.4 Preliminary Work

There are many possibilites as to how and what kind of information the genes carry. Naturally the definition of a g-expression depends on the definition of genes, so we have intentionally been a bit vague on that. We are going to write small prototypes for different models to test their properties in practice, before nailing down a specific choice. In addition to the primary project results we would like to release a buch of these and other small applications with a brief documentation.

In our preliminary work we have identified two essential problems the genes should solve. The first problem is leakage of semantics to syntax. Protocols could fairly easily be processed using existing tools for handling syntactic problems, like checking whether a packet follows a simple grammar, if there were not certain structures that are usually found from the semantics side, like evaluating 4 bytes as a signed little endian integer and using it to define the length of something.

Another problem is how to handle association. When a gene is added to a g-expression, it’s surroundings may affect it’s behavior. We have separated different gene models into 0-, 1- and n-systems based on the amount of implicit or explicit association. In the 0-system a gene always works the same, like a piece of regular expression. The 1-system allows a gene to look what it’s immediate neighbours are and the n-system allows each gene to use any information about the whole g-expression. Functional genes are a variation of a n-system. We are also planning a logic based prototype to model a 0-system.

There are a number of approaches that could be applied in parallel to manual gene sequencing. Methods for finding structural similarities have been developed in areas such as compression, gene sequencing, algorithmic information theory and artificial intelligence. Formal grammars, which have been used extensively in bioinformatics [5], natural language processing[6] and computer science, would be one interesting base for building gene sequences, since they are already used for similar tasks in the real world, and there would be a natural way to extend the system to handle probabilistic analysis and constraints while the system still remains more human understandable than a corresponding neural network. Training of grammars and other systems manually and by supervised machine learning could prove to be useful for identification purposes.

Another useful track might be using various simple figerprinting methods for optimizations. Bit and byte distributions, magic numbers in payloads, cheap submatches and other similar techniques could together prove to be an efficient basis for protocol identification even by themselves.

1.4.1 A Case Study: Infer

During the preliminary stages of the project we built a prototype structure inference prototype program called infer, which is essentially a naive implementation of the Sequitur[2] grammar inference algorithm. It reads a sequence of bytes from a file and generates a grammar that describes that byte sequence. Below is a small example transcript of a Scheme session using the program. The file small.txt is loaded and analyzed. After processing a description of the structure is printed. In this case the grammar states that the whole text (root node) consists of two equivalent pieces of text, which consist of two lines where either abba or poro occurs twice.

```
> (infer "small.txt")
inferring structure from text {
   abbaabba
   poroporo
   abbaabba
   poroporo
}
processing: ************ done.
inferred structure:
   $root = $3 $3
   $1 = a b b a
   $2 = p o r o
   $3 = $1 $1 \n $2 $2 \n
```

1.4.2 Another Case Study: Regexper

Another structure inference program developed during pre-project phase is called Regexper. Regexper simply takes a set of packets of arbitrary protocol data and creates a sequence of regular expressions - regexps - that match all the packets in the set. Any incoming packet belonging to the same protocol should also be matched by the expressions, while each expression is more specific than a previous one. That is, the expression contains less wild card elements (arbitrary data of arbitrary length) and more elements of some known quality (constant values or arbitrary data with constant length). Thus, the first expressions in the sequence match more packets not belonging to the protocol than the latter ones.

The created regular expression shows rudimentary structure in the data: constant headers, data delimiters, magic values etc. This can be used as a good starting point in spotting some more advanced structural elements, the likes of checksums or length fields, as they usually have a defined length.

As a file format can be thought as a protocol, files sharing the same format have been good test input for the program. Below is the first regular expression the program dug from 23 GIF image files:

```
GIF87a.{1}\^C.{1}\^B.{1}\000\000\000\000\000\ß\ß\ß.*\,\000\000\000\000.{1}\^C.{1
}\^B\000\^B\þ\<8C>\<8F>\l’\Ë\í\^O\č\<9C>\t’\Ú\<8B>\ş\Þ\ij\û\^O\<86>\âH\<96>\æ\<89>\
ę\ê\Ê\ű\î\^K.{2}L\Œ\ö\<8D>\ç\ú\Î\œ\þ\^O\^L\
\<87>.{3}\<88>L\*\<97>\Ì\ę\ó\
\<8D>J\ğ\Ô.{2}\<8A>\Íj\ů\Ü\ő\œ\^K\^N\<8B>.*\Œ.*\
ì\ű.{3}\Ë\ç\ô\ž\ý\<8E>\Ï.{3}\¿\ß\^O\^X\(8.*\È.*\Ø.*\è.*\)9IYi.*\Ù.*\é\ù.*\
.*\:JZjz.*\<8A>.*\ž\Ê\Ú\ê\ú.*\
.*\;K\[k\{.*\ż.*\Ë\Û\ë.{4}\<L\\.*\ň.*\ij\Ì\Ü.*\ì.*\-\=M.{2}\}\<8D>\<9D>\■.*\¡.*\í
\ý\^M\^^.{3}\^n\~\<8E>.*\<9E>.*\Î.*\SS.*\þ.*\ż.*\þ.*\‘.*\þ.*\Î.*\þ.*\<.*\þ.*\<99>
.*\þ.*\ě.*\þ.*\þ.*\ë.*\Û.*\þ.*\Ï.*\þ.*\^?.*\^?.*\<83>.*\<85>.*\<87>.*\<88>.*\
<8A>.*\<8C>.*\Ĺ.*\þ.*\<95>.*\<89>.*\<9A>.*\<9A>.*\þ.*\ą.*\d’.*\þ.*\ł.*\ł.*\ň.*\þ.
*K.*\ż.*\þ.*\¿.*\£.*\Â.*\þ.*\<8C>.*\Ì.*\Ì.*\Ë.*\þ.*\<9D>.*\Í.*\Ý.*\þ.*\Ý.*\þ.*\é
.*\î.*\þ.*\õ.*\þ.*\000.*\þ.*\).*\þ.*\Â.*\þ.*6.*\þ.*\].*\þ.*\".*\þ.*\d’.*\þ.*V.*\þ
.*\þ.*\ä.*\þ.*\þ.*\Ï.*0.*\þ.*\^Q.*\^Q.*\þ.*R.*\ò.*\/.*\þ.*S.*\?.*\^T.*\þ.*U.*u.*
\þ.*l.*6.*\þ.*w.*7.*\þ.*\<8F>.*Y.*\ź.*y.*\:.*\;.*\ij.*\ü.*\ü.*\ü.*\=.*\;.*
```

An eye trained to the ways of regexps may see that even the first iteration finds some structural information: the magic identifier string (“GIF87a”) and most of the header. Coincidentally - and a bit ironically - the GIFs used are wood spectrum images which our neighbouring group working on image pattern recognition have used in completely different viewpoint. The resulting regular expression can be used in its current form for filtering away most GIFs that are not wood spectrum images from the same source. The system currently finds expressions that exactly match the training data. Adding probabilities would result in a system that closely resembles hidden markov models.

1.4.3 A Prototype: Functional genes

Functional genes are a prototype to test the n-systems. We have an implementation of the system that has been tested in practice to handle traffic and other content processing. The system is described in a separate document.

1.5 Tasks

The project consists of four main tasks, engaged partially in parallel. The first task, *State-of-the-Art*, consists of all supporting research of the field and providing the theoretical basis of the research. In the second task, *Prototyping*, the tools for other tasks are iteratively improved. Improvements are based on the feedback from *State-of-the-Art* and *Testing*. In the third task, *Testing*, the methodology is developed and experimented. *Management* takes care of day-to-day project management tasks and coordinates documentation.

1.5.1 State of the Art

The main purpose of this task is keeping up to date with current research on the field, and to support the other tasks by providing them background information. One intriguing prospect is researching methods developed originally for gene research and hijacking them to our context. This task also acts as a channel to distill the gems of the research to the outside world.

1.5.2 Prototyping

In this task, both the theoretical protocol gene model is developed and the prototypes of tools aiding the research are designed and implemented, based on the theory from *State-of-the-Art* and feedback from *Testing*. Commercial off the shelf components are used to build the infrastructure. The prototype tools will be used in *Testing* task to verify the developed methodology in practice.

The manual protocol gene pool inference follows these steps:

1. Dissect an initial set of protocols that have practical value and are easy to analyse. Basic IP protocols or especially file formats may turn out to be useful for bootstrapping the gene pool.
2. Derive a set of primitives/genes.
3. Repeat the first step with new knowledge, until a sufficient gene bucket is obtained.
4. Extend the protocol set to several simple protocols. Apply and extend gene pool as needed. Be prepared to accept some sort of enumerable/parameterised genes.
5. Same as 4. but with more complex protocols. Support some more structural changes. Call the gene descriptions g-expressions for historical compatibility.
6.
   - Examine the properties of the language and its expressions.
   - Examine protocol family connections in terms of their g-expressions, preferably graphically.
   - Examine the possibilities of protocol identification given g-expressions.
   - Examine the relationships with common vulnerabilities and applications to making automated structural mutations.
   - Examine possibilities of deriving accepting/filtering programs from g-expressions.

1.5.3 Testing

In this task, theory from *State-of-the-Art* and tools from *Prototyping* are applied to practice. Conclusions regarding the usability of the methodology are drawn. Feedback is provided mainly for *Prototyping*.

1.5.4 Management

This task includes traditional project management and document co-ordination.

1.6 Schedule

The project starts tentatively on 2004-07-01 and lasts for three years. An iterative process model will be utilised. In this kind of process, draft deliverables are produced reasonably fast and they are refined through whole project. This model has been used successfully in the context of the PROTOS project since 1998.

1.7 Cost Estimate and Funding

The cost estimate and their distribution presented in Table 1 is based on known salaries and calculated personnel resources. The project takes three years, total of 108 (3x36) man-months. A full three year commitment from all funding partners is crucial for the successful completion of the project. This is because we expect results to be moderate in the beginning and to improve over each iteration.

Cost	2004	2005	2006	2007	**Total**
Personnel salaries, 36mm/year	45000	90000	90000	45000	270000
Side expenses (multiplier 2.00)	45000	90000	90000	45000	270000
Equipment and travels	10000	20000	20000	10000	60000
Total	100000	200000	200000	100000	600000

Table 1: Cost estimate (euros) 2004-2007

The side costs include social security benefits and other expenses for the university. The total personnel expenses (salaries plus side expenses) are based on the salaries multiplied by a multiplier the university assigns to the project. Multiplier 2.00 assumes the university considers this project as basic research. The final judgement will be made based on the restrictions set by the funding parties, the rule of the thumb being that stricter restrictions mean a greater multiplier.

Some of the analysis related to the research can be expected to be quite CPU intensive. The costs of equipment and travels are calculated to be 10% of the total project budget.

Initial funding negotiations with UK National Infrastructure Security Co-ordination Centre (NISCC)[4] are currently underway, Cisco Critical Infrastructure Assurance Group (CIAG) has been contacted and Microsoft Research will be contacted shortly.

Project personnel consists of the following people:

- Juha Röning, Ph.D., University of Oulu (responsible director)
- Aki Helin, University of Oulu (project manager)
- Timo Hyart, University of Oulu
- Joachim Viide, University of Oulu

1.8 Background Information

1.8.1 Computer Engineering Laboratory

The Oulu University Computer Engineering Laboratory is responsible for education in computer engineering, embedded systems and software engineering. The curriculum of the laboratory includes basic courses in computer and software engineering and advanced courses on operating systems, embedded system development, computer architectures, quality engineering, real-time object-oriented programming and telecommunication software.

Research of the laboratory is carried by the Intelligent Systems Group (ISG) and concentrates on mobile and context-aware systems, data mining methods and secure programming. The goal of the research is to develop both industrial applications and components for an intelligent environment that gives versatile services for its inhabitants. The application areas are diverse, including among others: health monitoring, modeling of the steel making process, personal robots, mobility aids for the elderly, smart living room, and security testing of programs.

1.8.2 OUSPG

Inside the Intelligent Systems Group, the Oulu University Secure Programming Group (OUSPG) has kept its focus on implementation level security issues and software security testing. Software implementation may introduce potential for unanticipated and undesired program behaviour, e.g. an intruder can exploit the vulnerability to compromise the computer system. The group has researched different approaches to testing implementations of protocols using black-box (i.e. functional) testing methods in PROTOS project, the vulnerability process work, and systematic methodologies to identify information security related vulnerabilities in a complex multi-modal network scenario in Frontier project.

References

[1] PROTOS - Security Testing of Protocol Implementations. Oulu University Security Programming Group (OUSPG) PROTOS project URL: http://www.ee.oulu.fi/research/ouspg/protos/

[2] Nevill-Manning, C.G. (1996) Inferring Sequential Structure. Ph.D. thesis, Department of Computer Science, University of Waikato, New Zealand.

[3] Critical Infrastructure Assurance Group (CIAG). CIAG Research Project Wishlist. URL: http://www.cisco.com/security_services/ciag/initiatives/research/wishlist.html

[4] UK National Infrastructure Security Co-ordination Centre (NISCC). URL: http://www.niscc.gov.uk/

[5] Yasubumi Sakakibara, Michael Brown, Rebecca C. Underwood, I. Saira Mian, David Haussler (1993) Stochastic Context-Free Grammars for Modeling RNA, Proceedings of the 27th Hawaii International Conference on System Sciences

[6] Miles Osborne, Ted Briscoe Learning Stochastic Categorial Grammars (1997), CoNLL97: Computational Natural Language Learning