# Threat Analysis using Vulnerability Databases – Topic Model Analysis using LDA and System Model Description –

Katsuyuki Umezawa
*Department of Information Science*
*Shonan Institute of Technology*
Fujisawa, Kanagawa 251–8511, Japan
e-mail: umezawa@info.shonan-it.ac.jp

Yusuke Mishina
*Cyber Physical Security Research Center (CPSEC)*
*National Institute of Advanced Industrial Science and Technology (AIST)*
Koto-ku, Tokyo 135–0064, Japan
e-mail: yusuke.mishina@aist.go.jp

Sven Wohlgemuth
*Research & Development Group*
*Hitachi, Ltd.*
Yokohama, Kanagawa 244–0817, Japan
e-mail: sven.wohlgemuth.kd@hitachi.com

Kazuo Takaragi
*Cyber Physical Security Research Center (CPSEC)*
*National Institute of Advanced Industrial Science and Technology (AIST)*
Koto-ku, Tokyo 135–0064, Japan
e-mail: kazuo.takaragi@aist.go.jp

***Abstract*—We proposed a threat analysis method utilizing topic model analysis and vulnerability databases. The method is based on attack tree analysis. We create an attack tree on a evaluation target system and some attack trees on a known vulnerability, and combine the two types of attack trees to create more concrete attack trees. This enables us to calculate the probability of occurrence of a safety accident and to utilize attack trees in future analysis. In this paper, we formulate a topic model analysis and confirm the feasibility of matching known attack cases to vulnerability databases using a topic model analysis tool. In addition, we show that our proposed method can use the results of past threat analysis for the next one. Moreover, we create a system model description based on the attack tree of Tesla's case created using our proposed method. It shows that fake commands can be transmitted from the external information system to the in-vehicle control system. Our approach to automatic threat analysis supports risk analysis in discovering previous unknown relationships and so threats including their potential escalation within an connected IT system.**

***Keywords–Threat Analysis; Vulnerability Information; Attack Tree; Topic Model Analysis; System Model Description.***

## I. INTRODUCTION

We proposed a threat analysis method utilizing topic model analysis and vulnerability databases [1]. The background of this proposal is as follows.

Interference and interruption to safety due to security incidents are recognized as a big problem in safety critical systems, such as those for electric power, information communication, automobile, aviation, railway, and medical care. For security of in-vehicle communication in the EVITA project [2], authors have conducted a risk management process. Specifically, risk analysis, security requirement setting, architecture design, prototyping, and demonstration was held. The EVITA project uses attack trees for risk analysis. One way to analyze the causal relationship between safety (hazard) and security (threat) is to express that relationship with a combination of a Fault Tree (FT) and Attack Tree (AT) [3]. The US-based MITRE Corporation provides several tools for vulnerability reporting and aggregation in a vulnerability database (DB). In Common Vulnerabilities and Exposures (CVE) [4], individual software vulnerabilities are stored in a DB. In Common Weakness Enumeration (CWE) [5], common vulnerabilities are cataloged with a focus on the cause of the vulnerability. Furthermore, Common Attack Pattern Enumeration and Classification (CAPEC) [6] is a DB classified by attack pattern. Scientific literature related to safety analysis using FTs is, nowadays, mature. However, the complexity of the problem has significantly increased in security analysis. Elaborate attacks occur with multiple combinations of those vulnerabilities. It is not easy to create an AT that comprehensively captures such possibilities.

We have focused on such problems and proposed a threat analysis method using a vulnerability DB as a practical approach [7][8]. First, we assumed that many attacks were imitations or minor changes of known attacks. Therefore, we believed that expressing attack cases that occurred in the past by using an AT could enable a designer (defender) to become aware of related attacks (recognize the danger). By gradually and continuously applying this approach, it can be useful for reducing vulnerability.

We proposed an algorithm that includes a process for matching each node of an AT described in natural language [7][8]. However, the matching method utilized was not specified. We evaluated the feasibility of this unspecified matching process using a topic model analysis method. In this paper, in addition to our study, we add the formulation of our proposed algorithm, application to examples of a Tesla case [9], the formulation of a topic model, and the possibility of recognizing dangers by using the system model description.

In Section II, we summarize the threat analysis method we proposed in our previous studies [7][8]. In Section III, we formulate the algorithm shown in Section II. We apply the proposed algorithm to the Tesla case examples in Section IV. In Section V, we introduce the topic model analysis. In Section VI, we verify the feasibility of matching attack cases to vulnerability DBs and show the result. We describe a system model description using the attack tree created from the proposed algorithm in Section VII. Section VIII concludes this paper by summarizing the key points and providing an outlook on future activities.

## II. THREAT ANALYSIS USING VULNERABILITY DATABASES

This section presents a summary of our proposed method [8]. An overview of the threat analysis method using the vulnerability DB is shown in Figure 1. The proposed threat analysis method conducts the following three procedures:

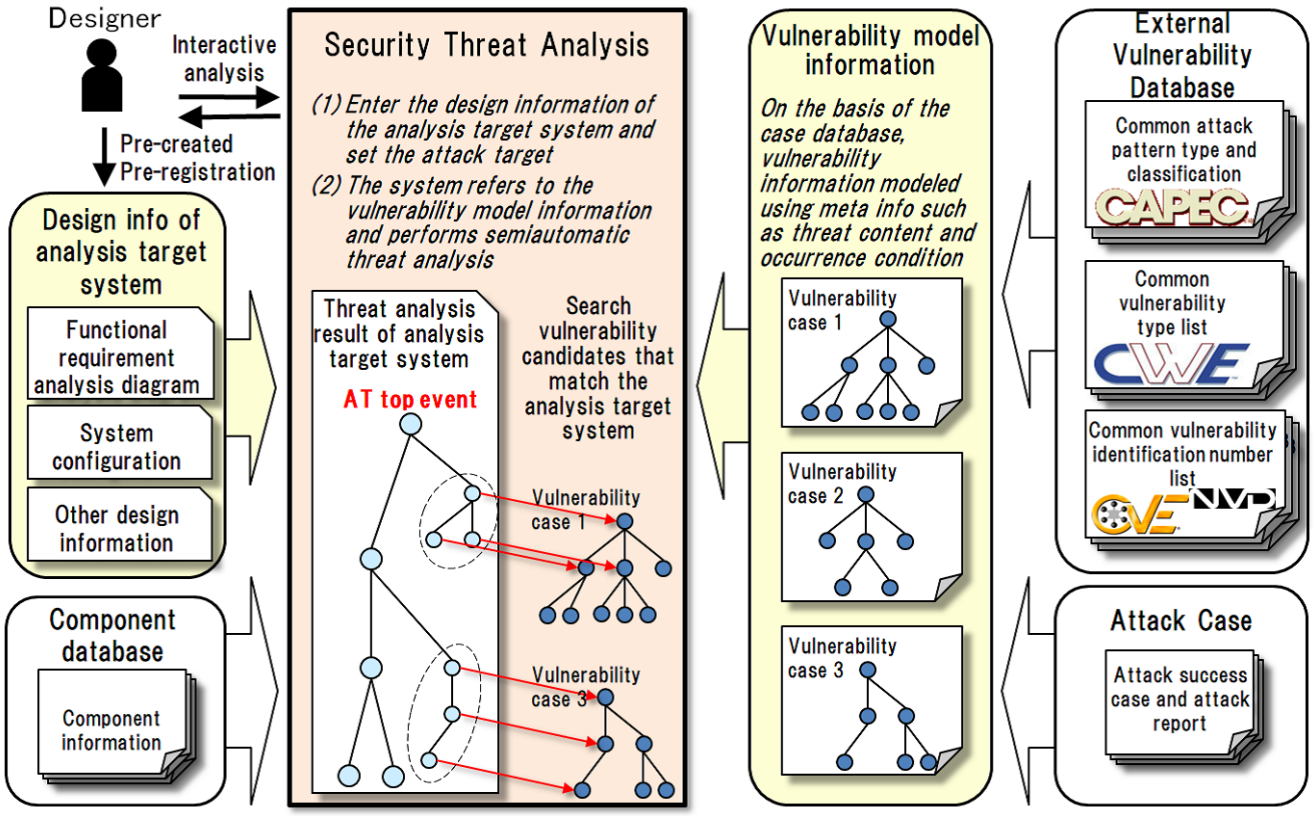


Figure 1. Overview of proposed threat analysis method

- Create vulnerability model information.
- Create lower-level component information embedded in software.
- Perform threat analysis on the basis of design information of analysis target system.

### A. *Creating vulnerability model information*

The MITRE Corporation has published several forms of vulnerability DBs [4]–[6]. However, it is difficult to create an AT for a concrete target (for example, a connected car) simply by referring to these DBs, because elaborate attacks occur with multiple combinations of vulnerabilities. We will create an AT with a reference to existing attack case literature, reports, etc. Thus, let the AT be obtained from the existing vulnerability DB and existing attack report be called the first_AT. This first_AT is hierarchically drawn into a top node, a collection of intermediate nodes and bottom nodes. A single first_AT is created for each vulnerability. A vulnerability DB such as CVE monotonically increases, so it is not necessary to recreate the first_AT once it has been generated.

### B. *Proposal of component database*

In some configurations of embedded systems, such as those for automobiles and in general the Internet of Things (IoT), required lower-level components embedded within the software, not the software itself, are incorporated. However, a vulnerability DB such as CVE only includes vulnerability information for software and does not describe information on the lower-level components embedded within the software. Therefore, a correspondence table between the software version and the version for its lower-level components would be beneficial. This makes it easy to check vulnerability information at the manufacturing stage of embedded systems such as those in IoT devices.

Specifically, using Tesla's browser hacking case as an example, the "UserAgent" property defined in Tesla's browser is "Mozilla / 5.0 (X11; Linux) AppleWebKit / 534.34 (HTML, like Gecko) QtCarBrowser Safari / 534.34." In contrast, the related vulnerability outlined in CVE describes "Google Chrome before 16.0.912.77". In this case, Chrome itself is not used, but the WebKit component built into Chrome is used. Therefore, a correspondence table indicating the version of the component built into certain software is required. Table I shows an example of Google Chrome's component DB. The method to create a component DB is outside the scope of this proposal.

TABLE I. Example of Google Chrome's component DB

| Version | Release date | Layout engine |
|---|---|---|
| 0.2.149 | 2008-09-02 | WebKit 522 |
| 0.3.154 | 2008-10-29 | WebKit 522 |
| 0.4.154 | 2008-11-24 | WebKit 525 |
| ... | ... | ... |
| 10.0.648 | 2011-03-08 | WebKit 534.16 |
| 11.0.696 | 2011-04-27 | WebKit 534.24 |
| 12.0.742 | 2011-06-07 | WebKit 534.30 |
| 13.0.782 | 2011-08-02 | WebKit 535.1 |
| 14.0.835 | 2011-09-16 | WebKit 535.1 |
| ... | ... | ... |
| 56.0.2924 | 2016-12-08 | Blink 537.36 |

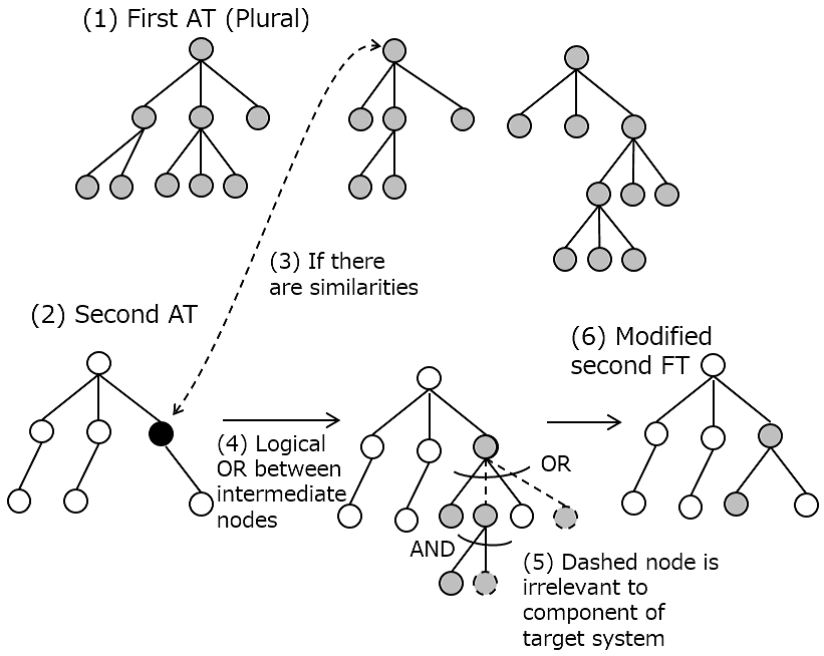


Figure 2. Threat analysis algorithm (cited from reference [8])

### C. Threat analysis algorithm

This section describes the threat analysis algorithm. It corresponds to the "Safety & Security Threat Analysis" section in Figure 1. The algorithm, which is based on the vulnerability model shown in Section II-A, the component DB shown in Section II-B, and the design information of the analysis target system, is as follows:

(1) Create a second_AT with the top node as a safety accident related to the evaluation target system. At this time, even if the component is not directly included in the evaluation target system, a component judged to be related by referring to the component DB is included in the second_AT (the black circle node in Figure 2 (2)). The second_AT is hierarchically depicted using the top node, the multiple intermediate nodes, and the lowest nodes. Thus, a second_AT is created (Figure 2 (2)).

(2) One of the top nodes or intermediate nodes of the second_AT is selected and Natural Language Processing (NLP) is used to mechanically determine whether there are first_ATs having a natural language expression similar to nodes of the second_AT (Figure 2 (3)). If this is the case, the first_AT is temporarily added to the second_AT (Figure 2 (4)). OR gate is attached to the node of the second_AT temporarily, and the first_AT is attached below it. This is done for all nodes of the second_AT. As a result, the second_AT is expanded more after considering the existing vulnerability database, that is, the entire set of the first_AT.

(3) The focus is now on the temporary added nodes in the expanded second_AT. We check whether the added node is necessary. Specifically, we define a node unrelated to the component of the second_AT (different components or different versions) as FALSE nodes, and the FALSE node and the AND gate that is just above the FALSE node are deleted (Figure 2 (5)).

(4) Repeat steps 1–3 for all the first_ATs that are related to the second_AT as described above. After the modification, we evaluate the occurrence probability of the top node by using the modified second_AT.

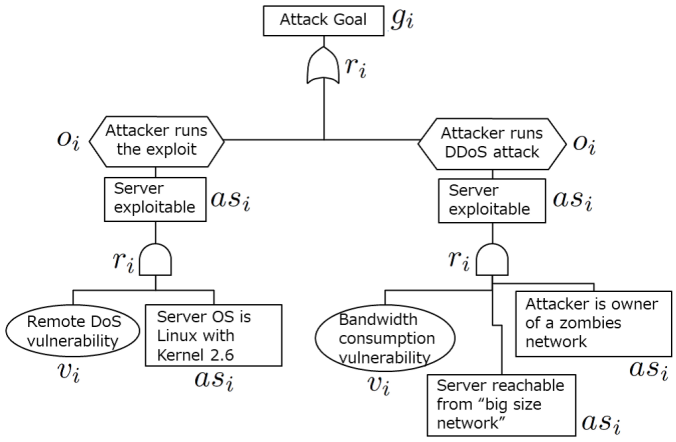


Figure 3. Example of AT (quoted from Figure 2, cited from reference [3])

## III. Formulation of Proposed Algorithm

In this section, we formulate the algorithm shown in Section II-C.

### A. Definition

The definition of the attack tree AT according to reference [3] is shown below. An example of AT is shown in Figure 3.

$$\boldsymbol{G} = \{g_i\} : AttackGoals \quad (1)$$
$$\boldsymbol{O} = \{o_i\} : Operations \quad (2)$$
$$\boldsymbol{AS} = \{as_i\} : Assertions \quad (3)$$
$$\boldsymbol{V} = \{v_i\} : Vulnerabilities \quad (4)$$
$$\boldsymbol{R} = \{r_i\} : Relationships \quad (5)$$

Here, an attack goal is the goal of all potential cyber attacks, and operations represent all the basic actions (reads, writes, etc.) that can be performed by either the attacker or the operator of the system. An assertion is a statement (for example, "Web server is not patched") representing " conditions to be verified" in order to take account of the actual branching of the attack tree. Vulnerability is a known vulnerability. Relationships are relationships that exist between elements that make up an attack tree (that is, attack goals, operations, assertions, vulnerabilities). The attack tree $AT_k$ is defined as follows.

$$AT_k = \{g_i, \boldsymbol{O}_i, \boldsymbol{AS}_i, \boldsymbol{V}_i, \boldsymbol{R}_i\} \quad (6)$$

Here, $g_i \in \boldsymbol{G}, \boldsymbol{O}_i \subseteq \boldsymbol{O}, \boldsymbol{AS}_i \subseteq \boldsymbol{AS}, \boldsymbol{V}_i \subseteq \boldsymbol{V}, \boldsymbol{R}_i$ is a set of relationships.

All ATs have one main goal $g$, and the logic gate output (upper side) becomes an assertion.

### B. Formulation of proposed algorithm

The first attack tree $AT_k^1$ and the second attack tree $AT^2$ are defined as follows.

$$AT_k^1 = \{g_k, \boldsymbol{O}_k, \boldsymbol{AS}_k, \boldsymbol{V}_k, \boldsymbol{R}_k\} \quad (7)$$
$$AT^2 = \{g_j, \boldsymbol{O}_j, \boldsymbol{AS}_j, \boldsymbol{V}_j, \boldsymbol{R}_j\} \quad (8)$$

Next, look for $k$, which is $g_j = g_k$ or $as_l \approx g_k$. However, it is $as_l \in \boldsymbol{AS_j}$. In addition, look for $n$ and $m$, which is $as_n \approx as_m$. However, $as_n \in \boldsymbol{AS_k}, as_m \in \boldsymbol{AS_j}$.

Here, $x \approx y$ means "Comparing the descriptions of both sides with words, it is judged that $x$ and $y$ are close."

Next, update the second attack tree $AT^2$ as follows.

$$AT^2 = \{ \; g_j, \boldsymbol{O}_j \cup \boldsymbol{O}_k \cup \boldsymbol{O}_n, \boldsymbol{AS}_j \cup \boldsymbol{AS}_k \cup \boldsymbol{AS}_n, \boldsymbol{V}_j \cup \boldsymbol{V}_k \cup \boldsymbol{V}_n, \boldsymbol{R}_j \cup \boldsymbol{R}_k \cup \boldsymbol{R}_n \backslash \boldsymbol{R}' \} \quad (9)$$

Here, $\backslash$ represents the difference set. Also, $\boldsymbol{R}'$ is $\boldsymbol{R}' = \boldsymbol{R}'_{OR} \cup \boldsymbol{R}'_{AND}$. $\boldsymbol{R}'_{OR}$ is the relationship of the FALSE node, and $\boldsymbol{R}'_{AND}$ is the relationship of the upper nodes of the AND relationship just above the FALSE node. A FALSE node is $o \in \boldsymbol{O}_k \cup \boldsymbol{O}_n, as \in \boldsymbol{AS}_k \cup \boldsymbol{AS}_n, v \in \boldsymbol{V}_k \cup \boldsymbol{V}_n$ that is unrelated to the components of $AT^2$ (such as different components and different versions).

### C. Calculation of attack probability

According to the formulation in the previous section, it is possible to calculate the probability of attack with the following formula using the calculation method of the conventional research [3].

If the inputs to the logic gates are independent, the probability of the output value from the $i$th AND gate $P_{out}AND_i$ and the probability of the output value from the $i$th OR gate $P_{out}OR_i$ are as follows.

$$P_{out}AND_i = \prod_{k=1}^{n} P_{in}(k,i) \quad (10)$$
$$P_{out}OR_i = \sum_{k=1}^{n} P_{in}(k,i) \quad (11)$$

However, $P_{in}(k,i)$ is the probability of the input of the $k$th input to the $i$th gate with $n$ inputs ($1 \leq k \leq n$).

In addition, in reference [3], calculation formulas when the inputs to the logic gates are not independent are also shown. Furthermore, reference [3] suggests rewriting the operation node with an AND gate and an assertion in order to obtain the probability of the top event (attack goal) of the attack tree. Specifically, replace the operation node with an AND gate, substitute the description of the original operation as an assertion, and input it as an input to the AND gate. With this replacement, the description of the operation disappears from the attack tree, and the probability of the top event can be calculated by sequentially calculating the above expression (10) and expression (11). In addition, reference [8] describes the application of actual cases of car attacks [9][10].

## IV. Application of Proposed Method to Actual Case

In this section, we apply the proposed algorithm to examples in the Tesla case [9].

### A. Creating first_AT

First, a first_AT is created on the basis of the vulnerability DB. We must create first_ATs for all vulnerabilities. In this case, we created a first_AT for CVE-2011-3928 as shown in Figure 4.

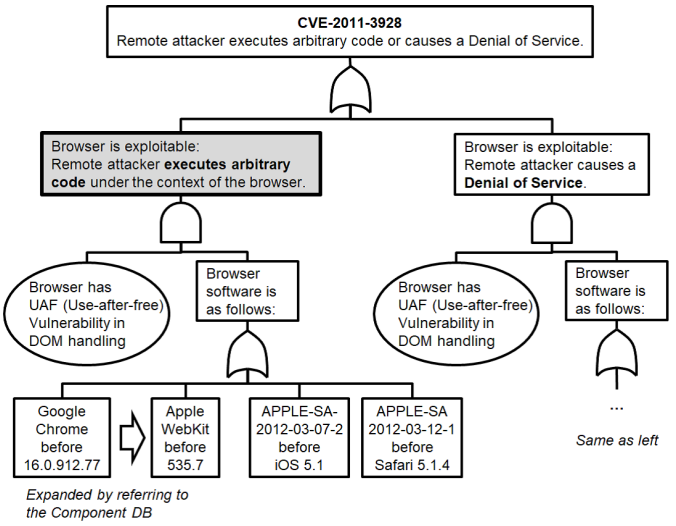


Figure 4. first_AT created from CVE-2011-3928 (cited from reference [8])

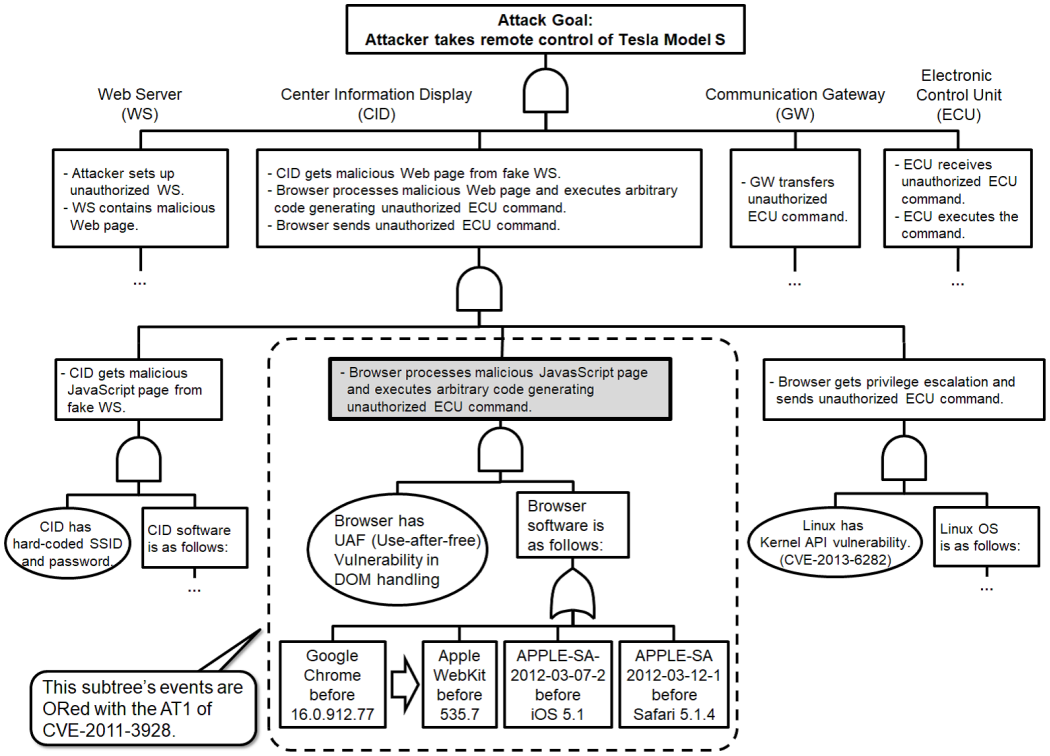


Figure 5. second_AT created from Tesla model S case

### B. Application to case of Tesla model S

Next, we attempted to create a second_AT for the Tesla model S case [9]. The second_AT we created is shown in Figure 5. The dashed-line area of Figure 5 is the resultant subtree of combining the second_AT with the first_AT shown in Figure 4.

To apply our method to this case, matching of the nodes shown in gray in Figure 4 and Figure 5 was performed manually by humans. The next section will outline how this matching process will be performed automatically using topic model analysis.

## V. Topic Model Analysis

In this section, we describe latent Dirichlet allocation (LDA), which is a method of topic model analysis and cosine similarity.

### A. LDA

Topic models are formed under the notion that a document contains a number of latent topics, with each keyword either attributing to a certain topic or being generated as a result of said topic. In topic model analysis, we estimate latent topics from keywords. One of the analysis methods of topic models is LDA [12]. This is a language model that assumes the probability distribution of the topic (parameter $\theta$ of the multinomial distribution) follows the Dirichlet distribution. In LDA, topics are selected in accordance with the Dirichlet distribution and words are selected in accordance with the probability distribution of words for that topic.

### B. Formulation of LDA

The LDA can be represented by the graphical model shown in Figure 6.

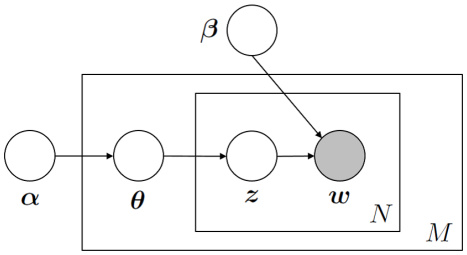


Figure 6. Graphical model representation of LDA (cited from reference [12])

Here, $d$ is the number ID of a document, $n$ is the number ID of a word in a document, and $k$ is the number ID of a topic. $M$ is the number of documents and $N$ and $K$ are the number of words and topics in a document, respectively. The number of words in document $d$ is represented as $N_d$. The ranges of d, n, and k are $1 \leq d \leq M$, $1 \leq n \leq N_d$, and $1 \leq k \leq K$, respectively. $w_{dn}$ represents the $n$th word of document $d$. $z_{dn}$ represents the latent topic of the $n$th word in the document $d$. $\theta_{dk}$ is the mixing ratio of the latent topic $k$ of document $d$. For example, if the number of topics of document $d$ is 3, the mixing ratios of topics 1, 2, and 3 are 10%, 70%, and 20%, respectively then $\theta_{d1} = 0.1$, $\theta_{d2} = 0.7$, $\theta_{d3} = 0.2$, in which $\boldsymbol{\theta}_d = \{0.1, 0.7, 0.2\}$. $\boldsymbol{\alpha}$ is a hyper parameter related to the mixing ratio of the latent topic, and $\boldsymbol{\beta}$ is one related to the word generation rate. A set of documents is called a

corpus, and a corpus of $M$ documents is denoted by $\boldsymbol{D} = \{\boldsymbol{w_1}, \boldsymbol{w_2}, \cdots, \boldsymbol{w_M}\}$. We obtain the probability of a corpus as follows:

$$p(\boldsymbol{D}|\boldsymbol{\alpha}, \boldsymbol{\beta}) = \prod_{d=1}^{M} \int p(\boldsymbol{\theta}_d|\boldsymbol{\alpha}) \left( \prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\boldsymbol{\theta}_d) p(w_{dn}|z_{dn}, \boldsymbol{\beta}) \right) d\boldsymbol{\theta_d} \quad (12)$$

Here, we can only observe the word $w_{dn}$. We want to know from which topic those words were generated. In other words, when document $d$ is given, we calculate the probability distribution of $\boldsymbol{\theta}_d$ and $\boldsymbol{z}$ as follows:

$$p(\boldsymbol{\theta}, \boldsymbol{z}|\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{p(\boldsymbol{\theta}, \boldsymbol{z}, \boldsymbol{w}|\boldsymbol{\alpha}, \boldsymbol{\beta})}{p(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})} \quad (13)$$

The variable Bayes method [12], Markov chain Monte Carlo (MCMC) [13], Gibbs sampling method [14], which is one kind of MCMC, etc. are proposed as a method to solve the above equation.

### C. Cosine similarity

We can compute the similarity between document $d$ and $d'$ by calculating the cosine similarity for $\boldsymbol{\theta}_d$ obtained approximately by the above LDA method. The cosine similarity can be expressed by the following equation.

$$sim_{cos}(d, d') = \frac{\sum_{k=1}^{K} \theta_{dk} \theta_{d'k}}{\sqrt{\sum_{k=1}^{K} {\theta_{dk}}^2} \sqrt{\sum_{k=1}^{K} {\theta_{d'k}}^2}} \quad (14)$$

### D. Topic model analysis tool

The National Institute of Advanced Industrial Science and Technology (AIST) has developed a security requirement analysis support tool using topic model analysis technology including LDA [15]. We preliminarily used this tool to verify whether the vast number of vulnerabilities CVE [4] listed in the order of discovery can be organized into a hierarchical structure by topic model analysis. Figure 7 shows the result of using 1500 cases from CVE-2011-3001 to CVE-2011-4500 after translating it to Japanese using Google Translate [16]. As shown in Figure 7, we see that similar vulnerabilities are classified near the hierarchical structure. Figure 7 is written in Japanese, the boxes in CVE-2011-3017 to CVE2011-3077 are described as "Use-after-free vulnerability in Google Chrome before xx.0.xxx.xx allows remote attackers to cause ... or possibly have unspecified other impact ...".

## VI. Matching Attack Cases to Vulnerability Database

In this section, we describe the method and results of experiments that match attack cases and vulnerability databases.

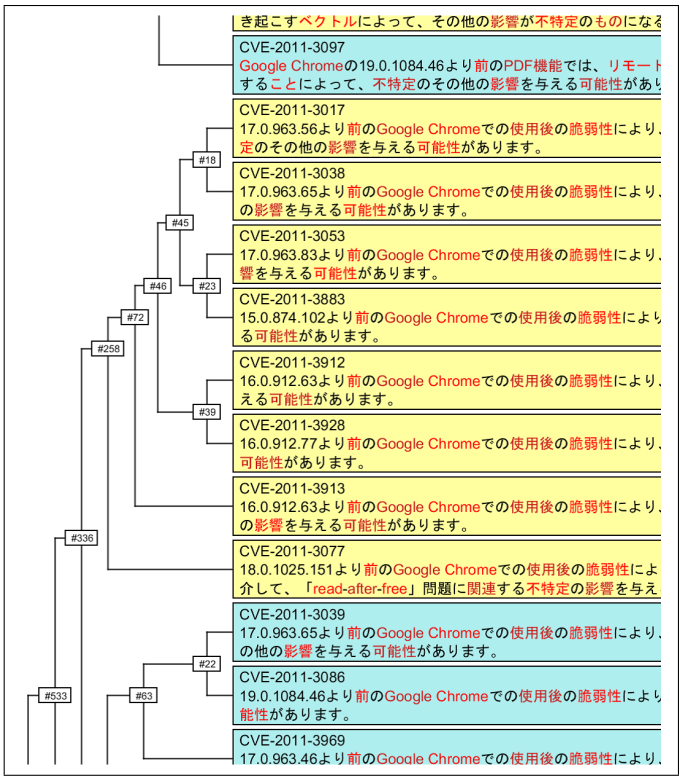


Figure 7. Example segment of vulnerability DB CVE hierarchy

### A. Outline explanation

As mentioned in Section II-C(2), we used NLP when matching and connecting the first_AT and the second_AT nodes. We verified the feasibility of this matching process.

We have investigated on various reports to find vulnerabilities that should be related in the second_AT of the target system. However, depending on the report, the procedure of attack is shown but the concrete CVE number is not specified. Even in such a case, we can extract the corresponding CVE number from the attack description described in natural language.

To achieve this, we must find a node of the second_AT that conceivably matches the description in CVE. However, a mechanical word matching process will probably not lead to a correct result as it is dependent on the words used to describe sentences. The context or meaning of the known attack description in each report should be thoroughly examined. Therefore, we have focused on the sentences of existing papers. Specifically, we have focused on the actual case of a car attack [9]. The process flow is as follows.

We translated the paper [9] into Japanese by using Google Translate because the tool we used only corresponded to Japanese. An advantage of utilizing such a translation is that it can prevent notation fluctuation of terms. The impact of Google translation will be discussed in the appendix. Since the section on BROWSER HACKING is long and its content is related to two vulnerabilities, it was divided into two. The vulnerabilities in question were CVE-2011-3928 and CVE-2013-6282. CVE-2011-3928 is described in the section on BROWSER HACKING, and CVE-2013-6282 is described in

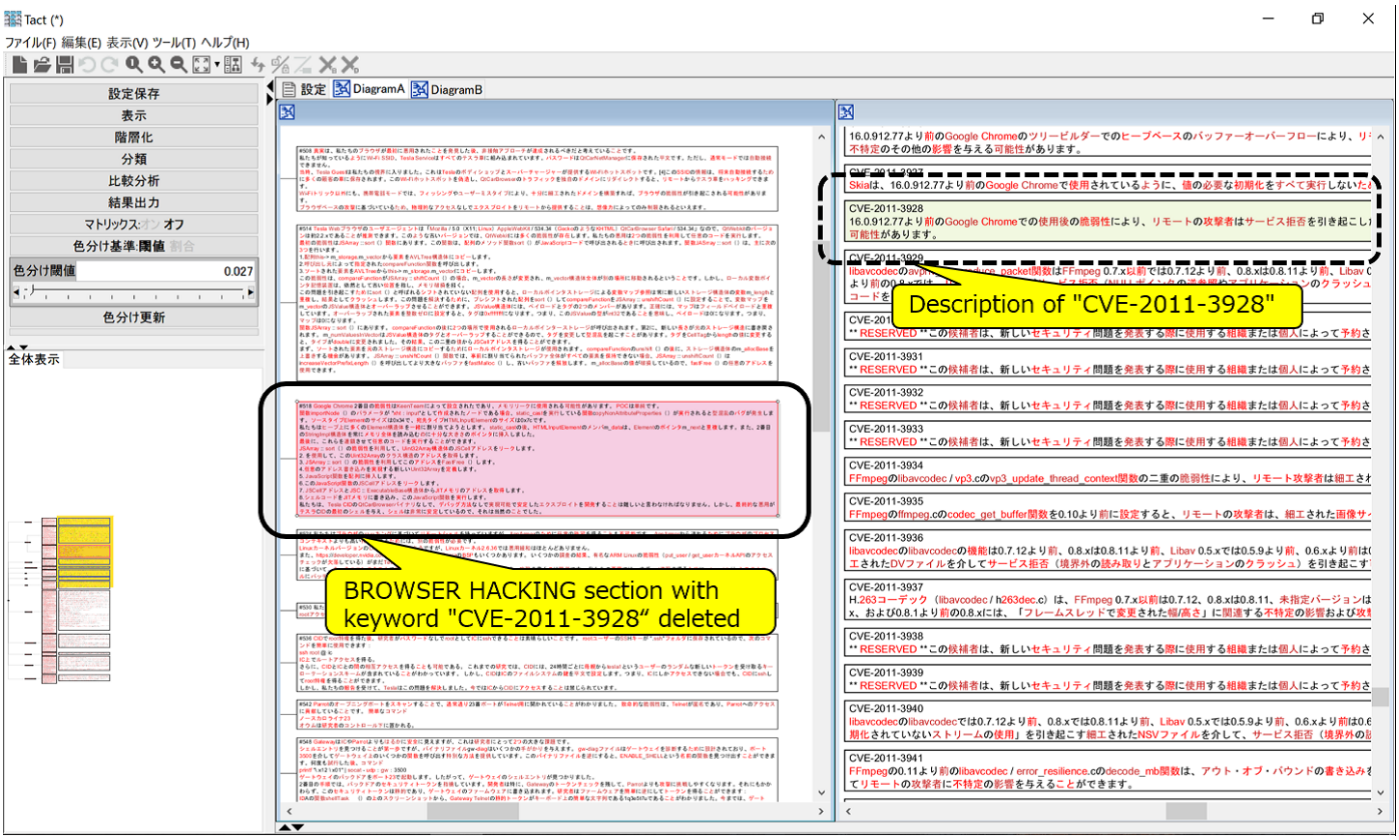


Figure 8. Matching attack cases to vulnerability DBs

the section on LOCAL PRIVILEGE ESCALATION. If "CVE-2011-3928" or "CVE-2013-6282" is included as a keyword, it may be detected by keyword matching, so the keywords "CVE-2011-3928" and "CVE-2013-6282" were deleted from BROWSER HACKING and LOCAL PRIVILEGE ESCALATION, respectively.

However, regarding BROWSER HACKING, there is a problem of component inclusion relationship stated in Section II-B, and the keyword "Google Chrome" is added to the sentences in which WebKit is described. This is considered to be equivalent to referring to the component DB of the proposed method. Since the topic analysis tool used has an upper limit on the number of items to be handled, it was not possible to cover all CVEs, so we targeted 500 items before and after including the target vulnerability. The limitation of 500 items is not a constraint of the topic model analysis, but an implementation limitation of the tools we used.

We specifically targeted CVEs from CVE-2011-3501 to CVE-2011-4000 including CVE-2011-3928 and those from CVE-2013-6001 to CVE-2013-6500 including CVE-2013-6282. For each section of the paper and each CVE vulnerability, similar sentences were evaluated by topic model analysis. The keyword extraction method was known as "noun and Kana", the feature quantity extraction method was "LDA", and the sentence similarity "Cosine" option was used.

### B. Analysis result

The result of matching each section of the paper to each CVE vulnerability is shown in Figure 8. Figure 9 shows the enlarged view of BROWSER HACKING section of Figure 8, and Figure 10 shows enlarged view of the description of "CVE-2011-3928" in Figure 8. When we click on a sentence in the left pane, this tool will highlight similar sentences in the right pane. The solid lined area in the left pane is the BROWSER HACKING section with the keyword "CVE-2011-3928" deleted. When clicking on this area, the dashed lined area, which is the description of CVE-2011-3928 in the right pane, is highlighted and is judged to be similar. The number of items that included the appropriate CVE from the original 500 was filtered down to 22. It can be said that the smaller the number, the better. Regarding CVE-2013-6282, a similar result was obtained by matching the information of LOCAL PRIVILEGE ESCALATION with that of CVE, in this case 23 out of the 500.

### C. Consideration of topic model analysis

In this research, we used a topic model, which treats a document as a set of words, and replaces distances between different documents with distances between different "sets of words" to measure proximity. Topic model analysis is one of the so-called AI methods that enables high-speed processing of big data and excels in clarifying the reason for the results. For example, the number of pages in the Tesla attack case paper [9], IRB 140 industrial robot attack case paper [11], and Jeep Cherokee attack cases paper [10] are 16, 48, and 91, respectively. Manually analyzing such a large amount of materials requires a great, if not greater, amount of work, placing a heavy burden one those doing it. Furthermore,

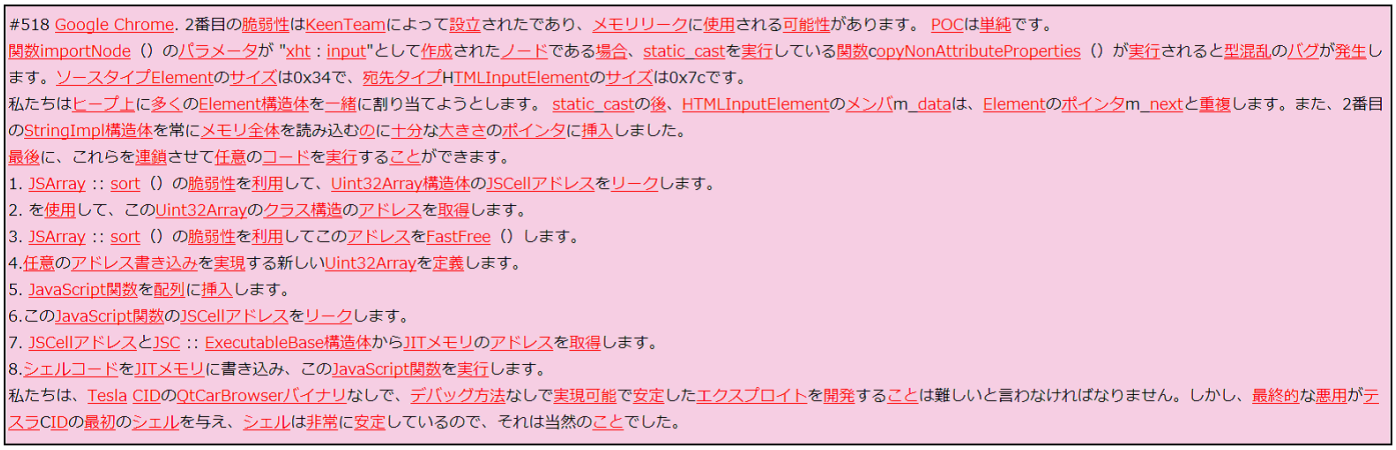


Figure 9. Enlarged view of BROWSER HACKING section shown in Figure 8

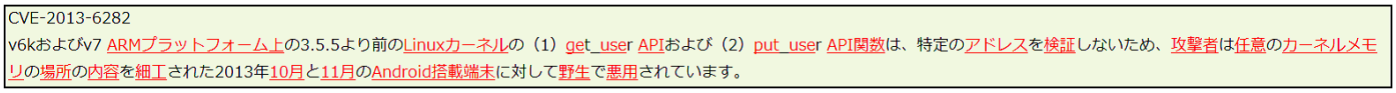


Figure 10. Enlarged view of “CVE-2011-3928” description shown in Figure 8

considering that the number of attack papers will increase in the future, the proposed method of processing attack papers automatically with a topic model is significant.

## VII. System Model Description

In this section, we create a system model description using the second attack tree shown in Figure 5 created using the proposed algorithm. This shows that we can recognize the dangers of related attacks using the description. We analyze the input and output of messages in the nodes directly under the attack goal node “Attacker takes remote control of Tesla Model S” in Figure 5 using the system model description. As a result, we achieve transparency in escalation of fake commands from an external information system to the in-vehicle control system.

### A. Date flow diagram

A system model description of Tesla’s threat analysis target system is shown in Figure 11, which is the description result of the data flow diagram used for safety verification. The system to be analyzed consists of four hardware blocks, five software function modules, an operating system (OS), Web browser, Web server (WS), and various networks. The electronic control unit (ECU), communication gateway (GW), and center information display (CID) of the hardware block are installed inside the vehicle. The CID of the vehicle is connected to the WS over the Internet via Wi-Fi.

First, the in-vehicle control system will be described. The CID executes the vehicle information display function module, which inputs an instruction from the driver and creates a corresponding vehicle control command (for example, a command for monitoring the operating state of the engine, a command for unlocking the door, etc.) to the GW. The vehicle communication control function module of the GW converts the received command into an ECU command and transmits it to the ECU via the CAN bus. The vehicle control function module of the ECU activates the control logic inside the unit corresponding to the received ECU command and executes the vehicle control instructed by the driver.

Next, the external information system will be described. The Web information display function module of the CID inputs a driver’s instruction, creates a necessary Web page request command (for example, a GET command of the HTTP protocol), and transmits it to the WS via Wi-Fi. The Web server function module of the WS searches the corresponding Web page in accordance with the received command and returns the search result (for example, HTML content or JavaScript code).

At this stage, a safety analysis is carried out assuming that there is no vulnerability. Figure 11 shows that the in-vehicle control system and the external information system are separate systems, and there is no interference with each other.

### B. Add vulnerability

Figure 12 expands the data flow diagram shown in Figure 11 to include the possibility of attack by adding Input / Output by vulnerability attributes as depicted by the gray boxes and dotted arrows.

As shown in Figure 5, the CID vulnerability is established by satisfying three conditions: “CID gets malicious JavaScript page from fake WS,” “Browser processes malicious JavaScript page and executes arbitrary code generating unauthorized ECU command,” and “Browser gets privilege escalation and sends unauthorized ECU command.” The first condition is established by “CID has hard-coded SSID and password.” The second condition applies to both “Browser has UAF (Use-after-free) Vulnerability in DOM handling” and “Apple WebKit before 535.7.” The third condition is established by “Linux has Kernel API vulnerability (CVE-2013-6282).”

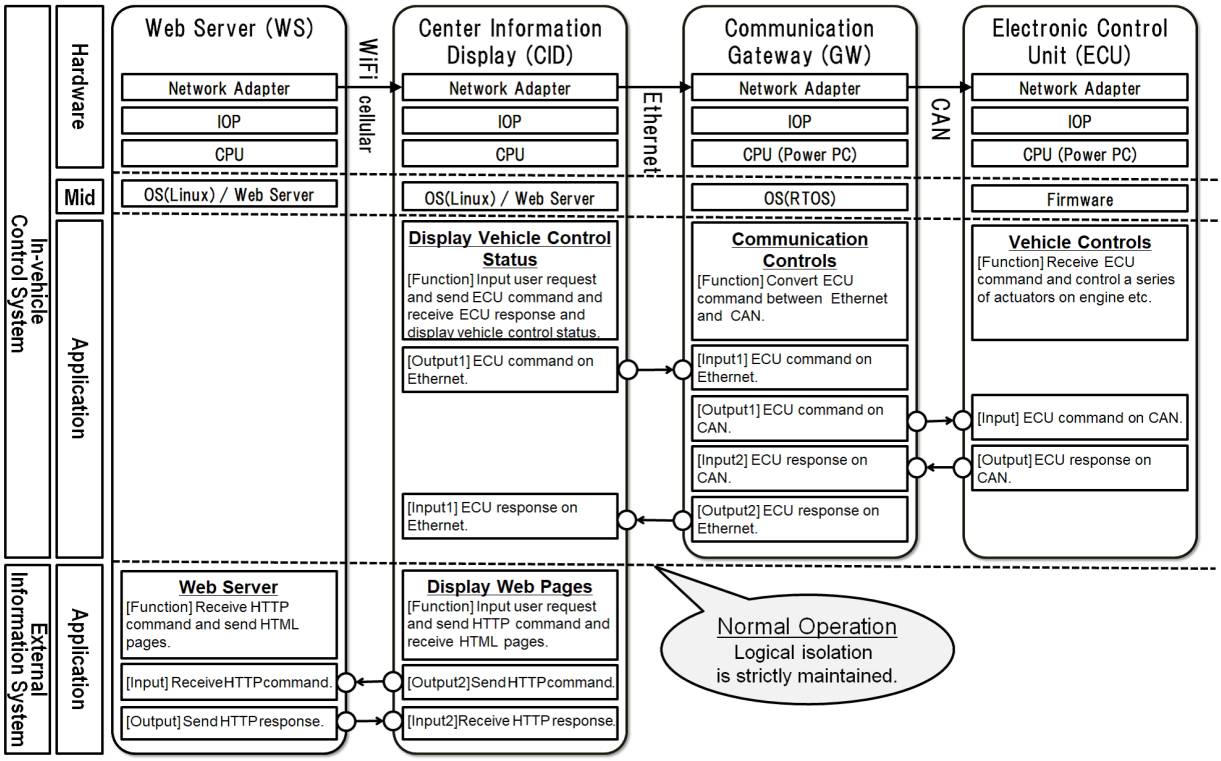


Figure 11. System model description

At this stage, as shown in Figure 12, it is possible to create a data flow diagram with attributes of these vulnerabilities added. Safety analysis is performed on this new data flow diagram. As a result, it became clear that fake commands can be transmitted from the external information system to the in-vehicle control system. With a fake command, it is possible to open and close the door of a running vehicle, and so on, demonstrating that a vehicle can be in danger.

## VIII. Conclusion

In this paper, we propose a threat analysis method using topic model analysis and vulnerability DBs. We confirmed the feasibility of matching known attack cases to vulnerability DBs using a topic model analysis tool. Moreover, we showed that our proposed method can use the results of past threat analysis for the next threat analysis. In addition, we created a system model description based on the attack tree of Tesla's case created using the proposed method. We achieve transparency in escalation of fake commands from an external information system to the in-vehicle control system. We have shown by the results of our work, that our approach to automatic threat analysis support risk analysis in discovering previous unknown relationships and so threats including their potential escalation with an connected IT system.

However, this approach does not guarantee discovery and prevention of new sophisticated attacks that are completely different from those that occurred in the past. To detect previously unknown critical vulnerabilities, it is necessary to apply this method to advanced security engineering by artificial intelligence that utilize vulnerability DBs and system design information and evaluate it in actual cases.

## Acknowledgment

Sven Wohlgemuth's contribution to this work is based on his research at Albert-Ludwig University, Freiburg, Germany, and other organizations before he joined Hitachi, Ltd. in February 2017. This work was supported by the Cabinet Office (CAO), Cross-ministerial Strategic Innovation Promotion Program (SIP), "Cybersecurity for Critical Infrastructure" (funding agency: NEDO).

## References

[1] K. Umezawa, Y. Mishina, S. Wohlgemuth, and K. Takaragi, "Threat Analysis using Vulnerability Databases – Matching Attack Cases and Vulnerability Database by Topic Model Analysis –," Proceeding of the Third International Conference on Cyber-Technologies and Cyber-Systems (CYBER 2018), pp. 74-77, Nov. 2018.

[2] A. Ruddle et al., "Deliverable D2.3: Security requirements for automotive on-board networks based on dark-side scenarios," Seventh Research Framework Programme of the European Community, July 2008, pp. 1–138.

[3] I. N. Fovino, M. Masera, and A. D. Cian, "Integrating cyber attacks within fault trees," Reliability Engineering and System Safety 94, 2009, pp. 1394–1402.

[4] MITRE Corporation, "CVE - Common Vulnerability and Exposure," https://cve.mitre.org/ [retrieved: May, 2019]

[5] MITRE Corporation, "CWE List - Common Weakness Enumeration," https://cwe.mitre.org/data/ [retrieved: May, 2019]

[6] MITRE Corporation, "CAPEC - Common Attack Pattern Enumeration and Classification," https://capec.mitre.org/ [retrieved: May, 2019]

[7] K. Umezawa, Y. Mishina, K. Taguchi, and K. Takaragi, "A Proposal of Threat Analyses using Vulnerability Databases," Proceeding of the Symposium on Cryptography and Information Security (SCIS2018), 1C2-6, January 2018, pp. 1–8.

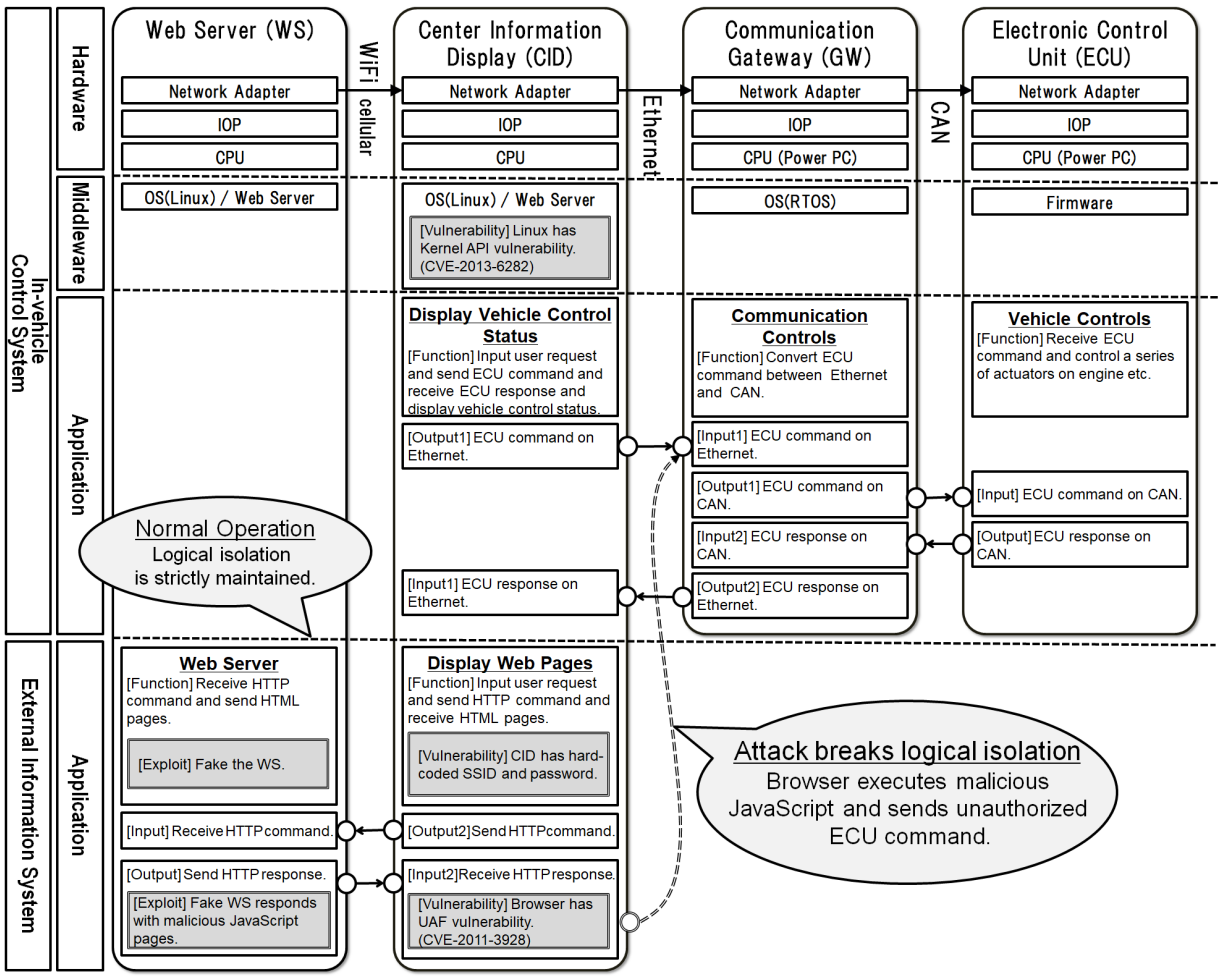


Figure 12. System model description with vulnerabilities

[8] Y. Mishina, K. Takaragi, and K. Umezawa “A Proposal of Threat Analyses for Cyber-Physical System using Vulnerability Databases,” 2018 IEEE International Symposium on Technologies for Homeland Security (IEEE HST), October 2018.

[9] S. Nie, L. Liu, and Y. Du, “Free-Fall: Hacking Tesla from Wireless to Can Bus,” Briefing, Black Hat USA 2017, July 2017. pp. 1–16.

[10] C. Miller and C. Valasek, “Remote Exploitation of an Unaltered Passenger Vehicle,” Briefing, Black Hat USA 2015, pp. 1–91.

[11] D. Quarta, M. Pogliani, M. Polino, A.M. Zanchettin, and S. Zaner, “Rogue Robots: Testing the Limits of an Industrial Robot’s Security,” Briefing, Black Hat USA 2017, July 2017.

[12] D. Blei, A. Ng, and M. Jordan, “Latent Dirichlet Allocation,” Journal of Machine Learning Research, 2003, pp. 1107–1135.

[13] T. L. Griffiths and M. Steyvers, “Finding scientific topics,” Proceedings of the National Academy of Science, 2004, pp. 5228-5235.

[14] Y. W. Teh, D. Newman, and M. Welling, “A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation,” Proceedings of Advances in Neural Information Processing Systems 19, NIPS ’07, Cambridge, MA, pp. 1353–1360, 2007.

[15] K. Handa, H. Ohsaki, and I. Takeuti, “Security Requirements Analysis Supporting Tool: TACT,” Information Processing Society of Japan (IPSJ) SIG Software Engineering (SIGSE), Proceeding of the Winter Workshop 2017. pp. 5–6.

[16] Y. Wu et al. “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” arXiv:1609.08144, 2016. pp. 1–23.

[17] G. Biggs, T. Sakamoto, and T. Kotoku, “A profile and tool for modelling safety information with design information in SysML,” Software & Systems Modeling 15, 1 (Jan 2016), pp. 147–178.

## APPENDIX

As shown in Section VI, we translate English documents into Japanese by Google translator for tool constraints before analyzing. This section examines the effect of this translation.

Figures 13 and 15 show the descriptions from Figures 9 and 10 translated back into English using Google Translate, respectively. Figures 14 and 16 show the original English descriptions.

The word underlined is the one of attention by the analysis tool. The topic model analysis assumes a model ignoring the order of words and relations between words. Therefore, it is considered that the difference in the order of the Japanese and English words does not affect the analysis result. Also, when comparing Figure 13 and 14, only a few words that are similar in meaning with double underline (such as “established” and “founded”, “occur” and “triggered”) were different. We can see that most other words are being retranslated to the same word. In other words, it can be said that there is almost no mixing of errors due to translation.

Google Chrome. The second vulnerability was **established** by KeenTeam and **may** be used for memory leaks. POC is simple.
If the function importNode () 's parameter is a node created as "xht: input", a type confusion bug will **occur** if the function copyNonAttributeProperties () **running** static_cast is executed. The size of the source type Element is 0x34, and the size of the destination type HTMLInputElement is 0x7c.
We try to allocate many Element structures together on the heap. After static_cast, the member m_data of HTMLInputElement will **duplicate** the pointer m_next of Element. We also inserted the second StringImpl structure into a pointer that is always **large** enough to read the **entire** memory.
Finally, we can chain them to execute arbitrary code.
1. **Using** the vulnerability of JSArray :: sort (), leak the JSCell address of the Uint32Array structure.
2. Use to get the address of the class structure of this Uint32Array.
3. FastFree () this address **using** the vulnerability of JSArray :: sort ().
4. Define a new Uint32Array that **realizes** arbitrary address write.
5. Insert the JavaScript function into the array.
6. Leak the JSCell address of this JavaScript function.
7. Get the JIT memory address from the JSCell address and JSC :: ExecutableBase structure.
8. Write the shell code to the JIT memory and execute this JavaScript function.
We have to say that it is difficult to develop a feasible and stable exploit without the debugging method without the QtCarBrowser binary of Tesla CID. But it was obvious that the final **abuse** gave the first shell of Tesla CID and the shell is very stable.

Figure 13. Description re-translated Figure 9 into English

The second vulnerability is CVE-2011-3928 **founded** by KeenTeam, which **could** be used for leaking memory. The POC is simple.
If the parameter of function importNode() is a node created as "xht:input", a type confusion bug will be **triggered** when the function copyNonAttributeProperties() **doing** static_cast. The size of source type Element is 0x34 and the size of destination type HTMLInputElement is 0x7c.
We try to allocate many Element structures together on heap. After static_cast, the member m_data of HTMLInputElement will **overlap** with the pointer m_next of Element. Also, we inserted the second StringImpl structure always be a pointer which is **big** enough for us to read the **whole** memory.
Finally, we can chain these together to achieve arbitrary code execution:
1. Leak a JSCell address of a Uint32Array structure by **utilizing** the vulnerability in JSArray::sort().
2. Get the address of the class structure of this Uint32Array by utilizing ~~CVE-2011-3928~~.
3. FastFree() this address by **utilizing** the vulnerability in JSArray::sort().
4. Define a new Uint32Array to **achieve** arbitrary address write.
5. Insert a JavaScript function into an array.
6. Leak the JSCell address of this JavaScript function.
7. Get the address of the JIT memory from JSCell address and JSC::ExecutableBase structure.
8. Write shellcode to JIT memory and execute this JavaScript function.
We must say it is difficult to develop a feasible and stable exploit without any debugging method and without QtCarBrowser binary from Tesla CID. However, it was deserved as the final **exploit** gave us the first shell from Tesla CID and the shell is very stable.

Figure 14. Original description before translating Figure 9

CVE-2013-6282
Since the (1) get_user API and (2) put_user API functions of Linux kernels prior to 3.5.5 on the v6k and v7 ARM platforms do not validate certain addresses, the attacker can not **change** the contents of **any** kernel memory location It is exploited wildly against crafted October 2013 Android device in November.

Figure 15. Description re-translated Figure 10 into English

CVE-2013-6282
The (1) get_user and (2) put_user API functions in the Linux kernel before 3.5.5 on the v6k and v7 ARM platforms do not validate certain addresses, which allows attackers to read or **modify** the contents of **arbitrary** kernel memory locations via a crafted application, as exploited in the wild against Android devices in October and November 2013.

Figure 16. Original description before translating Figure 10