# A Formal Model for Attack Mutation Using Dynamic Description Logics

Zhuxiao Wang[1,*], Jing Guo[2], Jin Shi[2], Hui He[1], Ying Zhang[1], Hui Peng[3], and Guanhua Tian[4]

[1] School of Control and Computer Engineering,
State Key Laboratory of Alternate Electrical Power System with Renewable Energy Sources,
North China Electric Power University, Beijing 102206, China
{wangzx,huihe,yingzhang}@ncepu.edu.cn

[2] National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing 100029, China
guojing.research@gmail.com, shijin@cert.org.cn

[3] Education Technology Center, Beijing International Studies University, Beijing 100024, China
penghui@bisu.edu.cn

[4] Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
guanhua.tian@ia.ac.cn

**Abstract.** All currently available Network-based Intrusion Detection Systems (NIDS) rely upon passive protocol analysis which is fundamentally flawed as an attack can evade detection by exploiting ambiguities in the traffic stream as seen by the NIDS. We observe that different attack variations can be derived from the original attack using simple transformations. This paper proposes a semantic model for attack mutation based on dynamic description logics (DDL(X)), extensions of description logics (DLs) with a dynamic dimension, and explores the possibility of using DDL(X) as a basis for evasion composition. The attack mutation model describes all the possible transformations and how they can be applied to the original attack to generate a large number of attack variations. Furthermore, this paper presents a heuristics planning algorithm for the automation of evasion composition at the functional level based on DDL(X). Our approach employs classical DL-TBoxes to capture the constraints of the domain, DL-ABoxes to present the attack, and DL-formulas to encode the objective sequence of packets respectively. In such a way, the evasion composition problem is solved by a decidable tableau procedure. The preliminary results certify the potential of the approach.

**Keywords:** Intrusion Detection/Prevention Systems, Multi-protocol Evasions, Advanced Evasion Techniques, Knowledge Representation and Reasoning, Dynamic Description Logics.

[*] Corresponding author.

## 1 Introduction

A weakness of most currently available Network-based Intrusion Detection Systems (NIDS) that rely upon passive protocol analysis is their inability to recognize an attack that evades detection by exploiting ambiguities in the traffic stream as seen by the NIDS. Exploitable ambiguities may arise in different ways: (1) The NIDS may lack complete analysis for the full range of behavior allowed by a particular protocol. (2) Without detailed knowledge of the victim end-system's protocol implementation, the NIDS may be unable to determine how the victim will treat a given sequence of packets if different implementations interpret the same stream of packets in different ways. (3) Without detailed knowledge of the network topology between the NIDS and the victim end-system, the NIDS may be unable to determine whether a given packet will even be seen by the end-system[1,2]. Advanced Evasion Techniques (AET's) is a lately established term in network security industry referring to a set of non-trivial and expensive means to bypass NIDS in order to deliver an exploit, attack, or other form of malware to a target network or system, without detection. Advanced evasion techniques can be identified according to certain underlying principles: (1) Delivered in a highly liberal way; (2) Employ rarely used protocol properties; (3) Use of unusual combinations; (4) Craft network traffic that disregards strict protocol specifcations; (5) Exploit the technical and inspection limitations of security devices: memory capacity, performance optimization, design flaws[3].

Since we are interested in testing the ability of a NIDS to properly identify real intrusions, we need a way to ensure that executing each of mutants generated by advanced evasion techniques against the vulnerable application, we are going to obtain the same effect as executing the original exploit script. In this article, we define an attack mutation model that describes all the possible transformations and how they can be applied to the original attack to generate a large number of attack variations. The semantic model for attack mutation is based on dynamic description logics (DDL(X)), extensions of description logics (DLs) with a dynamic dimension. The attack mutation model is self explanatory. Given an original attack, the attack mutation model can provide a proof that a sequence of transformations used for obfuscation is a real attack. Developers can use the model to analyze attacks and to identify the exact transformation that their NIDS fails to handle. The attack mutation model is exhaustive, capable of generating all attack variations from a known base attack using a set of rules. All the mutation techniques are individually sound, and also that any possible composition of them is sound. So the model is sound, generating only instances that implement the original attack.

In this article, we present an approach for automatic evasion method plans based on dynamic description logics[4-6] named DDL(X), extensions of DLs [7] with a dynamic dimension[8,9]. Our approach used classical DL-TBoxes to capture the domain constraints, DL-formulas to encode the objective sequence of packets, and DL-ABoxes to be a special representation of the attack that provides to the underlying mutation engine the mechanism to manipulate the attack, respectively. Actions in DDL(X) were used to abstract the functionalities of the available evasion methods. In such a way, the automatic evasion plans can be reduced to formula satisfiability checking in DDL(X) and solved by a decidable tableau procedure.

In the following sections, we firstly illustrate how variants of a real exploit can be derived from the original exploit script in Section 2. In Section 3, we demonstrate the descriptions of evasion techniques can be formalized as actions in DDL(X), and formalize the notion of the attack mutation model using dynamic description logics. Afterwards, in Section 4 we detail the problem of evasion composition can be solved by reasoning about actions in DDL(X) and present a heuristic planning algorithm for automated composition of evasions. Finally, we summarize the paper in Section 5.

## 2 Approach Overview

First of all, we need an instance of the exploit script that we want to mutate. The base instance is then used to derive another attack instance by repeatedly applying single step transformations. Our example vulnerability is a published buffer overflow in a commonly used Windows XP SP2 host (MSRPC Server Service Vulnerability, CVE-2008-4250 in www.cve.mitre.org.); exploiting the overflow may allow arbitrary code execution. The exploit causes the overflow by providing a crafted RPC request that triggers the overflow during path canonicalization. We call this exploit $E_{MSRPC}$.

Given an exploit E, Trace(E) denotes a sequence of packets from a NIDS perspective and the function Post(E) to be the post-conditions of the execution of E against a target system. Consider the following twelve transformation rules in Table 1 that can be applied to an existing operational description of how a vulnerability is exploited to generate a new different version of the same exploit.

We call these rules semantics preserving because they do not alter the semantics of E, i.e. the transformation does not affect the results of the execution of the exploit. If E is an instance of the $E_{MSRPC}$ attack, then by using rules in Tab.1 it is possible to derive the conclusion that the E' is also an instance of $E_{MSRPC}$ and the instance E' contains the necessary data for a successful $E_{MSRPC}$ attack. For example, we apply $r_1$ on E to send extra NetBIOS packets to break the packet flow. Then, we apply $r_9$ and change the order of TCP segments. In dynamic description logic terminology, starting with an exploit E, we can successively apply a set of transformations $T = \{r_0, r_1, \ldots, r_n\}$ to create a complex mutant exploit E'.

To formalize the notion of the semantics preserving transformation, we get:

$$\text{Trace(E)} \neq \text{Trace(E')}$$

$$\text{Post(E)} = \text{Post(E')}$$

The first condition requires that the transformation manifests itself as a change in the sequence of packets. The second condition requires that the transformation preserves the attack post-conditions. While the E and the E' might look different from a NIDS perspective, from the attacker point of view the E' generated by evasion technique is still an “effective” attack (i.e., that executing the E' against the vulnerable application, we are going to obtain the same effect as executing the original exploit script E). Intuitively speaking, one can infer E' from E, and vice versa.

**Table 1.** Atomic evasions

| | Name | Description |
|---|---|---|
| NetBIOS | NetBIOS chaff($r_1$) | Send extra NetBIOS packets to break the packet flow |
| | NetBIOS initial chaff($r_2$) | Send chaff NetBIOS packets when establishing the NetBIOS connection |
| SMB | SMB filename obfuscation($r_3$) | Obfuscate the tree name used in the SMB NT Create AndX method |
| | SMB WriteAndX padding($r_4$) | Insert extra padding between the WriteAndX header and payload |
| MSRPC | MSRPC request segmentation($r_5$) | Set the maximum number of bytes written in a single MSRPC fragment |
| | MSRPC NDR modifications($r_6$) | Set NDR types not related to endianness |
| TCP | TCP Chaff($r_7$) | Send chaff TCP segments to baffle inspection |
| | TCP timestamp option settings($r_8$) | Set initial TCP timestamp option settings |
| | TCP segment order($r_9$) | Change the order of TCP segments |
| IP | IPv4 chaff($r_{10}$) | Send chaff IPv4 packets interleaved with normal packets |
| | IPv4 fragmentation($r_{11}$) | Fragment IPv4 packets to given size |
| | IPv4 fragment order($r_{12}$) | Change the order of IPv4 fragments |

In our description frameworks for evasions, functional descriptions are essentially the state-based and use at least pre-state and post-state constraints to characterize intended executions of an evasion. Many evasion techniques can be combined. When evasion techniques are combined at different levels, NIDSs that detect each separate evasion technique often fail at detecting some permutations. First, we apply the transformation rules in a Breadth-First order: we first apply application level rules because they are independent against TCP-level and IP-level rules, then we segment each instance into small pieces, change the order of TCP segments we get, and send chaff IPv4 packets interleaved with normal packets(Figure 1). Second, we prune away some of the derivation branches to decrease the number of instances. In some cases, only a subset of possible factors evades NIDS detection; adding additional evasion measures to an evasive attack may cause the NIDS to detect it.

## 3 A Formal Model for Attack Mutation

A DDL(X) model is a tuple M = ($W$, $T$, $\Delta$, $I$), where,

$W$ is a set of states;

$T : N_A \rightarrow 2^{W \times W}$ is a function mapping action names into binary relations on W;

$\Delta$ is a non-empty domain;

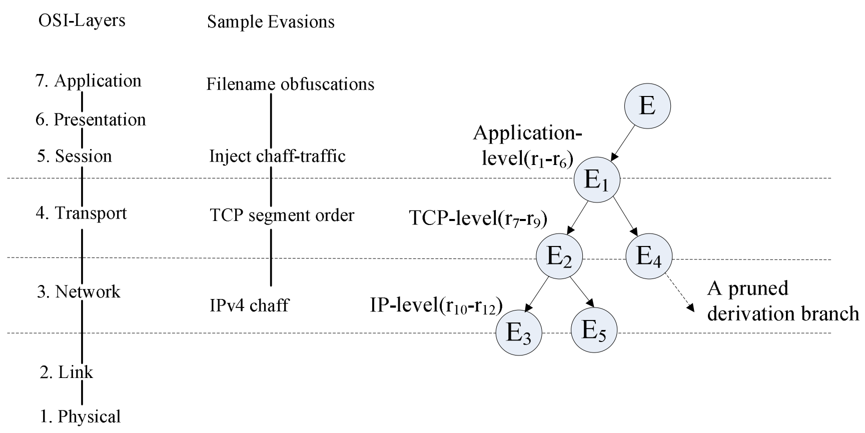

**Fig. 1.** An Illustration of a Sample Multi-protocol Evasion

$I$ is a function which associates with each state $w \in W$ a description logic interpretation $I(w) =< \Delta, \cdot^{I(w)} >$, where the mapping $\bullet^{I(w)}$ assigns each concept to a subset of $\Delta$, each role to a subset of $\Delta \times \Delta$, and each individual to an element of $\Delta$.

**Definition 1 (Atomic evasions).** An atomic evasion is a tuple t=<Pre, Effects>, where Pre is a finite set of formulas in DDL(ALCO) specifying the preconditions for the execution of t; and Effects is a finite set of assertions or their negation in ALCO, which is the facts holding in the newly-reached world by the evasion's execution.

Composite evasions are constructed from atomic evasions with the help of classic constructors in dynamic description logics[4].

Below we formally define a mutation model for an exploit as well as some reasoning tasks and the planning problem.

**Definition 2 (Mutation Model of an Exploit).** Let E be an instance of an exploit and *T* be a set of sound inference rules with respect to E.

**A mutation model of E** is a DDL(X) model $(W, T, \Delta, I)$.

Such a model enables derivation of new exploits by applying the inference rules (like evasion methods) on already known exploits. For an attack E, we envision the attack mutation model that, with respect to a set of rules, is sound: derives only sequences of packets that implement E; complete: can derive any sequence of packets that implements E; and decidable: given a sequence of packets, there is an algorithm that determines whether or not the sequence is derived from the already known exploit.

**Definition 3 (NIDS View).** Let N be a NIDS. N's view with respect to an exploit E, denoted $S^{E}_{N}$, is the set of sequences of packets that N recognizes as E.

Given a NIDS and an instance of an exploit E, the basic reasoning task for DDL(X) is to find an instance of E that evades the NIDS.

**Definition 4 (Reasoning Task 1).** Let ($W$, $T$, $\Delta$, $I$) be an attack mutation model of E, and N be a NIDS. Let $S^E_N$ be the view of N with respect to E. The reasoning problem is to find a sequence of packets S that is derivable from E, but is not in $S^E_N$. More formally, find $S \notin S^E_N$ such that $M \models S$ and $M \models E$.

Given an instance of an exploit E and a sequence of packets S, another important reasoning task for DDL(X) is to determine whether S is an instance of E.

**Definition 5 (Reasoning Task 2).** Let M = ($W$, $T$, $\Delta$, $I$) be an attack mutation model of E and S be a sequence of packets. S is an instance of E if and only if there exists a model M = ($W$, $T$, $\Delta$, $I$) and a state $w \in W$ such that $M \models E$ and $(M, w) \models S$.

The last inference problem we will investigate is the planning problem. Given a goal statement (i.e. a sequence of packets) and a set of actions(i.e. evasion methods), the planning problem is to find an action sequence that will lead from the initial state(i.e. an already known exploit) to states in which the goal will hold. With DDL(X), we define the plans as follows.

**Definition 6 (The Planning Problem).** An action sequence $r_1, ..., r_n$ is a plan for a goal S w.r.t. M = ($W$, $T$, $\Delta$, $I$) if and only if (i) the sequence-action $r_1;...; r_n$ is executable on states described by E and (ii) S is a consequence of applying $r_1;...; r_n$ on states described by E.

It is intuitive to model evasions by actions in DDL(ALCO). As demonstrated in this section, the functionalities of evasions can be semantically transformed into actions in DDL(ALCO) by a proper domain ontology (TBox). All kinds of reasoning tasks concerning the functionalities of evasions thus can be reduced to the reasoning about actions in DDL(ALCO), which are the topic of the next section.

## 4 An Efficient Algorithm for Evasion Composition

In this section we demonstrate that the problem of evasion composition can be reduced to satisfiability checking of formulas in DDL(ALCO) and then be solved by a decidable procedure after the transformation process that transforms evasions to actions in DDL(ALCO). Afterwards, we propose a heuristic planning algorithm for automated composition of evasions. The algorithm achieves good balance between computational performance and accuracy.

When facing a problem of evasion composition, we firstly collect the relevant evasions and transform these preexisting evasions to atomic actions by constructing the specification of the domain.

Let us analyze the following two formulas:

$\neg( [ (\alpha_1 \cup ... \cup \alpha_n)^* ] \Pi \wedge Con\, j(E) ) \vee < plan > true$, where Conj(E) represents the conjunction of all the elements of the set E, $\Pi$ the formula $\wedge_{i=1}^{n} (\neg Con\, j(P_i) \vee <\alpha_i> true)$ and $P_i$ the precondition of action $r_i$ for each i: $1 \le i \le n$.

The whole formula above states that the action sequence plan is executable on states described by E. The above formula is valid if its negation (labeled as Eq. 1) is not satifiable:

$$[\ (\alpha_1 \cup \ldots \cup \alpha_n)^*\ ]\ \Pi \wedge \mathrm{Con\ j}(E) \wedge \neg < \mathrm{plan} > \mathrm{true} \tag{1}$$

$$\neg \mathrm{Con\ j}(E) \vee [\mathrm{plan}]S\ .$$

This formula indicates that the goal S is a consequence of applying the action sequence plan on world states described by E. Similarly, the above formula is valid if its negation (labeled as Eq. 2) is not satifiable:

$$\mathrm{Con\ j}(E) \wedge \neg\ [\mathrm{plan}]S \tag{2}$$

Given a goal statement S and a set of actions $\sum = \{r_1, r_2, \ldots, r_m\}$, Algorithm 1 shows how to produce a plan (i.e. an action sequence) that will lead from the initial state to states in which the goal S will hold. The heuristics algorithm ActionPlan() travels the possible world states in a Breadth-First manner and terminates at a successful plan, or failure after the algorithm attempts nearly exhaustively. However, instead of travelling all possible world states, the heuristic selects world states more likely to produce an successful plan than other world states. It is selective at each decision point (Line.11), picking world states that are more likely to produce solutions.

**Algorithm 1 ActionPlan(*E*, *T*, ∑, *S*, *plan*)**

**Input:** initial ABox *E*; TBox *T*; the set $\sum = \{r_1, r_2, \ldots, r_m\}$ of available actions; and formula *S*, seen as a goal statement.

**Output**: an successful plan or *nil* as failure;

**Begin**

```
 1. initialize plan with an empty list;
 2. Initialize queue QueOfPlans with plan;
 3. while (QueOfPlans is unempty) do
 4.    remove head of QueOfPlans and set it to plan;
 5.    if (Eq. 2 is unsatifiable) then
 6.        return plan as a successful plan;
 7.    else
 8.        for each $r_i \in \sum$ do
 9.            newplan ← < plan, $r_i$ >;// appending $r_i$ to the rear of the list plan.
10.            if([ $(r_1 \cup \ldots \cup r_k)^*$ ] $\Pi \wedge Con\ j(E) \wedge \neg <$ newplan $>$ true is unsatifiable) then
11.                if(EvaluatePlan(E, T, S, plan, newplan) $\geq 0$) then
12.                    queue QueOfPlans with newplan;
13.                end if
14.            end if
15.        end for
16.    end if
17. end while
18. return nil.
```

**End**

**Algorithm 2 EvaluatePlan(*E*, *T*, *goal*, *srcplan*, *dstplan*)**

**Input:** initial ABox *E*; TBox *T*; formula *goal*, seen as a goal statement; *srcplan*, a plan for left-hand side of comparison; *dstplan*, a plan for right-hand side of comparison.

**Output:** an integer greater than, equal to, or less than 0, if the number of primitive formula achieved through dstplan is greater than, equal to, or less than the number of primitive formula achieved through srcplan, respectively.

```
Begin
    1.  srccount←0;
    2.  dstcount←0;
    3.  rewrite formula goal to a disjunctive normal form formula
        $G=\vee_{i=0}^{d} (\wedge_{j=0}^{m} \varphi_{ij})$;
    4.  for each primitive formula $\varphi_{ij}$ do
    5.      if $Con\,j(E) \wedge \neg [srcplan]\, \varphi_{ij}$ is unsatifiable then
    6.              srccount←srccount+1;
    7.      end if
    8.      if $Con\,j(E) \wedge \neg [dstplan]\, \varphi_{ij}$ is unsatifiable then
    9.              dstcount←dstcount+1;
    10.     end if
    11. end for
    12. return dstcount－srccount;
End
```

## 5 Summary

In this article, we aim at providing an effective framework for the composition of evasion techniques to test the quality of intrusion detection signatures. Our approach supports multiple evasion techniques and allows the developer of the test to compose these techniques to achieve a wide range of attack mutations. We define an attack mutation model that describes all the possible transformations and how they can be applied to the original attack to generate a large number of attack variations. The attack mutation model is based on DDL(X), extensions of description logics (DLs) with a dynamic dimension. In particular, we proposed a heuristic planning algorithm for automated composition of evasion methods by a reduction to the formula satisfiability checking in DDL(X) and a selection of world states more likely to produce a successful plan. The functionalities of the evasion methods are abstracted by actions in DDL(X), while the domain constraints, exploits, and the objective sequence of packets are encoded in TBoxes, ABoxes and DL-formulas, respectively. Then the problem of evasion composition can be reduced to formula satisfiability checking in DDL(X) and solved by a decision procedure. Afterwards, instead of travelling all possible world states in a Breadth-First manner, a heuristic planning algorithm selects world states more likely to produce an successful plan than other world states. Our approach has several important advantages: the attack mutation model permits the application of analytical methods for deriving sound evasion composition; executing each of mutants generated by our approach against the vulnerable application, we are going to obtain the same effect as executing the original exploit script.

**Acknowledgements.** This work is supported by the National Natural Science Foundation of China (No. 61300132, No.61363058, No.61372182, No.61305056, No. 61303172), the Specialized Research Fund for the Doctoral Program of Higher Education (No.20120036120003), the Fundamental Research Funds for the Central Universities (No.12ZP09, No.13ZP010) and the Scientific Plan Project of Beijing Municipal Commission of Education(No.SQKM201420131001).

## References

1. Ptacek, T.H., Newsham, T.N.: Insertion, evasion, and denial of service: Eluding network intrusion detection. Secure Networks INC Calgary Alberta (1998)
2. Handley, M., Paxson, V., Kreibich, C.: Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In: USENIX Security Symposium 2001, pp. 115–131 (2001)
3. Niemi, O.P.: Protect Against Advanced Evasion Techniques, McAfee (2014), `http://www.mcafee.com/us/resources/white-papers/wp-protect-against-adv-evasion-techniques.pdf`
4. Chang, L., Shi, Z., Gu, T., Zhao, L.: A Family of Dynamic Description Logics for Representing and Reasoning About Action. J. Autom. Reasoning, 1–52 (2010)
5. Wang, Z., Yang, K., Shi, Z.: Failure Diagnosis of Internetware Systems Using Dynamic Description Logic. J. Softw. China 21, 248–260 (2010)
6. Wang, Z., Peng, H., Guo, J., Zhang, Y., Wu, K., Xu, H., Wang, X.: An architecture description language based on dynamic description logics. In: Shi, Z., Leake, D., Vadera, S. (eds.) Intelligent Information Processing VI. IFIP AICT, vol. 385, pp. 157–166. Springer, Heidelberg (2012)
7. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F.: The description logic handbook: theory, implementation, and applications. Cambridge University Press (2003)
8. Artale, A., Franconi, E.: A temporal description logic for reasoning about actions and plans. J. Artif. Intell. Res. USA 9, 463–506 (1998)
9. Baader, F., Lutz, C., Milicic, M., Sattler, U., Wolter, F.: Integrating description logics and action formalisms: First results. Proc. Natl. Conf. Artif. Intell. USA 2, 572–577 (2005)