Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 9 (2014) pp. 167-176


# Normalisation of Log Messages for Intrusion Detection

*Invited paper. Preliminary version of this paper appears as "Hierarchical Object Log Format for Normalisation of Security Events" in Proceedings of the 9th International Conference on Information Assurance and Security (IAS 2013).*

**Andrey Sapegin, David Jaeger, Amir Azodi, Marian Gawron, Feng Cheng, Christoph Meinel**

[1]Hasso Plattner Institute (HPI), University of Potsdam,
P.O.Box 900460, 14440 Potsdam, Germany
{*andrey.sapegin, david.jaeger, amir.azodi, marian.gawron, feng.cheng, christoph.meinel*}*@hpi.de*

***Abstract***:
**The differences in log file formats employed in a variety of services and applications remain to be a problem for security analysts and developers of intrusion detection systems. The proposed solution, i.e. the usage of common log formats, has a limited utilisation within existing systems for security management. In our paper, we reveal the reasons for this limitation and show disadvantages of existing common log formats for normalisation of security events. To deal with it we have created a new log format that fits for intrusion detection purposes and can be extended easily. Based on our developing intrusion detection system, we demonstrate advantages of offered format. However, taking previous work into account, we would like to propose a new format as an extension to existing common log formats, rather than a standalone specification.**
***Keywords***: log normalisation, intrusion detection, common log format

## I. Introduction

Nowadays, every application has its own log file format. Such a variety of formats complicates log analysis [1] and causes problems for system administrators, developers of intrusion detection systems (IDS) and security analysts [2, 3]. Therefore, several log format standards [4] were introduced aiming to solve this problem: CEE [3], CEF [5], IODEF [6]. However, there are many developers who avoid using common log formats in their projects. This statement particularly applies to intrusion detection systems. Many software vendors and open source projects still use their own log format for IDSs [7, 8, 9]. The reasons for this situation hide in the IDS architecture. To operate rapidly, such systems need to analyse huge amount of logs from hundreds of devices and services on the fly [10]. Therefore, the log format should be lightweight and able to encapsulate all network and host logs from large datasets.

Log file standards that aim to handle all possible log output of any service, do not fit for these purposes. Moreover, in some cases, the variety of standardised fields, offered by such a common log file format, does not provide a suitable schema for normalisation and correlation of security events. During the development of our own IDS —Security Analytics Lab [11]— we have faced the same challenges and came to the same solution: a log format that is specially designed for IDSs. However, the object log format we present in this paper noticeably differs from the existing solutions.

Our goal was to experiment with a flexible lightweight format to optimise attack detection. First, we analysed a variety of large log files and came up with a list of common fields that can be found in the logged events. Then, this list was reduced to the fields that we considered most important for the security analysis of occurred events.

As a result, the developed log format facilitates the detection of attacks and simplifies their correlation. We demonstrate the ability of our format to simplify attack detection within a case study. Besides this particular use case, the proposed log format retains to be adjustable and could be utilised for generic normalisation concepts.

The remainder of the paper has the following structure: Section II describes the data set used, Section III evaluates and compares existing log formats, Section IV provides details on the log format we developed and Section V tells about results of a case study. Next, Section VI provides details on the architecture of Security Analytics Lab, while Section VII describes future work. We discuss our results and conclude in the Section VIII.

## II. Data

To prove our concepts, we utilised data from "Scan of the Month" Honeynet Challenge: "Scan 34 - Analyze real honeynet logs for attacks and activity" [12]. The Honeynet Project [13] aims to help security specialists to sharpen their forensics and attack detection skills. To achieve it, the project shares real attack data collected from honeynets from all over the world. The "Scan of the Month" challenge provides archives with log files from different hardware and software systems for the postmortem analysis. The "Scan 34" challenge contains log files being collected between 30 January and 31 March 2005. We chose this particular challenge because it implies analysis of multiple log files for different services and includes various heterogeneous attack traces. We describe the log files in Table 1.

| Server | Service | Date | Total number of lines |
|---|---|---|---|
| n/a | HTTP Server | Jan 30 - Mar 16 | 3925 |
| bridge | iptables firewall | Feb 25 - Mar 31 | 179752 |
| bastion | snort IDS | Feb 25 - Mar 31 | 69039 |
| combo | syslog | Jan 30 - Mar 17 | 7620 |

*Table 1*: Log files available to solve the challenge

The "Scan 34" challenge also provides the description on the attacks and important events in the log files. See the details of the 15 most important events in Table 2.
In total, four attacks led to the server's compromisation. Two of four attackers used the vulnerability of 'awstats.pl' installed after the server reboot, probably by a system administrator. Another two attackers successfully brute-forced the SSH password. The first attacker has also installed an IRC bot right after the intrusion. Other events listed in the table were not harmful for the server; they represent suspicious activity that did not result into a successful intrusion.

## III. Existing log formats for normalisation of security events

Before we decided to use our own log format, we have evaluated existing solutions, trying to estimate if they could fit for our purposes of normalisation of security events. Please see the overview of proposed log formats in Table 3.
The selected formats have different structure, size and even usage purpose. CEE is a generic format for logging any types of events [3], IDMEF was developed for exchange of security messages [14]. IODEF is specially designed for computer security incidents [6], as well as CEF developed as a part of intrusion detection system [5]. Although the described formats have different purposes, each of them could be used for log normalisation.
However, not every part of the log message could be normalised to the standard fields defined in every format. To deal with it, each format allows to use extra fields or extend its structure. Therefore we decided to check how many changes we need to fully normalise log messages we have into each of 4 formats. The Table 4 shows, that some parts of log messages have no corresponding field in the log format. This small observation reveals a deeper problem, if investigated. The reviewed log formats offer limited number of options to parse the textual event description precisely. For example, using the IDMEF, messages like "authentication failure" and "Relaying denied. IP name lookup failed" are supposed to be written into the Classification class as whole. Such single field available in the log format does not allow to effectively normalise descriptions like "BLEEDING-EDGE WORM Mydoom.ah/i Infection IRC Activity [Classification: A Network Trojan was detected]". CEE format demonstrates a similar issue, as all descriptions and other details are supposed to be stored in the "message" field. Since the information still could be stored in such generic fields, we do not mention it in the table as parts without corresponding field.
Another common issue affecting all log formats implies complications while normalising the event details like user name, rule/action applied, method (e.g., GET) and response (404 or 550). In other words, the existing formats offer a limited number of fields to specify, what has happened. CEF deals with it better than other formats, due to the presence of keys like 'act' (Action mentioned in the event), 'app' (Application protocol), 'request', 'requestMethod' and others.
Finally, all formats have too few – or do not have any – fields for expression of properties specific for intrusion detection, like relation between events, security metrics or classifiers such as CVE [15] and CWE [16] identifiers, object and subject of the event. These properties usually could not be extracted from the log lines itself, but are expected to be filled by IDS.
Thus, not only missing fields from the table should be added to each of formats, but also the fields to improve parsing of message parts and allow to store intrusion detection properties. That's why we decided to select and re-engineer one of the formats. However, the structure, number of fields/attributes and possible values or contents are different for all formats, which make them hard to compare with each other. Therefore, we chose the following additional criteria to examine log formats:

- *scalability*. There should be an ability to add custom fields, if some log formats could not be normalised into standard ones.
- *light weight*. The log format should have reasonable number of fields/attributes to avoid redundancy and fit for high-speed normalisation purposes.
- *multilevel schema*. Intrusion detection implies correlation of the normalised logs. The hierarchical or other connected structure is preferable as it will allow to categorise different fields into classes.

Applying the first criterion, all selected formats are scalable to some extent. IDMEF [14] and IODEF contain an *Additional Data* class, CEF supports *Custom Dictionary Extensions* and CEE allows to create custom events and fields.
Analysing the second criterion, i.e. the number of fields/attributes, the CEE format is much more compact than IODEF, IDMEF and CEF. We also would like to mention, that IDMEF is designed for data exchange for intrusion detection and security management systems only. So we could expect a lot of additional attributes being added to IDMEF to adopt it for normalisation purposes.
Finally the last criterion is examined. The CEF has a flat structure; IODEF and IDMEF have similar multi-level schemas with connections between different classes; and CEE has a clear object-based hierarchical structure.
Based on the criteria defined, we selected CEE to be extended as the log format that fits better for our purposes.

| N | Event | Date | Details |
|---|---|---|---|
| 1 | Reboot | Feb 11, 2005 | n/a |
| 2 | Software installation | Feb 25, 2005 | AWSTATS installed |
| 3 | Server compromised | Feb 26, 2005 | Code injection through awstats.pl |
| 4 | Server compromised | Mar 04, 2005 | Code injection through awstats.pl |
| 5 | Server compromised | Mar 06, 2005 | ssh brute-force successful |
| 6 | Server compromised | Mar 13, 2005 | ssh brute-force successful |
| 7 | Software installation | Feb 26, 2005 | IRC bot installed by an attacker |
| 8 | ICMP alert | n/a | ICMP Destination Unreachable |
| 9 | Slammer worm | n/a | Worm propagation attempt |
| 10 | IIS attacks | n/a | WebDAV search access, cmd.exe access, etc. |
| 11 | SMTP scan | n/a | POLICY SMTP relaying denied |
| 12 | Typot trojan | n/a | trojan traffic |
| 13 | RPC scan | n/a | RPC portmap status request |
| 14 | Port scan | n/a | NMAP -sA (ACK scan) |
| 15 | Slapper worm | n/a | Worm propagation attempt |

*Table 2*: Security related events that occurred during the monitoring for the challenge

| Format name | Organisation | Size estimation | Format structure |
|---|---|---|---|
| CEE (Common Event Expression) | MITRE | 58 fields, 7 objects | two levels, hierarchical, object-based |
| IDMEF (Intrusion Detection Message Exchange Format) | IETF | 118 elements, 5 core classes, 53 attributes | multi-level, class-based |
| IODEF (Incident Object Description Exchange Format) | IETF | 53 elements, 19 top-level classes, 83 attributes | multi-level, class-based |
| CEF (ArcSight Common Event Format) | HP | 104 keys | one level, key-value pairs |

*Table 3*: Existing standards for log formats

| Log line | CEE | IDMEF | IODEF | CEF |
|---|---|---|---|---|
| 81.181.146.13 - - [15/Mar/2005:05:06:53 -0500] “GET //cgi-bin/awstats/awstats.pl? configdir= —%20id%20— HTTP/1.1” 404 1050 “-” ”Mozilla/4.0 (compatible; MSIE 6.0; Windows 98) | HTTP/1.0, GET, 404 | Mozilla/4.0, 404 | 81.181.146.13, GET, //cgi-bin/awstats/ awstats.pl, 404, Mozilla/4.0 | - |
| Mar 15 13:38:03 combo sshd(pam_unix)[14490]: authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=202.68.93.5.dts.net.nz user=root | - | - | 14490, 202.68.93.5 .dts.net.nz, user=root | 14490 |
| Mar 1 20:45:12 bastion snort: [1:2001439:3] BLEEDING-EDGE WORM Mydoom.ah/i Infection IRC Activity [Classification: A Network Trojan was detected] [Priority: 1]: TCP 11.11.79.67:2568 -> 129.27.9.248:6667 | TCP | - | Priority: 1, 11.11.79.67, 129.27.9.248 | Priority: 1 |
| Mar 24 19:46:50 bridge kernel: INBOUND ICMP: IN=br0 PHYSIN=eth0 OUT=br0 PHYSOUT=eth1 SRC=63.197.49.61 DST=11.11.79.100 LEN=32 TOS=0x00 PREC=0x00 TTL=111 ID=1053 PROTO=ICMP TYPE=8 CODE=0 ID=512 SEQ=29421 | ICMP, eth0, eth1, br0 | - | SRC = 63.197.49.61, DST = 11.11.79.100, eth0, eth1, br0 | br0 |
| Feb 1 10:08:32 combo sendmail[32433]: j11F8FP0032433: ruleset=check_rcpt, arg1 = <china9988@21cn.com>, relay=[61.73.94.162], reject=550 5.7.1 <china9988@21cn.com>... Relaying denied. IP name lookup failed [61.73.94.162] | ruleset = check_rcpt, 550, china9988@ 21cn.com | ruleset = check_rcpt, 550, china9988@ 21cn.com | check_rcpt, relay= [61.73.94.162], 550, 61.73.94.162 | 32433, ruleset = check_rcpt |

*Table 4*: Parts of log messages without corresponding field in the log format.

## IV. Object Log Format

As mentioned in the previous Section, the decision to develop our own log format for IDS came with the attempt to utilise the CEE format in our experimental intrusion detection system [11].

We started with modifying the CEE format and adding custom fields to cover all possible log message variants from the Honeynet challenge. However, this draft log format still had several problems. First, we have used less than half of fields

(24 of 58) defined in the standard. Compared to the number of custom fields added – 59 – and the fact that standard CEE fields do not always present the key properties of the log message, it becomes hard to argue for using the resulting format inside the IDS system. Second, the object-field hierarchy defined in the Field Dictionary [17] contains only one abstraction level. Taking into account overlapping semantics (e.g. *appname* and *app.name* fields) in the CEE notation, this relatively flat structure contains many similar fields and adds a lot of confusion for a developer. To handle with our goals, we made changes to the CEE structure as well. We extended the hierarchical structure of CEE to three levels to achieve flexibility and improve clarity.

We present the proposed format in Figure 1. The first level (marked with bold) describes global parameters or classes of parameters, such as *network* or *original_event*. On the second level of our hierarchy (written in normal font) we describe the most significant properties. And on the third level (marked with italics) we place specific information such as network protocol fields.

Compared to the formats examined in the Section III, the proposed format offers multiple fields to store event details describing what has happened (*tag* and *application classes*), as well as fields, which are highly relevant for intrusion detection (mainly, classes *related_ids*, *relation* and *security*).

Let's consider the example from the Table 4 to show how to parse real log data into the proposed format:

```
81.181.146.13 - - [15/Mar/2005:05:06:53 -0500] ``GET
    //cgi-bin/awstats/awstats.pl? configdir= |\%20id
    \%20| HTTP/1.1'' 404 1050 ``-'' "Mozilla/4.0 (
    compatible; MSIE 6.0; Windows 98)
```

The log line listed above is taken from the 'access_log' file on the HTTP server. Using it, we now describe the most significant elements and how the information from our example log entry should be distributed over them.

- **network** This class covers all properties around the lower network layers, i.e. the link, network and transport layer of the TCP/IP protocol stack. It describes information about the source and destination endpoints of network events, including their MAC address, IP address and ports. The protocol fields in captured network packets are organised in subclasses, such as *ether*, *ip*, *icmp*, *tcp* and *udp*. These details can then be used to analyse the exact workings of a network communication. It should be noted, that since we are parsing security events and not only the network packets, this class could provide details for multiple network packets associated with an event, even if they were reported in different logs.

  From the log line sample, we extract only the IP address—81.181.146.13—to fill the 'network.src_ipv4' field.

- **application** This class covers all properties of applications and services involved in the event. The application class can represent different kind of involved applications in an event. In a network connection, such an application could be the client application, which initiates a connection or sends a packet to the server, or the service application, which is the target of a sent packet or initiated connection. To cover the various special characteristics in service communications, the *application* class also allows to further specify parameters in the context of HTTP, FTP and more. In a host-based event, this application is usually the application that initiates an action, e.g. an application that writes a file. The semantics of the specified application can be obtained from the *subject* and *object* field of the *tag* class.

  Analysing the example log line, it is possible to notice, that the application is a user agent, so the 'application.http.user_agent' field should be filled with "Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)" line and 'application.http.query_string' field – with the following line:

  "GET //cgi-bin/awstats/awstats.pl? configdir= —%20id%20— HTTP/1.1".

- **producer** This class gives information on the application that observed and eventually persisted an event. It should describe the first application that persisted the events and should not be changed to one of the intermediate processing applications.

  In our example, a producer is a HTTP server, namely "Apache HTTP Server", which should be written into the 'producer.appname'. If we would know some details about the host from the other log lines, we could also fill the 'producer.host' field, e.g. with *http_server*.

- **file** This class describes the files that were involved in an event. A file can appear in different contexts, i.e. mainly as a data source and target of access operations, such as read and write. In the case of an FTP or HTTP connection, this parameter could give information on the accessed resource. Similar to the *application* class, the concrete context for one event is defined by the *subject* and *object* field of the *tag* class.

  Considering the sample log line, the 'file.name' field will contain 'awstats.pl' and 'file.path' – *'//cgi-bin/awstats/awstats.pl'*.

- **original_event** This field keeps the original log as found in the log source. This is usually a string containing full log line or fields from log database.

  The whole sample log line should be filled into the 'original_event' in our case.

- **relation** This class serves the cases when several events are related to each other. In the case of multiple events sharing a common identifier for correlation, this identifier is stored in the *common_id* field. The *prev_dep* property indicates if the current event depends on a previous one.

  For the example considered, these fields will be empty, because the logged event is standalone and not related to any other events.

- **tag** This class provides abstract information on top of the message details, mainly to categorise and tag events. As this information is not always directly represented in

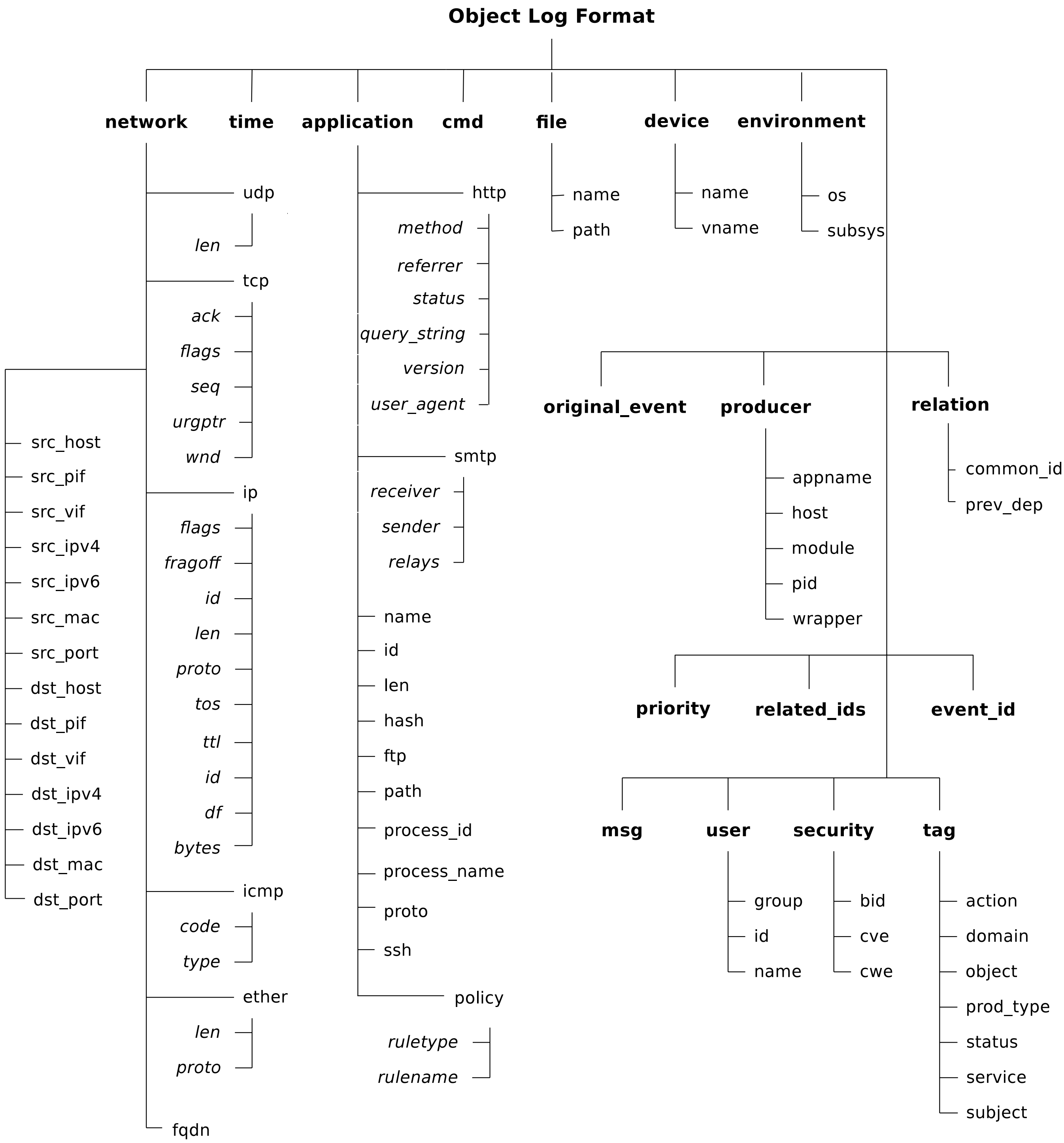


**Figure. 1**: Tree of properties in our log format

the log data, this is required to be set by the user. In some cases the action (and other fields as well) cannot be explicitly identified with a single term. Therefore, we propose all fields within this class as multi-value.

E.g. the 'tag.action' could be {*get,access*}, 'tag.subject' – *host* or *user_client*, 'tag.object' – *file* or *web_document*, 'tag.prod_type' – *web_server* and 'tag.service' could be *web*.

- **security** This class provides links to identifier of vulnerability, related to the logged event. For example, 'security.cve' could be "CVE-2010-4369".
- **event_id** This field contains a unique internal id of the event. This should be unique among all generated events in a management system. For our example, we used an SQL database for automatic id generation.

The developed format structure is easy to present, extend and map into database relations. These features simplify the developer's tasks and clarify the semantics of fields with similar names. For example, now the former *appname* and *app.name* fields are easier to distinguish by using *produc-*

*er.appname* and *application.name* as names. Finally, with 107 fields used, we were able to effectively normalise every log message from the dataset. After the normalisation step, most of the attacks could be discovered using simple search queries, as shown in the next section.

## V. The role of the common log format in attack detection

As mentioned earlier, we normalise files before searching for attacks. This pre-processing step includes parsing of logs into described log format with regular expressions[1] and inserting them into SQL database.
The proposed log format allows to detect attacks described in the Table 2 using simple queries, without the usage of a correlation engine or other advanced intrusion detection techniques. The challenge winners on the other part have used self-written scripts and manual analysis [19, 20]. We now provide several use cases to demonstrate the benefits of the proposed log format. E.g. to check for SMTP scans, we use the following query:

```
select * from event where application_protocol = '
    smtp' and tag_status = 'failure'
```

All 220 log lines returned correspond to event 11 from Table 2. These lines include both *sendmail* messages from the mail server ("combo") and *snort* alerts from the logging server ("bastion").
Next, to detect the code injection attempts, we suggest another simple SQL query:

```
SELECT * FROM event WHERE application_cmd LIKE '%␣
    %\%3b%' ESCAPE '\'  AND application_protocol = '
    http' ORDER BY time
```

We search for a semicolon in the URL being processed by any HTTP Server mentioned in the log files[2]. Resulting 20 log lines relate to the code injection cases through awstats.pl. 15 lines correspond to events 3 and 7 from Table 2, two other lines – to event 4. The remaining 3 lines relate to the awstats.pl code injection attempts on 2 and 12 March 2005, not being mentioned in the official challenge results[3].
Obviously, many attacks could not be captured with such simple queries. But the log format structure allows a developer to easily create more sophisticated checks. Please see Table 5 for sample patterns of malicious HTTP events. The proposed object hierarchy of the log format allows to create object-specific patterns (in this example – patterns for elements of HTTP header). This flexibility simplifies creation of rules for matching of malicious events. Now, if we check the normalised log messages with all the patterns selected, we could match 740 HTTP events related to line 10 from Table 2, as well as 35 redirects and 2 cases of calls to the 'libwww' library.

[1] to speedup selection of relevant regular expression for each log line, we store all available regular expressions together with example of corresponding log line in the Lucene[18] index. When new log line comes for normalisation, we search the index for the most relevant regular expression and apply it.

[2] In almost all cases, the semicolon in the URL indicates a malicious event like code injection.

[3] these events were found by the challenge winner [19].

## VI. System architecture

The proposed Object Log Format, as well as attack detection techniques, is already integrated into our own Intrusion Detection System. Security Analytics Lab, as any other IDS, aims to support various log sources by using different import modules. The overview of the logging infrastructure is presented in Figure 2.

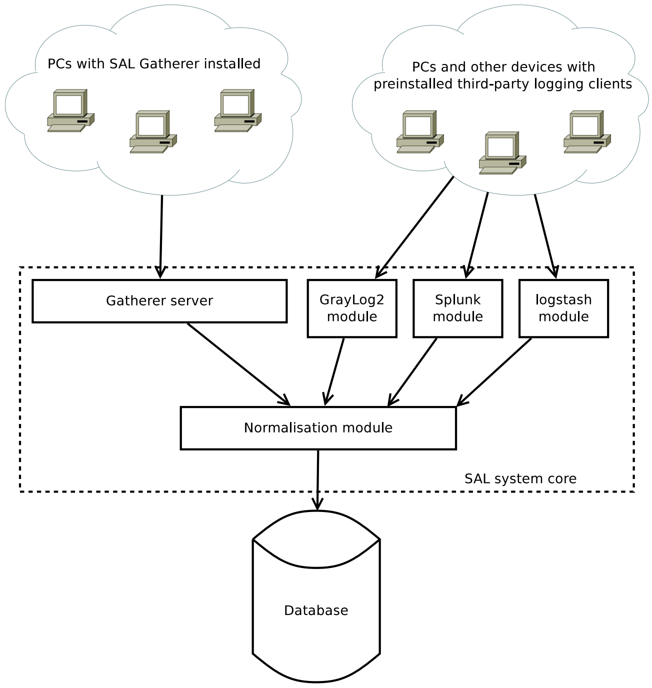


**Figure. 2**: Log gathering architecture of SAL system

At the current system state, we offer modules for direct import from Splunk, GrayLog2 and logstash systems. Besides that, we provide our own log gatherer, that requires an installation of the client on each monitored machine (Linux, Apple MacOS and Microsoft Windows are supported). Compared to direct import modules, SAL Gatherer allows to collect some meta-information in addition to log lines. For example, if the timestamp in the log line does not contain a year, this information could be received from the Gatherer.
Using all these modules, we support gathering logs from a variety of systems and formats, such as Cisco ASA/IOS, SAP NetWeaver, SAP Moonsoon, Windows Events, syslog, etc. Gathered logs are normalised into the proposed Object Log Format and stored in the SQL Database for further analysis.

### A. Gathering real-world data

Although this paper covers only data from "Scan of the Month" Honeynet Challenge, we had an interest to analyse real data as well to prove our concepts in real world. However, installation of a self-developed monitoring system (or at least log gatherers) on the productive system could be complicated due to legal issues. The reasons for this are increased requirements for privacy, fault tolerance and security which could be described as follows:

- **security**. Since all logs are initially collected by domain controllers, the gatherer should be installed there.

| HTTP request header element | Log Format Object | Patterns |
|---|---|---|
| Host | Network.fqdn & File.path | “eval(”, “concat”, “union!+!select”, “(null)”, “base64_”, “/localhost”, “/pingserver”, “/config.”, “/wwwroot”, “/makefile”, “crossdomain.”, “proc/self/environ”, “etc/passwd”, “.exe”, “.sql”, “.ini”, “/.bash”, “/.svn”, “/.tar”, “ ”, “¡”, “¿”, “/=”, “...”, “+++”, “/&&” |
| Content-Location | Application. http. query_string | “?”, “:”, “[”, “]”, “../”, “127.0.0.1”, “loopback”, “%0a”, “%0d”, “%22”, “%27”, “%3b”, “%3c”, “%3e”, “%00”, “%2e%2e”, “%25”, “union”, “input_file”, “execute”, “mosconfig”, “environ”, “scanner”, “path=.”, “mod=.” |
| User-Agent | Application. http. user_agent | ”binlar”, ”casper”, ”cmswor”, ”diavol”, ”dotbot”, ”finder”, ”flicky”, ”jakarta”, ”libwww”, ”nutch”, ”planet”, ”purebot”, ”pycurl”, ”skygrid”, ”sucker”, ”turnit”, ”vikspi”, ”zmeu” |

*Table 5*: HTTP patterns.

However, the domain controller is always the most important part of an IT infrastructure, and therefore has increased security requirements. Hence the installation of the third-party software, like the gatherer, could be a complicated issue.

- **fault tolerance**. Nowadays we have to deal with a large of security logs on the domain controller in a big network. Therefore, collecting and exporting high amounts of security logs could take the resources of the Domain Controller and affect the processing of other tasks.
- **privacy**. Finally, the data such as Domain Controller logs often contain personal information, e.g. user ID, time of logon and logoff events. This information is often an object of data privacy and should be anonymised before exporting for analysis.

To deal with such issues, we create standalone scripts for log export. For example, to extract Windows security events, we provide a script, that anonymises the data related to user privacy and takes care about hardware resources of the Domain Controller. Moreover, the script is a light-weight PowerShell executable, which could be easily checked for security issues and approved by system administration staff.

### B. Visualisation

Security Analytics Lab also provides a web interface for visualisation of logs, as presented in Figure 3. Left pane shows list of hosts in the network; middle pane presents overview of raw log messages, while right pane highlights individual normalised fields of Object Log Format for each message. The search string on the top supports SQL-style queries. Finally, the logging timeline on the bottom provides overview of all available logs and simplifies navigation through them.
Even though the log messages are stored in the Object Log Format, we also keep original log lines for the web interface, since they could be easier to recognise. All queries, however, are executed on normalised data only.

## VII. Future work

Normalisation of events into one common format opens wider capabilities for data analysis, since the data is unified. In our future work, we plan to improve our system and apply various correlation algorithms to such unified data collected from different systems, as described in following subsections.

### A. Testbeds for dataset generation

To collect the data for correlation analysis, we use the following testbeds:

- **Active Directory testbed**, that we created as a virtual network with a Microsoft Windows Domain, where we collect logs of manually generated attacks.
- **Capture The Flag challenges**, that are organised at our institute as a part of teaching process. Within these challenges students create a virtual network with Microsoft Windows, Linux and Cisco IOS virtual machines, as well as network monitoring system. During the attack phase, we collect data from the monitoring system and import them into our database for further analysis.
- **Honeypot network with SAP software**, that we develop as a standalone project. The goal of this project is to gather information about previously unknown vulnerabilities of SAP software. We open Internet access to systems installed within honeypot network and record all attacks performed on them. The recorded data is also normalised and saved in our database.

### B. In-memory correlation and visualisation

The correlation analysis of collected data, even if stored in the normalised form, could be challenging due to high volumes of events to be analysed [21]. To deal with this issue and enable high-speed analytics, we use in-memory SAP HANA database as data storage. The SAP HANA provides an integrated library for machine learning analysis, namely Predictive Analysis Library [22]. In contrast with traditional approach, when the data should be first retrieved from the database, we are able to analyse data directly in the memory of database engine.
The high-speed correlation of events allows us to identify and visualise an attack path within an attacked network. The example of such attack path visualisation (which is currently under development) is presented in Figure 4[4].
Figure 4 shows another interface of Security Analytics Lab, which is provided as Java application. Besides visualisation of attack path and other capabilities[5], in Java interface we support signature-based intrusion detection and offer a dashboard view.

[4] number of log messages and timeline differ from Figure 3, since we used different dataset for this example

[5] similar to the implemented in the web interface described in Figure 3

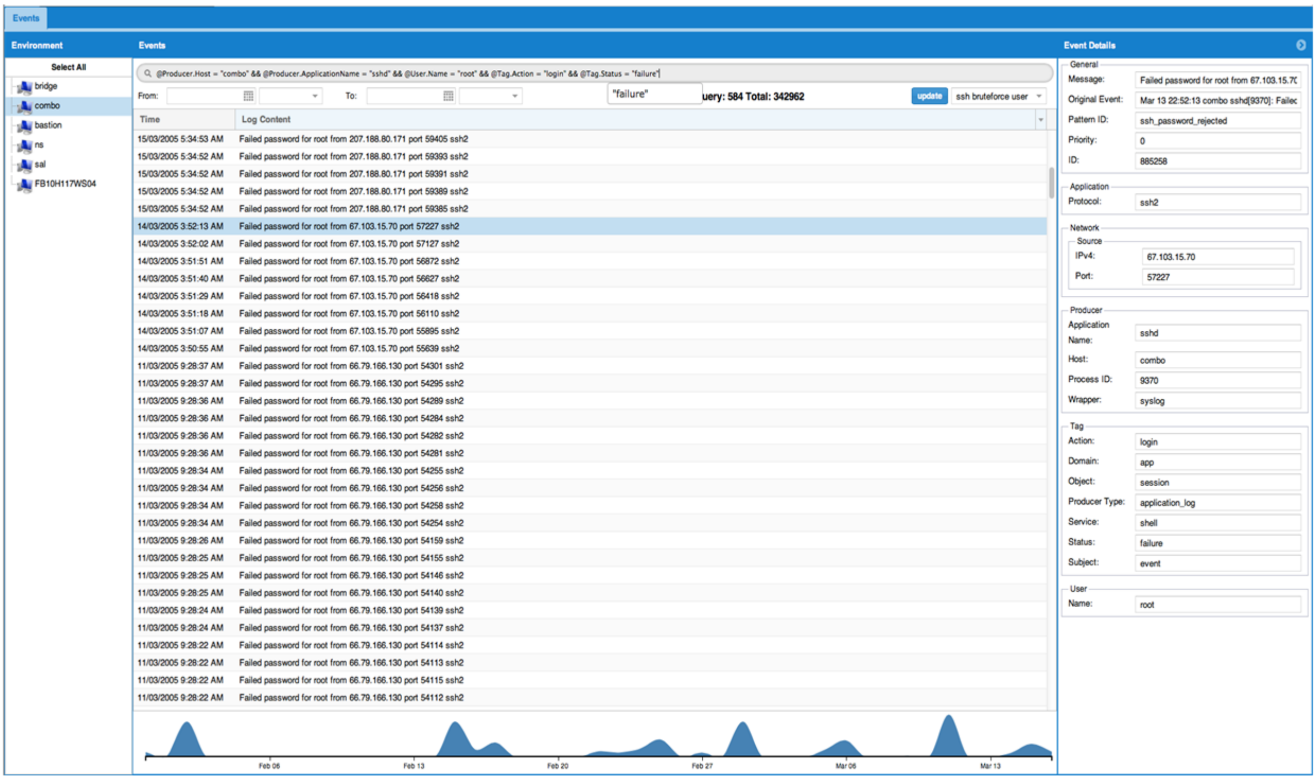


**Figure. 3**: Visualisation of logs in the proposed format within web interface

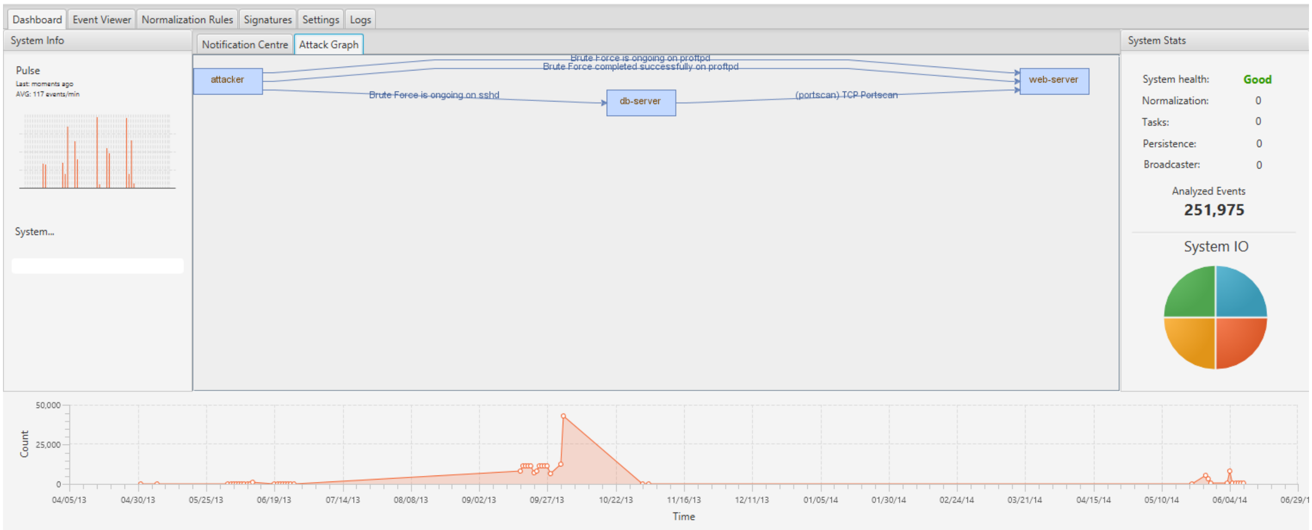


**Figure. 4**: Sample visualisation of attack path

### *C. Vulnerability database for security analysis*

In addition to visualisation of attack path, we plan other features for further development of the Security Analytics Lab, such as connection to vulnerability database and inventory system. The use of inventory system, such as OCS Inventory NG [23] or GLPI [24] together with vulnerability database, namely HPI-VDB [25], allows us to provide information about vulnerable hosts in the network directly in the dashboard of Security Analytics Lab. Moreover, results of vulnerability analysis could be used to construct attack graph and attract the attention of security operator to possible attack vectors.

## VIII. Discussion

In this paper we have shown how the common log format, if thoroughly developed with the regard for specific usage conditions (intrusion detection in our case), could facilitate a lot of operations, including search for attack patterns and correlation of events from different servers. However, the structure of existing log format standards could differ from the one imposed by specific use conditions. On the one hand, common log formats being developed nowadays (CEE [3] and IODEF [6]) try to handle all possible use cases. Such unified approach often needs a manual adoption to be a best fit for a specific use case. On the other hand, log formats,

specially established for intrusion detection sometimes have a limited scope. IDMEF [14] defines the format for intercommunication only (between intrusion detection, response and management systems). Furthermore, CEF [5] has a flat hierarchy, which makes it less flexible in comparison with object-based log formats like CEE [3].
Within our proposed format we try to utilise the strong sides of both approaches and design the flexible object log format that fits the specific purposes, but could be easily extended for generic use cases. However, we do not intend to create a replacement for standards proposed by MITRE [26], IETF [27] and others. Rather, we hope that existed standards could be more flexible to be used for specific purposes and be able to combine extensible structure, light weight and multifarious capabilities, such as search and correlation facilities for intrusion detection systems.

## References

[1] D. Casey. Turning log files into a security asset. *Network Security*, 2008(2):4–7, 2008.

[2] Liu Yang, Pratyusa Manadhata, William Horne, Prasad Rao, and Vinod Ganapathy. Fast submatch extraction using OBDDs. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, ANCS '12, pages 163–174, New York, NY, USA, 2012. ACM.

[3] Common Event Expression White Paper. `https://cee.mitre.org/docs/Common_Event_Expression_White_Paper_June_2008.pdf`, June 2008.

[4] Gerard T. McGuire. The state of security automation standards - 2011. `http://www.mitre.org/work/tech_papers/2011/11_3822/`, November 2011.

[5] ArcSight. Common Event Format. `http://mita-tac.wikispaces.com/file/view/CEF+White+Paper+071709.pdf`, July 2009.

[6] R. Danyliw, J. Meijer, and Y. Demchenko. The incident object description exchange format. IETF RFC5070, December 2007.

[7] BlackStratus LOG Storm. `http://www.blackstratus.com/enterprise/supported-technologies/log-storm/`.

[8] Sawmill log analysis tool. `http://www.sawmill.net/`.

[9] Open Source Host-based Intrusion Detection System. `http://www.ossec.net/`.

[10] Samuel Marchal, Xiuyan Jiang, Radu State, and Thomas Engel. A Big Data Architecture for Large Scale Security Monitoring. In *Proceedings of the 3rd IEEE Congress on Big Data*, pages 56–63. IEEE, July 2014.

[11] Sebastian Roschke, Feng Cheng, and Christoph Meinel. An extensible and virtualization-compatible ids management architecture. In *Proceedings of the 2009 Fifth International Conference on Information Assurance and Security - Volume 02*, IAS '09, pages 130–134, Washington, DC, USA, 2009. IEEE Computer Society.

[12] Anton Chuvakin. Scan 34 - analyze real honeynet logs for attacks and activity. `http://old.honeynet.org/scans/scan34/`, February 2005.

[13] The Honeynet Project. `http://honeynet.org/`.

[14] H. Debar, D. Curry, and B. Feinstein. The Intrusion Detection Message Exchange Format (IDMEF). IETF RFC4765, March 2007.

[15] Common Vulnerabilities and Exposures. `http://cve.mitre.org/`.

[16] Common Weakness Enumeration. `http://cwe.mitre.org/`.

[17] CEE Field Dictionary. `http://cee.mitre.org/language/1.0-alpha/dictionary.html`.

[18] Apache Lucene. `http://lucene.apache.org/`.

[19] M. Richard, M. Ligh, A. Magnusson, S. Seale, and K. Standridge. Project honeynet scan of the month 34. `http://old.honeynet.org/scans/scan34/sols/1/index.html`, May 2005.

[20] C. Kronberg and A. Freeworld. Analysis of the logfiles given in SotM34. `http://old.honeynet.org/scans/scan34/sols/2/proc.pdf`, 2005.

[21] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *J. Network and Computer Applications*, 36(1):16–24, 2013.

[22] Predictive Analysis Library. `http://help.sap.com/hana/SAP_HANA_Predictive_Analysis_Library_PAL_en.pdf`.

[23] Open Computers and Software Inventory Next Generation. `http://www.ocsinventory-ng.org/en/`.

[24] The GLPI Project. `http://www.glpi-project.org/`.

[25] HPI Vulnerability Database. `https://hpi-vdb.de/vulndb`.

[26] The MITRE Corporation. `http://www.mitre.org/`.

[27] Internet Engineering Task Force. `http://www.ietf.org/`.

# Author Biographies

**Andrey Sapegin** is PhD student and Research Assistant at the Hasso Plattner Institute (HPI), University of Potsdam, Germany. He received his Diploma with Honours at the Kaliningrad State Technical University, Russian Federation in 2008. Andrey Sapegin has gathered practical experience while working as a software developer at the Unister GmbH and as a system administrator at the saxess.ag. His research interests include intrusion detection, network security and routing protocols.

**David Jaeger** is currently a PhD Student in the IT-Security Engineering Team at Hasso Plattner Institute (HPI) at University of Potsdam, Germany. He studied IT-Systems-Engineering from 2006 to 2012 at the Hasso-Plattner-Institute and got his B.Sc. degree in 2009 and his M.Sc degree in 2012. His research interests are mainly in the field of intrusion detection, especially in the topics attack monitoring and analytics as well as normalization of security-related information.

**Amir Azodi** has a B.Sc. degree in Communication Networks from Oxford Brookes University and a M.Sc. degree in Information Security from University College London. He is currently a PhD student at the Department of Internet Technologies of the Hasso Plattner Institute. His research interests include Event Normalization, Intrusion Detection, Attack Path Detection and Visualization.

**Marian Gawron** is PhD student and research assistant at the Hasso Plattner Institute (HPI), University of Potsdam, Germany. He received his M.Sc. degree from HPI, University of Potsdam in 2013. During his Master project, he successfully developed and managed the online version of HPI vulnerability database, https://www.hpi-vdb.de. His research interests include intrusion detection, vulnerability databases, and network security.

**Feng Cheng** is now senior researcher at Hasso Plattner Institute (HPI) at University of Potsdam (Uni Potsdam), Germany. His research is mainly focused on network security, firewall, IDS/IPS, security analytics, attack modeling and penetration testing, SOA and Cloud Security, etc. He has published more than 30 papers in the international conferences and Journals. He served as coordinator, program committee member, or reviewer for many international workshops and conferences. Now, he heads the ITSecurity Engineering team and is conducting research, development and teaching activities around new technologies of Network Security at HPI. He holds M.Eng. from Beijing University of Technology (BJUT), and PhD from University of Potsdam (Uni Potsdam).

**Christoph Meinel** is a CEO and a Scientific Director of the Hasso Plattner Institute for IT-Systems Engineering (HPI) and Professor for Internet-Technologies and Systems at the University of Potsdam.
Christoph Meinel is full professor for computer science at HPI and the University of Potsdam, holding a chair in "Internet Technologies and Systems". His research focuses on Future Internet Technologies, in particular Internet and Information Security, Web 3.0, Semantic-, Social- and Service-Web, as well as on innovative Internet applications, especially in the domains of e-Learning and Telemedicine. Besides this, he is scientifically active in the field of Innovation Research and Design Thinking.