International Refereed Journal of Engineering and Science (IRJES)
ISSN (Online) 2319-183X, (Print) 2319-1821
Volume 9, Issue 5 (May 2020), PP. 41-51

A type-erasure based DSEL for huge time-series session data analysis

Ruo Ando*, Youki Kadobayashi**, Hiroki Takakura***
** (Associate Professor by special appointment, Center for Cybersecurity Research and Development,National Institute of Informatics, Tokyo)*
*** (Professor, Laboratory for Cyber ResilienceInformation Science Division, Nara Institute of Science and Technology, Nara)*
**** (Associate Professor by special appointment, Center for Cybersecurity Research and Development,National Institute of Informatics, Tokyo)*
Corresponding Author:Ruo Ando

Abstract:*The Science Information Network (SINET) is a Japa-nese academic backbone network. SINET consists of than 800 universities and research institutions. In the operation of a huge academic backbone network, more flexible querying technology is required to cope with massive time series session data and analysis of sophisti-cated cyber attacks. In this paper, we propose a C++ im-plementation of DSEL (Domain Specific Embedded Lan-guage) for time series data using type erasure. Our meth-od can achieve the efficiency and flexibility is required for complex intrusion detection tasks in a huge academic backbone network. In our model, the function object is implemented by type erasure for constructing internal DSL for processing time-series data. Type erasure ena-bles our parser to store function pointer and function object into the same *void type with class templates. Be-sides, we apply a novel operator (=) overloading with tag dispatch for handling a branch of pointer and object. In implementing tag dispatch, a compiler time programming technique called as SFINAE (Substitution Failure Is Not An Error) is adopted. In the experiment, we have meas-ured the elapsed time in parsing and inserting IPv4 ad-dress and timestamp data format ranging from 1,000 to 50,000 lines with 24 row items. It has been turned out that proposal method can work in feasible computing time.*
Keywords*: DSEL; Type Erasure; time-series analysis; session data; Tag dispatch; SFINAE.*

Date of Submission: 15-12-2020 Date of acceptance: 30-12-2020

I. INTRODUCTION

The Science Information Network (SINET) is a Japanese academic backbone network. SINET consists of than 800 universities and research institutions. SINET serves for variety of research facilities in space science, seismology, high-energy physics, nuclear fusion, computing science. Currently SINET is being used by over 2 million users. Also SINET supports international research collaboration in academic backbone network. Since 2016, NII has been running a service of NII-SOCS (NII Security Operation Collaboration Services). Our team of NII-SOCS have de-ployed security monitoring system consists of PA-7080, Elasticsearch, Splunk, and NVidia Multi-GPU server. In this talk, we introduce our system and some operational experience of handling huge session data ranging from 400,000,000 to 800,000,000 per day. During four years of 2016-2019, We have faced many challenges in terms of number of hosts, protocol proliferation, probe place-ment technologies, and security incident response.

The PA-7000 Series leverages a scalable architecture aimed the purpose of adopting the flexible and powerful processing the key functional tasks of networking, secu-rity, and management. Session data format is shown in Table 1. No.1 - 9 is concerned about TCP/IP packet head-er. NO 19-23 is retrieved to generate statistics. Particular-ly, No.12 (application) and No.17 (category) is inspected in detail. Firewall such as PaloAlto-7080 plays an essen-tial role in network security. Also, as cyber-attacks be-come sophisticated, the language to achieve the efficiency and flexibility is required for complex intrusion detec-tion tasks.

For example, the query such as capture_time = 2020/11/*/*(No.1) , source_IP=X.Y.0.0/16 (No.5), application=web_browsing (NO12).is required to detect session data under inspection. Unfortunately, although popular intrusion detection systems has their own policy language with complicated logic requires architecture-dependent code.In this paper, we propose a DSEL (Domain Specific Embedded Language) fornetwork traffic processing thatcan be real-world time-series session data on huge academic backbone network.

Table 1 Palo Alto session data format

No	item name	value
1	capture time	2018/01/01 00:00:00.000
2	generated time	2018/01/01 00:00:00.000
3	start time	2018/01/01 00:00:00.000
4	elapsed time	3
5	source IP	xxx.xxx.xxx.xxx
6	source Port	64354
7	source country code	JP
8	destination IP	yyyy.yyyy.yyy.yyyy
9	destination Port	2939
10	dest country code	US
11	protocol	tcp
12	application	NA
13	subtype	NA
14	action	NA
15	session end reason	NA
16	repeat count	0
17	category	NA
18	packets	0
19	packets sent	0
20	packets received	0
21	bytes	0
22	bytes sent	0
23	bytes received	0
24	device name	NA

II. DOMAIN SPECIFIC LANGUAGE

In this paper, we discuss two sorts of computer language: A domain-specific language (DSL) and a general-purpose language (GPL).DSL is specialized to a particular application domain whereas GPL is broadly applicable across domains. Nowadays DSL has a wide variety ranging from widely used languages such as HTML, XML, SQL and so on. DSL is further classified by the kind of language including domain-specific markup, domain-specific modeling and domain-specific programming languages. Also, DSL is sometimes called as mini-languages in the sense that it is used by a single application.

2.1 External and internal DSL

There are two main categories of DSL: external and internal. In external DSLs, a language is parsed independently of the host GPL. CSS with regular expressions is a good example of external DSL. Internal DSLs are im-plemented with a particular form of API in a host GPL. A fluent interface [1] is often adopted in internal DSL.Mocking libraries such as JMock and Ruby on Rails are good examples of internal DSL. There has been a long tradition of usage of internal DSL, particularly in the LISP community.

Figure 1 shows the architecture of internal and external DSLs. In the view of typical compiler architecture, two kinds of DSL are common: parser, type checker and generator. However, in external DSL, the language is parsed independently of the host GPL and even independent from the rest of the program. On the contrast, internal DSL is implemented inside GPL. Giving up the flexibility of custom syntax of external DSL, internal GPL takes advantages in the learning curve and performance. Generally, internal DSL is easier to write because the language can be tailored to the idioms of the domain. In some cases, code generator part is omitted in internal DSL.

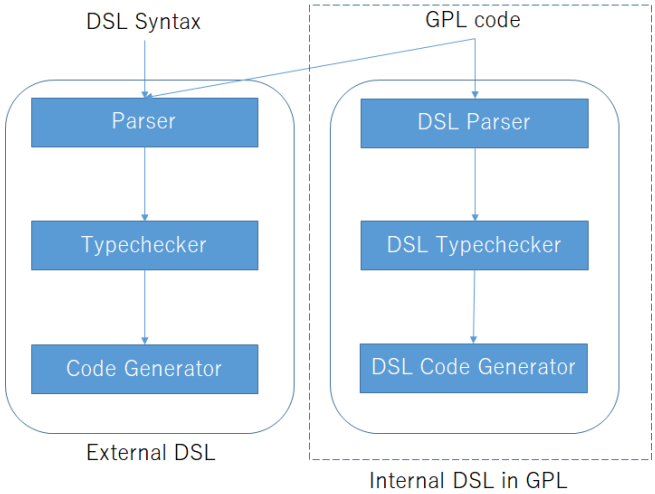

Fig. 1. Internal/External DSL

2.2 DSEL

There is yet another representation of DSL - a domain-specific embedded language (DSEL). DSEL is almost the same meaning of internal DSL. However, DESL is the language which consists of many small internal DSLs. Strictly, DESL is a concept where small internal DSL interoperate with one another. Boost Spirit and YACC could be good examples of DSEL.

III. METHODOLOGY – TEMPLATE METAPROGRAMMING

Template metaprogramming (TMP) which is also simply called as metaprogramming is a technique for generating or manipulating program code. In TMP, templates are used by a compiler to yield temporary source code before compiling. Then, the compiler merges the temporary code and the rest of the source code. The metaprogramming adopts constants, structures and functions in compile-time based on the concept of compile-time polymorphism.

3.1 Compiler-time programming

C++ templates provides a powerful computation subsystem to enable sophisticated recursive and branching logic which executes at compile-time.In Compile-time programming, three kinds of templates are available: function templates, class templates and structure templates. The key technology of compile-time programming is a specialization of a class templates.

3.1.1 Branching

We can create an explicit specialization of the lexical_cast function template to perform a branching at compiler-time. Branching is based on the types involved in the conversion. For example, the primary template of lexical_cast can be used this way:

Listing.1. Branching

```
1: std::string strPi = "1.23456789";
2: double pi = boost::lexical_cast<double>(strPi);
3: template <typename Target, typename Source>;
4: Target lexical_cast(const source&);
```

With branching enabled, we can improve the perfor-mance of these string-to-double conversion without re-placing lexical_cast with some other function calls.

3.1.2 SFINAE

Compiler creates a set of an overload resolution of matching template when compiler encounters a call to several functions with the same name as a function template.SFINAE (Substitution Failure Is Not An Error) is the technique to avoid the compilation abort even if the substitution of the deduced type arguments in the templates argument list or function parameter list causes an error.

Listing.2. SFINAE

```
1: template <class T>
2: bool equal(T x, T y)
3: {
4:   return x == y;
5: }
6: template <>
7: bool equal(const char* x, const char* y)
8: {
9:  return std::strcmp(x, y) == 0;
10: }
11: int main()
12: {
13:  int n = 3;
14:  const char* s = "C++";
15:  std::cout << equal(s, "Java") << std::endl;
16:  std::cout << equal(n, 2) << std::endl;
17: }
```

At line 15, the substitution of the deduced type arguments (in the argument list of template) or the function parameter list is succeeded. At line 16, the substitution causes the error. Consequently, the compiler removes the candidate from its overload resolution set. The compiler only flags an error if, at the end of the process, the overload resolution set is empty (no candidates) or has multiple equally good candidates (ambiguity).SFINAE can be regarded as one of the branching techniques of compile-time programming. Also, SFINAE is implemented based on the explicit specialization.

3.2 Higher-order programming

3.2.1 Function objects

A function object which is also called as functor provides the persistent object to operate functions like variables during execution. To put it simply, the main purpose of function objects is implementing callback functions.

Listing.3. Function objects

```
1: bool is_substr_of( const string& sub, const string& all )
2: {
3:   return all.find( sub ) != string::npos;
4: }
5: int main()
6: {
7:    function<bool (const string&, const string&)> f;
8:    f = &is_substr_of;
9:    cout << f( "a", "abc" )  << endl;
10: }
```

At line 1-4, the function object of is_substr_of() is defined. At line 7-8, function object is generated and pointed to the variable f.

3.2.2 Binding functions

Function objects become more effective with binding functions. To name a few, binding functions are lambda expressions, Boost.Phoenix and Boost.Bind. Compared with a straight function call, function objects have two thrusts (advantages). At first, a function object can holds state. Secondly, a function object is a type, which result in that it can be utilized as the template parameters.

.

- Linear static analysis C++ Boost provides Boost::bind which is a generalization of the standard functions of std::bind1st and std::bind2nd. Bind supports arbitrary function objects, pointers and member function pointers. Bind is able to connect any arguments or route input arguments in arbitrary position. Also, Bind does not place any requirements on the function object; in particular, it does not need the result_type, first_argument_type and second_argument_type standard typedefs.

- Lambda expression is an anonymous function utilities provided by C++, Java and so on. Broadly, anonymous function is defined at the site where it is called. Lambda expression is originated from Alonzo Church's λ-calculus. The concept of anonymous come from that it consists of a function body but not bound to a function name. It takes advantages in generating a function definition at any point in the lexical scope of a program, where you would expect to pass a function object.

IV. PROPOSAL METHOD

Our DSEL requires the polymorphism corresponding to the data format of items shown in TABLE I. For example, our parser needs to switch template functions by the formats such as X.X.X.X (IP address) and YYYY-MM-DD (timestamp).In this case, function templates are not always the best way to handle polymorphism.
Instead, we apply type erasure for handle several callback functions for each data items (address, timestamp, application and so on).

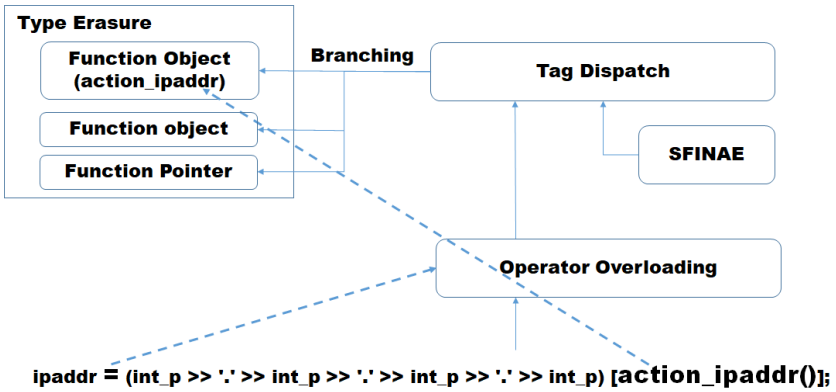

Fig. 2. Proposal method using Type Erasure

Figure 1 depicts the proposal method. The parsing code on lower side of the slide as follows

Listing.4. Calling Function Object

```
1: definition( const AddrParse& self )
2: {
3: ipaddr = (int_p >> '.' >> int_p >> '.' >> int_p >> '.' >> int_p)[Action_ipaddr()];
4: }
```

Action_ipaddr() is function object which is invoked when the parser recognized the format of X.X.X.X (ip address). Besides, other function objects and pointers are stored with type erasure.When the parser finds =, operator overloading function is called. Inside the function, tag dispatch is implemented to switch the function objects correspondingly. As we discussed in section III-A, the word switching is equivalent to branching. Therefore, tag dispatch is enabled by applying SFINAE.

4.1 Linear static analysis or equivalent static analysis

Type erasure is a technique for removing explicit type annotations from a program in the load-time process. It is executed in compile-time (before run-time). Instead of type-passing semantics, type erasure adopts operational semantics which does not require programs accompany by types. In the view of the abstraction principle, type erasure ensures that the run-time program execution is independent of type information. On the other hand, reification is recognized as the opposite of type erasure. In C++, type erasure is achieved by encapsulating a concrete implementation in a generic wrapper. Also, it is en-abled by providing virtual accessor methods to the concrete implementation via a generic interface.

Listing.5. Type Erasure

```
1: union any_pointer {
2:  void (*func_ptr)();
3:  void* obj_ptr;
4: };
5: template <class Func, class R>
6: struct function_ptr_manager {
```

```
7:  static R invoke(any_pointer function_ptr)
8:  {}
9:};
10:
11: template <class Func, class R>
12: struct function_obj_manager {
13:  static R invoke(any_pointer func_obj)
14:  {}
15: };
```

Listing 5 shows the example of type erasure. Types of function pointer and function object are erased and store in *void of union at line 1-4.

4.1 Operator overloading

Operator overloading is another technique for handling a compile-time polymorphism.The operator is overloaded to yield the specific meaning to the user-defined data type.

Listing.6. Operator overloading

```
1: template <class Func>
2: function& operator=(Func func)
3: {
4: typedef typename
5: get_function_tag<Func>::type func_tag;
6: assign_to(func, func_tag());
7: return *this;
8: }
```

Listing 6 shows the example of operator overloading. In listing 6, it is used to redefines of =. By doing this, it can perform on the user-defined data type.In this case, when the compiler finds =, the function assign_to() is invoked with the type of get_function_tag<Func>.

4.2 Tag dispatch

Like ducktyping, tag dispatching is a technique for applying function overloading to dispatch corresponding to properties of a type. Tag dispatching adopts function overloading for yielding executable code based on type properties information.

Listing.7. Tag dispatch

```
1:struct function_ptr_tag {};
2:// function pointer
3:struct function_obj_tag {};
4:// function object
5:
6:template <class Func>
7:struct get_function_tag {
8: typedef typename if_<is_pointer<Func>,
9: function_ptr_tag,
10: function_obj_tag>
11:  ::type type;
12:};
```

Listing 7 shows the example of tag dispatching. Tag at line 1 and 3 is simply an empty class whose only purpose is to convey some information at compile time. In this case the iterator concept modeled by a given iterator type.At line 8, our parser adopts SFINAE to switch type of function (pointer or object) with tag dispatch.

V. IMPLEMENTATION

5.1 Boost Spirit

The Boost Spirit parser framework is designed for recursive decent parser generation based on template metaprogramming techniques. One of core techniques of Boost Spirit is expression templates which enables

users to approximate the syntax of ENBF (Extended Backus Naur Form) like grammar.In Boost Spirit, parser object is a backtracking LL(∞) parser that is capable of parsing rather ambiguous grammars.

5.2 Parsing expression grammar

Parsing Expression Grammars (PEG) [2] are a derivative of Extended Backus-Naur Form (EBNF) [3] with a differ-ent expression. PEG is implemented to cope with a recur-sive decent parser. In other words, a PEG can be directly interpreted in a recursive-descent parser's manner like EBNF.

PEG is designed for describing a formal language in or-der to represent a set of rules applied to recognize strings and tokens. Another advantage of PEG over EBNF is that it performs with an exact interpretation. In each PEG grammar, only one valid parse tree is determined.

Listing.8. Parsing expression grammar

```
1: definition( const AddrParse& self )
2: {
3: ipaddr = (int_p >> '.' >> int_p>> '.' >> int_p >> '.' >> int_p)[Action_ipaddr()];
4:
5: timestamp = (int_p >> '-' >> int_p >> '-' >> int_p >> 'T' >> int_p >> ':' >> int_p >> ':' >> int_p >> 'Z')
6: [Action_timestamp()];
7
8: r = timestamp | ipaddr;
9: }
```

Listing 8 shows the example of PEG of Boost Spirit. At line 3, program defines the format of IP address (X.X.X.X).Also, timestamp format (YYYY-MM-DDThh:mm:ssZ) is defined at line 5-6.

5.3 Semantic Actions

Semantic actions are technology for yielding some output and doing some task besides syntax analysis based on PEG. Actions can be attached to any expression at any level as long as it is inside the parser hierarchy. In a nutshell, a function object represented in a C/C++ is called if it discovers a match in the particular context under processing.

Listing.9. Semantic action for IP address

```
1: struct Action_ipaddr
2: {
3:  template<typename Ite>
4:  void operator()( Ite i1, Ite i2 ) const
5:  {
6:   m_IPAddr.insert(std::make_pair(counter,
7:   string(i1,i2)));
8:   }
9: };
```

Listing.10. Semantic action for timestamp

```
1:struct Action_timestamp
2:{
3: template<typename Ite>
4: void operator()( Ite i1, Ite i2 ) const
5: {
6:  m_timestamp.insert(std::make_pair(counter,
7:  string(i1,i2)));
8: }
9: };
```

Listing 9 and 10 are the examples of semantic actions. These are called when the parser finds the IP address and timestamp. In this case, parsed items are stored in the STL container at line 6-7 in Listing 9 and line 6-7 in Listing 10.

5.3 Data structures

After semantic action is completed, parser translates original traffic log (which is session data in this case) into the representation of multi-index. However, each semantic action is not capable of coping with multi-index directly due to some syntax restriction of Boost Spirit. Instead, our parser uses multimap to store the output of each semantic action.Figure 3 shows our proposal method. At first stage, parser extracts pairs of <line_number, timestamp>, <line_number, sourceIP> and <line_number, destIP> for each line of session log data.Then, after parsing, these pairs are reduced into the multi-index of <line_number, timestamp, sourceIP, destIP>.

5.3.1 Boost-Multimap

A Boost-Multimap is an extension of associative container for supporting equivalent keys which often contains multiple copies of the same key value). It provides quick retrieval of value of another type T based on the keys. Also, this multimap has an utility of bidirectional iterators. Compared with associative container, multimap satifies all the requirements of a container or reversible container. The value_type stored by this container is the value_type is std::pair<const Key, T>.

Listing.11. Operating Multimap

```
1: static int counter;
2: std::multimap<int, std::string> m_IPAddr;
3: std::multimap<int, std::string> m_timestamp;
4:
5: m_IPAddr.insert(std::make_pair(counter, string(i1,i2)));
6: m_timestamp.insert(std::make_pair(counter, string(i1,i2)));
```

Listing 11 is the example of multimup. At line 5-6, parser inserts the pair of ipaddress (IPAddr) and timestamp tied to line number (counter).

5.3.2 Boost-MultiIndex

The Boost Multi-Index Containers enables the construction of containers with one or more indices with different sorting and access semantics. In using multi-index, indices provide interfaces which are similar to those of STL containers. But the concept of multi-indexing over the same collection of elements is derived from relational database terminology.Consequently, multi-index allows for the specification of complex data structures in the case that simple sets and maps are not enough. Multi-index provides a wide selection of indices.

Listing.12. Multi-index

```
1: struct session {
2:  int linenumber;
3:  std::string source_ipaddr;
4:  std::string dest_ipaddr;
5:  std::string timestamp;
6:
7:  session(int linenumber, const std::string& source_ipaddr, std::string& dest_ipaddr, std::string& timestamp)
8:     : linenumber(linenumber), source_ipaddr(source_ipaddr), dest_ipaddr(dest_ipaddr), timestamp(timestamp)
{}
9:
10:  void print() const
11:  { std::cout << linenumber << "," << source_ipaddr << "," << dest_ipaddr << "," << timestamp << std::endl;
}
12: };
13:
14:sess.insert(session(itr->first, itr2->second, tmp_string, itr->second));
```

Listing 12 is the example of multi-index. Multi-index containing linenumber, source_ipaddr, dest_ipaddr and timestamp is defined at line 1-12. At line 14, three multimaps are reduced and inserted into multi-index as shown in Figure 3. Besides, Boost.Multiindex support subobject searching, range querying, in-place updating of elements and calculation of ranks.

VI. Experimental result

In experiment, we use workstation with Intel(R) Xeon(R) CPU E5-2620 v4 (2.10GHz) and 251G RAM. Figure 1 depicts the elapsed time in parsing session data log. X-axis is the number of lines of session data log file. Y-axis is the elapsed time. The elapsed time of parsing log file increases linearly corresponding to the file size except some spikes such as one around 45,000. Figure 2 depicts the elapsed time in inserting key-value pair data into multi-index. X-axis is the number of lines of session data log file. Y-axis is the elapsed time.

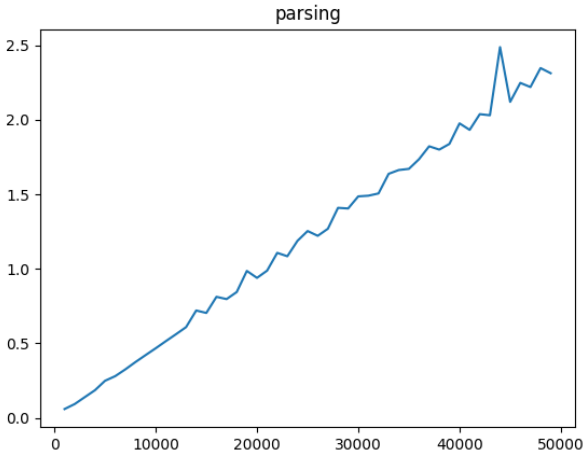

Fig. 3.Elapsed Time in Parsing Session Log File

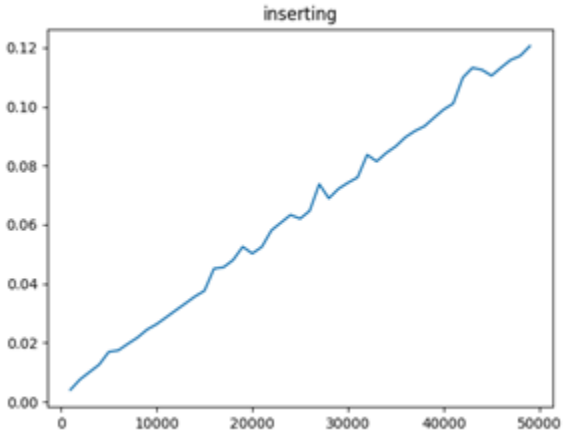

Fig. 4. Elapsed Time in Inserting Key-Value into Multi-Index

VII.RELATED WORK

7.1 Metaprogramming

Various approaches of metaprogramming [4] has been proposed to cope with embedding of DSLs.One of these approaches is code generation where code is converted to AST representation of the embedded DSL program at compile-time. Particularly, the multi-state programming approach [5] is proposed to interpret a program in several different phases. In [5], compiler operates at run-time, but at a different phase than the actual processing of a region. Another distinguished research about applying metaprogramming for DSL is given by Seefried et al. [6]. In [6], Template Haskell [7] is used to implement PanTHeon and Pan [8].

7.1 Functor

One of the concepts of functor [9][10][11] is originated from the module systems of SML[10] and OCaml [11]. It provides a way to abstract over required services in a statically type-checked setting.Funtor still

functors still pose severe restrictions when it comes to structuring components.Accordingly, functor is used with function-binding utilities such as Lambda expression and Boost.Bind.

7.3 Pattern Matching

Various techniques of pattern matching in object-oriented programming has been proposed to message exchange in distributed systems [13], semi-structured data [14] and UIevent handling [15].Moreau, Ringeissen and Vittek [16] propose the method to translate pattern matching code into existing languages, without any requirement extensions.For Java, Liu and Myers [17] add a pattern matching construct using a backward mode of execution.An alternative technique of multi-methods [18]is proposed. In [18], pattern matching is unified with method dispatch.Also, [19][20] extends multi-methods to predicate-dispatch.In [19][20], functional programming languages are proposed to convert from one data type to another in pattern matching.

7.3 Parser expression grammar

Parser expression grammar is inspired by Birman's TS/TDPLand gTS/GTDPL systems [23][24][25]. Adams [26] adopts TDPL in a modular language prototyping framework.Also, various practical top-down parsers such as ANTLR [27], PARSEC combinator library for Haskell [28] are available. These top-down parsers provides provide backtracking capabilities that conform to the model in practice.

7.3 Time series analysis

Another important topic of time series analysis is outlier/anomaly detection. In [29], a data structure which is k-ary sketch is proposed for efficient utilization of memory. Also, k-ary sketch enables a constant, per-period update and reconstruction cost. Popular algorithms of anomaly detection of temporal data are ARIMA, HMM and SVM.Paundu [30] et al. proposes a sequence-based analysis using SVM and HVM for anomaly detection of time-sequence of instrumentation data of VMM (virtual machine monitor).

7.3 DSL

Configurable language for network traffic analysis and intrusion detection is promising application of DSL. PADS [31] is a declarative data description language for describing both the physical layout and semantic properties of ad hoc data traffic. As an extension of PADS, Fisher [32] propose an algorithm of automated inference of the structure of ad hoc datasource and a format specification in the PADS.Chimera [33] provides a declarative query language for intrusion detection systems with a platform-independent SQL syntax.SAQL [34] is a stream-based query system for incorporating expert knowledge to perform timely anomaly detection in large scale traffic data.

VIII. DISCUSSION

As attacks are increasing in sophistication, analytics should be also sophisticated that detect them. Over time it is becoming more and more difficult to characterize malicious behavior with simple Snort rules [36]. As a result, many administrators rely on systems like Bro [35] that are able to perform stateful analysis on high-level protocol fields, rather than being constrained to individ-ual packet or flow analysis.

Besides, network traffic in general has become more invisible. To name a few, major cloud vendors such as Cloudflare recently deploys DNS over TLS/HTTPS. Also, with the spread of TLS 1.3, the middlebox appliances becomes less effective. Consequently, current Botnet running over cloud platform is harder and harder to detect and analyze.Given these circumstances, declarative languages which maintains as much expressive power as possible while not imposing the significantly impacting performanceon intrusion detection systems. Also more flexible framework is necessary for providing logical construction of the expression of sophisticated attacks.

IX. CONCLUSION

The Science Information Network (SINET) is a Japanese academic backbone network for more than 800 universi-ties and research institutions. For handling various and massive security incidents on SINET, a new framework of declarative language which maintains as much expressive power as possible without imposing significant perfor-mance.

In this paper, we have proposed lightweight C++ DSEL implementation for coping with huge session data on ac-ademic backbone network. Our method adopts simple type erasure techniques instead of one of virtual function table. Our method can achieve the efficiency and flexi-bility is required for complex intrusion detection tasks. In our model, the function object is implemented by type erasure for constructing internal DSL for processing time-series data. Type erasure enables our parser to store function pointer and function object into the same *void type with class templates. Besides, we apply a novel op-erator (=) overloading with tag dispatch

for handling a branch of pointer and object. In implementing tag dis-patch, a compiler time programming technique called as SFINAE (Substitution Failure Is Not An Error) is adopted.

In the experiment, we have measured the elapsed time in parsing and inserting IPv4 address and timestamp data format ranging from 1,000 to 50,000 lines with 24 row items. It has been turned out that proposal method can work in feasible computing time. For further work, our parser could be extended for interoperating expert knowledge to perform timely anomaly detection over the large-scale provenance data.

REFERENCES

[1]. I. Brcic: "Ideally Fast" Decimal Counters with Bistables. IEEE Trans. Electron. Comput. 14(5): 733-737 (1965)
[2]. Bryan Ford: Parsing Expression Grammars: A Recognition-Based Syntactic Foundation, http://pdos.csail.mit.edu/~baford/packrat/popl04/
[3]. Richard E. Pattis: EBNF: A Notation to Describe Syntax, http://www.cs.cmu.edu/~pattis/misc/ebnf.pdf
[4]. K. Czarnecki, J. T. O'Donnell, J. Striegnitz, and W. Taha. DSL implementation in MetaOCaml, Template Haskell, and C++. In Domain-Specific Program Generation, volume 3016 of LNCS, pages 51-72. Springer, 2003.
[5]. W. Taha. A gentle introduction to multi-stage programming. In Domain-Specific Program Generation, Springer LNCS 3016, pages 30-50, 2003.
[6]. S. Seefried, M. M. T. Chakravarty, and G. Keller. Optimising embedded DSLs using Template Haskell. In G. Karsai and E. Visser, editors, GPCE, volume 3286 of Lecture Notes in Computer Science, pages 186-205. Springer, 2004.
[7]. T. Sheard and S. Peyton Jones. Template metaprogramming for Haskell. In M. M. T. Chakravarty, editor, ACM SIGPLAN Haskell Workshop 02, pages 1-16. ACM Press, Oct. 2002.
[8]. C. Elliott. Functional images. In The Fun of Programming,"Cornerstones of Computing" series. Palgrave, Mar. 2003.
[9]. D. MacQueen. Modules for Standard ML. In Conference Record of the 1984 ACM Symposium on Lisp and Functional Programming, Papers Presented at the Symposium, August, 1984, pages 198-207, New York, August 1984. Association for Computing Machinery.
[10]. R. Harper and M. Lillibridge. A Type-Theoretic Approach to Higher-Order Modules with Sharing. In Proc. 21st ACM Symposium on Principles of Programming Languages, January 1994.
[11]. X. Leroy. Manifest Types, Modules and Separate Compilation. In Proc. 21st ACM Symposium on Principles of Programming Languages, pages 109-122, January 1994.
[12]. K. Fisher and J. H. Reppy. The Design of a Class Mechanism for Moby. In SIGPLAN Conference on Programming Language Design and Implementation, pages 37-49, 1999.
[13]. Lee, K., LaMarca, A., Chambers, C.: HydroJ: Object-oriented Pattern Matching for Evolvable Distributed Systems. In: Proc. of Object-Oriented Programming Systems and Languages (OOPSLA). (2003)
[14]. Gapeyev, V., Pierce, B.C.: Regular Object Types. In: Proc. of European Conference on ObjectOriented Programming (ECOOP). (2003)
[15]. Chin, B., Millstein, T.: Responders: Language Support for Interactive Applications. In: Proc. of European Conference on Object-Oriented Programming (ECOOP). (2006)
[16]. Moreau, P.E., Ringeissen, C., Vittek, M.: A Pattern Matching Compiler for Multiple Target Languages. In: In Proc. of Compiler Construction (CC), volume 2622 of LNCS. (2003) 61-76
[17]. Liu, J., Myers, A.C.: JMatch: Iterable Abstract Pattern Matching for Java. In: Proc. of the 5th Int. Symposium on Practical Aspects of Declarative Languages (PADL). (2003) 110-127
[18]. Clifton, C., Millstein, T., Leavens, G.T., Chambers, C.: MultiJava: Design Rationale, Compiler Implementation, and Applications. ACM Transactions on Programming Languages and Systems 28(3) (May 2006) 517-575
[19]. Ernst, M., Kaplan, C., Chambers, C.: Predicate dispatching: unified theory of dispatch. In: Proc. of European Conference on Object-Oriented Programming (ECOOP). Volume 1445 of Springer LNCS. (1998) 186-211
[20]. Millstein, T.: Practical Predicate Dispatch. In: Proc. of Object-Oriented Programming Systems and Languages (OOPSLA). (2004) 245?364
[21]. Wadler, P.: Views: A way for pattern matching to cohabit with data abstraction. In: Proc. of Principles of Programming Languages (POPL). (1987)
[22]. Okasaki, C.: Views for Standard ML. In: In SIGPLAN Workshop on ML, pages 14-23. (1998)
[23]. Alfred V. Aho and Jeffrey D. Ullman. The Theory of Parsing, Translation and Compiling - Vol. I: Parsing. Prentice Hall, Englewood Cliffs, N.J., 1972.
[24]. Alexander Birman. The TMG Recognition Schema. PhD thesis, Princeton University, February 1970.
[25]. Alexander Birman and Jeffrey D. Ullman. Parsing algorithms with backtrack. Information and Control, 23(1):1-34, August 1973.
[26]. Stephen Robert Adams. Modular Grammars for Programming Language Prototyping. PhD thesis, University of Southampton, 1991.
[27]. Terence J. Parr and Russell W. Quong. ANTLR: A PredicatedLL(k) parser generator. Software Practice and Experience, 25(7):789-810, 1995
[28]. Daan Leijen. Parsec, a fast combinator parser. http://www.cs.uu.nl/?daan.
[29]. Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, Yan Chen: Sketch-based change detection: methods, evaluation, and applications. Internet Measurement Conference 2003: 234-247
[30]. Ady Wahyudi Paundu, Takeshi Okuda, Youki Kadobayashi, Suguru Yamaguchi: Sequence-Based Analysis of Static Probe Instrumentation Data for a VMM-Based Anomaly Detection System. CSCloud 2016: 84-94
[31]. Kathleen Fisher, Robert Gruber: PADS: a domain-specific language for processing ad hoc data. PLDI 2005: 295-304
[32]. Kathleen Fisher, David Walker, Kenny Qili Zhu, Peter White: From dirt to shovels: fully automatic tool generation from ad hoc data. POPL 2008: 421-434
[33]. Kevin Borders, Jonathan Springer, Matthew Burnside: Chimera: A Declarative Language for Streaming Network Traffic Analysis. USENIX Security Symposium 2012: 365-379
[34]. Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R. Kulkarni, Prateek Mittal: SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection. USENIX Security Symposium 2018: 639-656
[35]. Vern Paxson: Bro: a system for detecting network intruders in real-time. Comput. Networks 31(23-24): 2435-2463 (1999)
[36]. Martin Roesch: Snort: Lightweight Intrusion Detection for Networks. LISA 1999: 229-238

Ruo Ando, et. al. "A type-erasure based DSEL for huge time-series session data analysis." *International Refereed Journal of Engineering and Science (IRJES)*, vol. 09, no. 05, 2020, pp 41-51.