A Formal Verification of Configuration-Based Mutation Techniques for Moving Target Defense

Muhammad Abdul Basit Ur Rahim[1(✉)], Ehab Al-Shaer[2], and Qi Duan[3]

[1] Montana Technological University, Butte, MT 59701, USA
mabdulbasiturrahi@mtech.edu
[2] INI/CyLab, Carnegie Mellon University, Pittsburgh, PA 15213, USA
ehab@cmu.edu
[3] University of North Carolina, Charlotte, NC, USA 28213
qduan@uncc.edu

Abstract. Static system configuration provides a significant advantage for the adversaries to discover the assets and vulnerabilities in the system and launch attacks. Configuration-based moving target defense (MTD) reverses the cyber warfare asymmetry for the defenders' advantage by mutating certain configuration parameters proactively in order to disrupt attacks planning or increase the attack cost significantly.

A key challenge in developing MTD techniques is guaranteeing design correctness and operational integrity. Due to the dynamic, asynchronous, and distributed nature of moving target defense, various mutation actions can be executed in an interleaved manner causing failures in the defense mechanism itself or negative interference in the cyber operations. Therefore, it is important to verify the correctness and operational integrity, of moving target techniques to identify the design errors or inappropriate run-time behavior that might jeopardize the effectiveness of MTD or cyber operations. To the best of our knowledge, there is no work aiming for the formal verification of the design correctness and integrity of moving target defense techniques.

In this paper, we present a methodology for formal verification of configuration based moving target defense. We model the system behaviors with system modeling language (SysML) and formalize the MTD technique using du-ration calculus (DC). The formal model satisfies the constraints and de-sign correctness properties. We use the random host mutation (RHM) as a case study of the MTD system that randomly mutates the IP addresses to make end-hosts untraceable by scanners. We validate the design correctness of RHM using model checking over various configuration-based mutation parameters.

Keywords: Formal specification · Moving target defense · Configuration-based mutation · Verification

N. Park et al. (Eds.): SecureComm 2020, LNICST 335, pp. 61–79, 2020.
https://doi.org/10.1007/978-3-030-63086-7_5

1 Introduction

Moving target defense allows the cyber systems to proactively defend against a wide-scale vector of sophisticated attacks by dynamically changing the system parameters and defense strategies in a timely and economical fashion. It can provide robust defense by deceiving attackers from reaching their goals, disrupting their plans via changing adversarial behaviors and deterring them through prohibitively increasing the cost for attacks. MTD is distributed by nature since the processes or actions in MTD are executed in an interleaved manner. It is important to verify the design correctness and integrity otherwise the design error can lead to failure or inconsistencies. The verification of the MTD technique should be performed in an earlier design phase to prevent system failures.

The successful implementation of MTD techniques over numerous mutable and configurable parameters is challenging. Even with the correct design, the system must be correctly configured before performing the simulation and validation. Due to the dynamic nature of MTD, the configurable parameters are dependent on one another. The change in the value of one parameter affects the other parameters. If the parameters are not configured correctly then MTD does not remain conflict-free or system integrity will disrupt. Moreover, MTD design must satisfy the mutation constraints and design properties which make sure that the system is randomized, conflict-free, live and progressive. The reachability, liveness, deadlock-freeness, and fairness are the properties that ensure the design correctness.

In the context of software or hardware, formal verification is the act of proving or disproving the correctness of the underlying system with respect to a certain formal specification [20]. In this work, we present a formal specification and verification methodology to ensure the design correctness and operational integrity of configuration-based MTD techniques. For this purpose, we have simulated and verified the MTD technique, random host mutation (RHM), against the design correctness properties over numerous configurable and mutable parameters. We have designed a generalized model of RHM that ensures the scalability of verification of RHM. Our simulation and verification ensure the design correctness and robustness of RHM.

In this paper, the functionality of individual entities of RHM is graphically presented. The system behaviors and basic functionality are modeled using system modeling language (SysML). The graphical model is helpful to understand the MTD technique. For detailed requirement specification, we use duration calculus which is an interval logic for the specification of a real-time system. DC covers dimensions that are required for specifications and verification for embedded and real-time systems [19]. DC is useful to specify the real-time events and actions such as the one used in the time-based mutation of MTD. Moreover, the DC specifications and constraints are also provable. The RHM is a real-time system as it handles the real-time events which need a real-time response. Therefore, DC best suits for formal requirement specification of RHM.

We use model checking tool for simulation and verification of MTD techniques. Model checking tools formally specify a system and help to find the

unknown or inappropriate behavior of a system [23]. We have used UPPAAL for modeling and verification of MTD techniques. UPPAAL is a model checking tool for verification of real-time systems [23,25]. The constraints and properties are specified in temporal format and validated against the formal model to ensure the design correctness.

The rest of the paper is organized as follows. Section 2 briefly describes the preliminaries. Section 3 defines the RHM protocol. Section 4 presents the RHM verification. Section 5 describes the RHM components modeling and Sect. 6 presents the MTD verification. We present the survey of related work in Sect. 7. Section 8 concludes the methodology.

2 Preliminaries

In this section, we have discussed the preliminary concepts required to understand the methodology.

2.1 System Modeling Language

The System modeling language (SysML) is a graphical modeling language which is an extension of a unified modeling language (UML). SysML consists of three types of diagrams which are behavioral diagram, structural diagram, and requirement diagram. The state machine diagram is a behavioral diagram of SysML that we use to describe the functionality of entities associated with RHM. The SMD describes the states of a model and the transitions between the states. The rectangle and arrow represent the state and transition respectively. The transitions are labeled with a guard to enable or disable the transition. The transition executes if the guard is evaluated as true. We have modeled all components of RHM using SMD. SysML covers all modeling aspects require for modeling of MTD. The protocol, triggers, actions, constraints, and parameters can be specified using SysML.

2.2 Duration Calculus

Duration calculus is an interval logic that describes the real-time behavior of dynamic systems. It establishes a formal approach to specify and validate time duration of states in real-time system [13]. DC is also used to specify an interval temporal logic and specify the functionality of the real-time system at the abstraction, concrete and low level.

The duration calculus is designed for specifying and reasoning about embedded and real-time systems [16]. DC is widely practiced and numerous extensions of DC has been proposed for specification and verification of real-time systems [4,17,19,29,30]. The DC expressively specifies the embedded and real-time systems. There are three important dimensions (reasoning about data, communication, and real-time aspects) which are required for verification of embedded hardware and software systems. All these dimensions can be expressed by DC

[19] and these dimensions are required for model checking. Numerous prominent approaches are available that perform verification of temporal specifications by translating the problem into an automata-theoretic setting [13,19].

DC has a subset which is known as DC-implementable which we have used to describe the detailed RHM model. The DC formal model covers all aspects for modeling of MTD. The MTD protocol, constraints, and parameters are specified using a DC-based formal model. Moreover, the DC formal model is a detailed model as compare to SysML. Even the scenarios for changing the values of parameters can be specified Using DC. The most important reason for using DC is that the DC constraints are mathematically provable.

Table 1 presents some symbols for specification of duration calculus. The ceilings $\lceil\rceil$ is used to specify the state. The right arrow is used for transition. The symbol ε with a right arrow indicates the transition with delay. The right arrow with number zero indicates the control is initially in that state. The appendix describes the DC implementables along-with its use for RHM [6,13]: DC implementables describe the type of transition among the states, and sets the invariants to make the system stable.

Table 1. Some symbols of duration calculus

Symbol	Description
$\lceil x \rceil$	region,
$\longrightarrow$	followed-by,
$\longrightarrow_0$	followed-by-initially,
$\overset{\varepsilon}{\longrightarrow}$	delay transition,
$\overset{\leq\varepsilon}{\longrightarrow}$	upto,
$\overset{\varepsilon}{\longrightarrow}_0$	followed-by-initially,
;	chop

3 RHM Protocol

The RHM architecture consists of hosts, moving target gateways (MTG), domain name server (DNS), switch and moving target controller (MTC). The MTG is responsible for all interactions of MT hosts with other entities. While the MTC is brain of RHM protocol which mutates *vIP* addresses of MT hosts frequently and updates the MTGs with new *vIP*. There are two types of hosts: mutable and non-mutable hosts. The IP address of the mutable host mutates frequently by MTC to save the MT host from the scanner. The mutable host is also called as MT host. Whereas, the non-MT host is a host whose IP does not mutate. The MTG contains the translation table which contains virtual IP (*vIP*) and real IP (*rIP*) addresses of all MT hosts. It hides the addresses of MT hosts from rest

of the entities. While the domain name server (DNS) contains (*rIP*) addresses of all hosts. The DNS directly interacts with MTG and the MT host interacts with DNS through MTG.

Figure 1 illustrates the RHM protocol [22]. We have extended our earlier research and the RHM protocol [3,22]. Initially, the source MT host only knows the name of destination MT host, therefore, source host requests DNS, through source MTG, for the address of destination MT host to start the communication (step 1). The DNS receives the name of destination MT host and responds with *rIP* address of destination MT host (step 2). The source MTG intercepts DNS response and translates the *rIP* address to *vIP* address, rewrites the DNS response, and forwards to source MT host (step 3). The MT host sends a packet to source MTG to forward it to destination MT host (step 4). The source MTG translates the destination MT host *vIP* address to *rIP* address, rewrites the packet, and forwards the packet to the switch with *rIP* addresses of the sender and receiver (step 5). The switch forwards the packet to destination MTG alongwith *rIP* address of source and destination MT hosts(step 6). At the end, destination MTG translates the *rIP* address of sender to *vIP* address, forwards the packet to destination MT host using *rIP*, and shares the senders' *vIP* address (step 7). The MT host sends back the packet in the same way however MT hosts can only see the *vIP* address of each other instead of *rIP* address. In both cases, the MTGs translate the *rIP* to *vIP*.

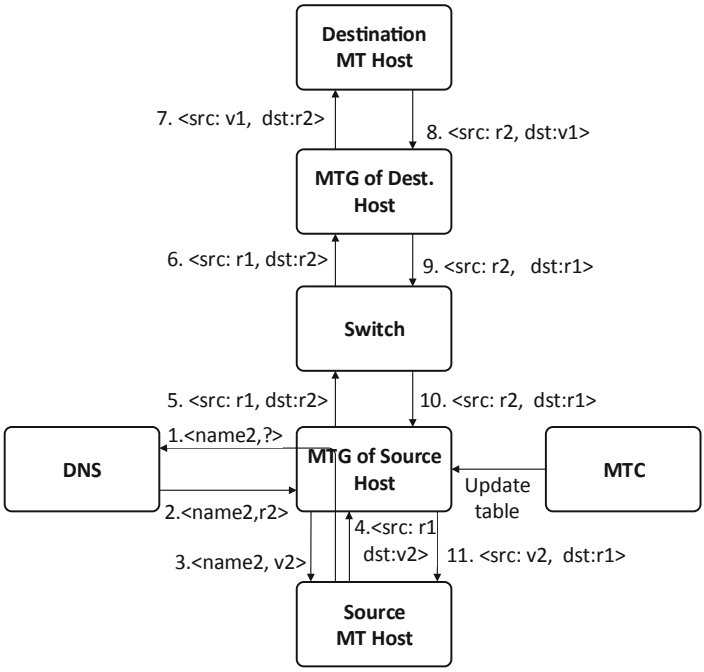

Fig. 1. RHM protocol

4 Verification Methodology

Figure 2 presents the process for specification, modeling, and validation of RHM. The SysML model is used for graphical presentation. We have modeled the behavior of RHM using SMD to present the basic functionality of MTD. The DC is used for formal specification of RHM. The DC formal model contains more details as compared to a SysML model. The DC implementables are used to describe the protocols, mutation function, and mutation and configuration parameters. The user-defined properties are specified using DC constraints. The constraints are specified as temporal properties in model checking tools.

The mapping among SysML, DC, and UPPAAL is done under the proposed mapping rules by researchers in [6,13,24]. Basit Ur Rahim et al. [6] proposed mapping rules from SysML to DC, however, the same rules are also applicable from SysML to DC. We have used the UPPAAL model checking tool for validation of design and the mutation configuration parameter. The model is analysed against the temporal properties which are specified in the form of timed computational tree logic (ECTL) properties. If the property does not hold then model checker shows the counter-example. Then the user can revise the model accordingly.

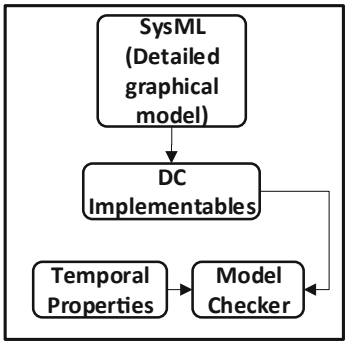

Fig. 2. Verification methodology

5 RHM Components Modeling

The MTG and MTC are graphically modeled using state machine diagram of SysML. The graphical models are simplified for a better understanding of RHM. The simplified RHM model is formally specified using DC which is a detailed formal model as compared to the graphical model. Due to page limits, we have only provided DC specification for MTG and MTC. The modeling and simulation of generalized RHM using UPPAAL model checker are quite interesting because the instance of each component interacts with the instance of other components. In this way, the generalized model has made easier the verification of different sizes of networks.

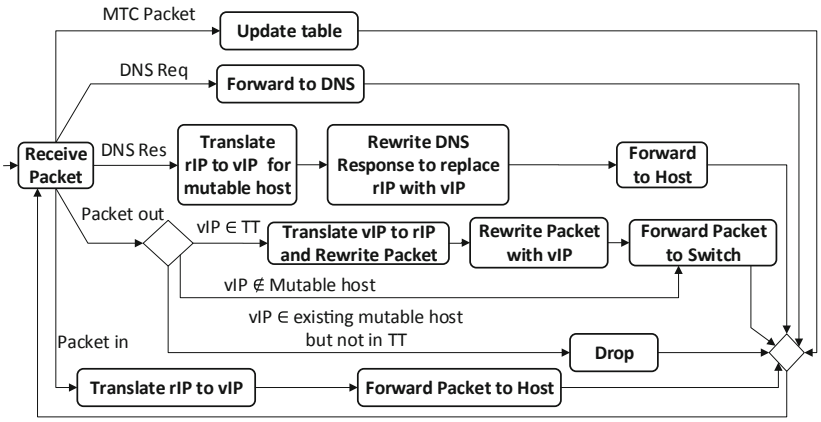

Fig. 3. State machine diagram for MTG

5.1 Moving Target Gateway

The MTG performs an important role for implementation of RHM. MTG is connected with the host, DNS, switch, and MTC. It hides the (*rIP*) addresses of MT hosts by performing address translation. Figure 3 illustrates the basic functionality of MTG for better understanding of protocol. It describes the input of MTG along-with three different address translations. In Fig. 3, the state Receive Packet is an initial state. The DNS request, DNS response, packet in, packet out, and MTC packets are inputs of MTG. The MTG updates the translation table (TT) on receiving MTC packet which is *vIP* address of MT host. MTG performs translation on receiving these inputs: DNS response, packet in, and packet out. After translation, the MTG rewrites the packet and forwards to respective entities. Table 2 presents all inputs, the performed address translation, and correspondence with respective entities.

Table 2 is the detailed DC-based formal specification of MTG. It is the detailed formal model as compared to SMD presented in Fig. 3. On receiving inputs, all step-by-step actions shown in 3 are specified using DC implementables. The implementable *Init-1* presents the initial state which is RP (receive packet). Initially, all invariants are set to false as shown by implementables from *Init-2* to *Init-8*. Most of the invariants are self-explanatory. The invariant *DReq* and *DRes* are input that represent the DNS request and DNS response respectively. The invariant *MTC_PKT* is also an input to MTG which is received from MTC to update the translation table. The invariant *MTH* is set to true when a communicating host belongs to MT host and invariant *TT* is set to true when the host is available in the translation table. By Seq-1, there are five possible transitions from initial state *Receive Packet* (RP). Syn-1 states the input for all possible transition from RP state. By Seq-2, Stab-1, and Seq-3, the MT host requests for address of destination MT host. The request is intercepted by MTG and then forwarded to DNS. Figure 3 also illustrates the MT host request with

Table 2. Implementables specifying the MTG

$Init-1:$	$\lceil \rceil \vee \lceil RP \rceil; true,$
$Init-2:$	$\lceil \rceil \vee \lceil \neg DReq \rceil; true,$
$Init-3:$	$\lceil \rceil \vee \lceil \neg DRes \rceil; true,$
$Init-4:$	$\lceil \rceil \vee \lceil \neg Pkt\_out \rceil; true,$
$Init-5:$	$\lceil \rceil \vee \lceil \neg Pkt\_in \rceil; true,$
$Init-6:$	$\lceil \rceil \vee \lceil \neg MTC\_Pkt \rceil; true,$
$Init-7:$	$\lceil \rceil \vee \lceil \neg MTH \rceil; true,$
$Init-8:$	$\lceil \rceil \vee \lceil \neg TT \rceil; true.$
$Seq-1:$	$\lceil RP \rceil \longrightarrow \lceil RP \vee Frw\_to\_DNS \vee$ Translate $\vee Drop \vee Update\_table \rceil,$
$Seq-2:$	$\lceil RP \rceil \longrightarrow \lceil Frw\_to\_DNS \rceil,$
$Seq-3:$	$\lceil Frw\_to\_DNS \rceil \longrightarrow \lceil RP \rceil,$
$Seq-4:$	$\lceil RP \rceil \longrightarrow \lceil TransDNS \rceil,$
$Seq-5:$	$\lceil TransDNS \rceil \longrightarrow \lceil Rewrite\_DNS\_Response \rceil,$
$Seq-6:$	$\lceil Rewrite\_DNS\_Response \rceil \longrightarrow \lceil Frw\_to\_host \rceil,$
$Seq-7:$	$\lceil Frw\_to\_host \rceil \longrightarrow \lceil RP \rceil,$
$Seq-8:$	$\lceil RP \rceil \longrightarrow \lceil TransOut \rceil,$
$Seq-9:$	$\lceil TransOut \rceil \longrightarrow \lceil Rewrite\_Packet \rceil,$
$Seq-10:$	$\lceil Rewrite\_Packet \rceil \longrightarrow \lceil Frw\_to\_Switch \rceil,$
$Seq-11:$	$\lceil Frw\_to\_Switch \rceil \longrightarrow \lceil RP \rceil,$
$Seq-12:$	$\lceil RP \rceil \longrightarrow \lceil Frw\_to\_Switch \rceil,$
$Seq-13:$	$\lceil Frw\_to\_Switch \rceil \longrightarrow \lceil RP \rceil,$
$Seq-14:$	$\lceil RP \rceil \longrightarrow \lceil Drop \rceil,$
$Seq-15:$	$\lceil Drop \rceil \longrightarrow \lceil RP \rceil,$
$Seq-16:$	$\lceil RP \rceil \longrightarrow \lceil TransIn \rceil,$
$Seq-17:$	$\lceil TransIn \rceil \longrightarrow \lceil Frw\_to\_host \rceil,$
$Seq-18:$	$\lceil RP \rceil \longrightarrow \lceil Update\_table \rceil,$
$Seq-19:$	$\lceil Update\_table \rceil \longrightarrow \lceil RP \rceil,$
$Syn-1:$	$\lceil RP \vee DReq \vee DRes \vee Pkt\_in \ \vee Pkt\_out \vee MTC\_Pkt \rceil \longrightarrow \lceil \neg RP \rceil,$
$Syn-2:$	$\lceil TransDNS \wedge rIP\_to\_vIP() \rceil \longrightarrow \lceil \neg TransDNS \rceil,$
$Syn-3:$	$\lceil TransIn \wedge rIP\_to\_vIP() \longrightarrow \lceil \neg TransIn \rceil,$
$Syn-4:$	$\lceil TransOut \wedge vIP\_to\_rIP() \longrightarrow \lceil \neg TransOut \rceil,$
$Stab-1:$	$\lceil \neg$RP$\rceil; \lceil$RP $\wedge$ (DReq)$\rceil \longrightarrow \lceil$Frw_to_DNS$\rceil,$
$Stab-2:$	$\lceil \neg$RP$\rceil; \lceil$RP $\wedge$ (DRes)$\rceil \longrightarrow \lceil$TransDNS$\rceil,$
$Stab-3:$	$\lceil \neg$RP$\rceil; \lceil$RP $\wedge (Pkt_out)\rceil \longrightarrow \lceil$TransOut$\rceil,$
$Stab-4:$	$\lceil \neg$RP$\rceil; \lceil$RP $\wedge$ (Pkt_out)$\&(vIP \notin MTH)\rceil \longrightarrow \lceil$Frw_to_Switch$\rceil,$
$Stab-5:$	$\lceil \neg$RP$\rceil; \lceil$RP $\wedge$ (Pkt_out&vIP $\notin TT)\rceil \longrightarrow \lceil$Drop$\rceil,$
$Stab-6:$	$\lceil \neg$RP$\rceil; \lceil$RP $\wedge$ (Pkt_in)$\rceil \longrightarrow \lceil$TransIn$\rceil,$
$Stab-7:$	$\lceil \neg$RP$\rceil; \lceil$RP $\wedge$ (MTC_Pkt)$\rceil \longrightarrow \lceil$Update_Table$\rceil$

an input DNS Req. Seq-4, Stab-2, and Syn-2 show that the DNS response is intercepted by MTG and then translate the *rIP* to *vIP*. The Seq-5 and Seq-6

lead to rewrite the DNS response and forward to host. By Seq-8, Stab-3, and Syn-4, the host requests for packet out and then MTG translates the destination MT host *vIP* to *rIP*. With Seq-9, Seq-10, and Seq-11, the MTG rewrite the packet, forwards to switch, and comes back to the initial state. Seq-12, Seq-13, and Stab-4 show that if the address does not belong to MT host then forwards the packet to switch without translation. Seq-14, Stab-5, and Seq-15 describe that if the address of destination MT host is not in the translation table (TT) then the will be drooped. Seq-16, Stab-6, Syn-3, and Seq-17 belong to the packet in. When a packet is received, the MTG translates the *rIP* address to *vIP* (Syn-3). Then MTG forwards the packet to MT host. Seq-18, Stab-7, and Seq-19 represent the updating the translation table on MTC response.

After DC specification, MTG is simulated and verified using UPPAAL. Figure 2 is the implementation of MTG using UPPAAL.

5.2 Moving Target Controller

MTC frequently mutates the *vIP* addresses of all MT hosts. Figure 4 is the SMD that illustrates the basic functionality of MTC. The *Check_time* state checks the time and starts mutation if time is above the threshold time value. The *Select_New_vIP* state performs the mutation. The MTC satisfies the constraints while selecting new *vIP*. The *m*utation function makes sure that all *N* number of hosts get new and random*vIP*, no host get the same *vIP* consecutively, and no two MT hosts of same subnet will have the same *vIP*. Once mutation process is completed at *Select_New_vIP* state, the MTC clock is reset and an MTC packet is sent to MTG to update the translation table. The mutation process also takes care of liveness and fairness properties that mutation will start after a certain interval and every host will have new *vIP*. Figure 4 represents the timed automaton of MTC where threshold time for mutation is set to 25-time units. It randomly selects an address from a given range.

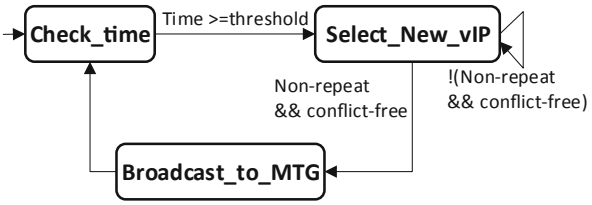

Fig. 4. State machine diagram for MTC

Table 3 presents the DC specification for MTC. The implementable *Init-1* describes the initial state of MTC which is *MTC_Ready* state. By Seq-1 and Prog-1, MTC checks the time, if the time is above the threshold time which is 25 time units then it changes the *vIP* address. Seq-1 and Stab-1 describe the mutation which iterates till all the MT hosts get new *vIP*. Stab-1 makes

sure that new *vIP* is no-repeating and conflict-free. The function vIP() in Stab-1 is responsible for mutation. It randomly selects an address and mutates if constraints are satisfied. By Seq-2 and Syn-2, when mutation process completes, the MTC broadcasts the *vIP* to MTG. In a normal system, MTC interval is an important factor and it affects the reachability of the system. We have verified the RHM over various MTC intervals which are up to 100 time units.

Table 3. Implementables specifying the MTC

$Init-1:$	$\lceil\rceil \vee \lceil MTC\_Ready\rceil; true,$
$Seq-1:$	$\lceil Check\_time\rceil \longrightarrow \lceil Select\_New\_vIP\rceil,$
$Seq-2:$	$\lceil Select\_New\_vIP\rceil \longrightarrow \lceil Broadcast\_to\_MTG\rceil,$
$Seq-3:$	$\lceil Broadcast\_to\_MTG\rceil \longrightarrow \lceil Check\_time\rceil,$
$Prog-1:$	$\lceil Check\_time\rceil \xrightarrow{25} \lceil \neg Check\_time\rceil,$
$Stab-1:$	$\lceil \neg Select\_New\_vIP\rceil; \lceil Select\_New\_vIP$ $\wedge(n < N \wedge \neg repeat \wedge \neg conflict \wedge vIP())\rceil \longrightarrow \lceil Select\_New\_vIP\rceil,$
$Syn-2:$	$\lceil Select\_New\_vIP \wedge n = N\rceil \longrightarrow \lceil \neg Select\_New\_vIP\rceil,$

The Eq. (1) is the DC constraint over MTC which specifies that initially MTC is in *MTC_Ready* state, all hosts must initialized with new *vIP* after every 25 time units. This constraint does hold by supported implemetables Prog-1, Stab-1 and Syn-2 respectively. The DC constraints are also mapped to temporal properties of model checking tool.

$$\models MTC \Rightarrow \begin{pmatrix} (\_Ready\rceil \Rightarrow \int M < \varepsilon) \\ \wedge(\lceil Check\_time\rceil \Rightarrow l \geq 25 + \varepsilon) \\ \wedge(\lceil Select\_New\_vIP\rceil \Rightarrow \int (\text{vIP} \wedge n < N) + \varepsilon) \end{pmatrix} \tag{1}$$

6 MTD Verification

The main contribution of this research is formal specification of MTD and verifying the different scenarios. We have formally specified all the entities of RHM protocol and verified it against the constraints. The DC is helpful in specifying the protocol and the constraints. Moreover, the RHM is evaluated over the configuration-based mutation parameters. We have simulated numerous scenarios for verification of RHM. The RHM has proven the robustness during simulation and verification.

6.1 Evaluation Methodology

We have simulated and verified the RHM over networks of various sizes. The DNS entry, routing entry, translation table, size of the network, size of a subnet,

and number of MTG are the important parameters that need to be correctly configured. Moreover, a mutable parameter is used to assign the random and non-repeating *vIP* address to all MT hosts. The configuration process makes sure that the network is correctly designed and configured before simulation and implementation. It makes sure that every entity gets an equal opportunity to interact with the respective components. These parameters need to be configured before the simulation and implementation start. In RHM, the MTC interval and, DNS delay are also important parameters that can affect the functionality of RHM. For the implementation of RHM in UPPAAL, the configuration process executes first and the rest of the processes execute accordingly. The configuration parameters must be correctly configured in start otherwise it will disrupt the systemś operation. We have created a configuration component in UPPAAL to configure the parameters. In this process, we have defined and configured the parameters for DNS entry, MTG entry, router entry. The mutation interval, and DNS delay, and MTG delay are initialized. The number of MT hosts and MTGs and the size of a subnet are also assigned in the configuration process.

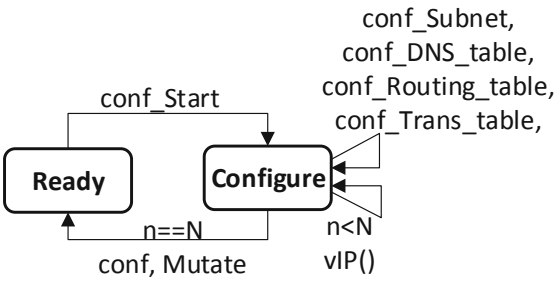

Fig. 5. State machine diagram for Configuration

Figure 5 is the SMD and presenting the configuration process. The *Ready* state is the initial state whereas the *Configure* state performs the configuration. The configuration process configures several tables that belong to MTG, DNS, and Switch which contain information about hosts and their respective MTGs. The invariant *conf_Start* represents the configuration process is started. Initially, this boolean invariant is set to false. The invariant *conf_Subnet* belongs to initialization of MT hosts to their respective subnet. The invariants *conf_DNS_table*, *conf_Routing_table*, and *conf_Trans_table* represent the initialization of DNS, routing, and translation tables respectively. Moreover, all the hosts are initialized with random *vIP* using function vIP(). A loop transition with *Configure* state executes till all N number of MT hosts are initialized with *vIP*. When the configuration process completes the invariant *Conf* and *Mutate* are set to true where the invariant *Mutate* activates the MTC for initiation of mutation process and *conf* represents the completion of configuration process. In UPPAAL, the *Configure* state is set to committed where the time does not elapse during

configuration. Moreover, all tables are initialized using user-defined functions *conf_TT()*, *cong_routing_table()*, *cong_DNS()*, *conf_subnet()*, *conf_availability()*, and *vIP()* respectively.

6.2 Properties Verification

The RHM design must satisfy the correctness properties. For this purpose, we have verified the reachability, liveness, and deadlock-freeness of RHM design. Our results prove the correctness, effectiveness, and reliability of RHM. We have found that the RHM satisfies all the correctness properties and ensures the correctness over various configuration parameters.

The RHM design is verified using ECTL temporal properties. The Reachability describes that every good state is reachable and every bad state is unreachable. The liveness describes that system is progressing to achieve a specific goal. Deadlock-freeness ensures that the system is not stopped and it is always in progressing state. The system is in deadlock state when it stops in a state and does not progress to other states [28]. Equal distribution of resources is known as fairness [28].

Table 4. Verification properties

Property	Temporal property	Result
Reachability	*A[] not (mtg.drop)*	Satisfied
	MTG should not drop the packet	
Liveness	$A[]\ mtc.Mutate\ \&\&\ n < N\ \&\&\ not(conflict\ \&\&\ repeat)$	Satisfied
	After certain interval, MTC should mutate vIP that should be conflict-free and non-repeatative	
Deadlock-freeness	*A[] not deadlock*	Satisfied
	Deadlock can prevent MTG from mutation	
Fairness	$A[]\ mtc.Broadcast\ \&\&\ n==N$	Satisfied
	New vIP for every MT host should be broadcast to MTG	

The mutation function *vIP()* of MTC randomly selects the *vIP* from an address range. The RHM satisfies the constraints that no two hosts will have the same *vIP* otherwise the packet will be delivered to the wrong destination or incorrect sender information will be shared with the receiver. It also satisfies the second constraint that no host will have the same *vIP* consecutively. The selection of random *vIP* from a long address range makes scanning difficult for an attacker. Moreover, if the mutation process is faster then even it can be more difficult for the scanner however it can affect the reachability. The system does not stop working and remains live.

The UPPAAL model checking tool verifies the time-based model against the temporal properties and results as satisfied or not-satisfied. The UPPAAL model is the detailed model as compared to MTD specification and SysML model. Table 4 presents the verification properties. The operator ([ ]) is used with temporal properties makes sure that the property is satisfied for all states on all paths.

6.3 Evaluation

While designing and implementing RHM model, we have considered different scenarios. First, the MTG keeps the older *vIP* address which is known as old window. When MTC mutates the *vIP* address, the MTG saves *viPs* of old window. If the MTG has already obtained the *vIP* address, meanwhile, MTC mutates the *vIP* address then MTG forwards the packet to older *vIP* address. Saving the older *vIP* address is also interesting that what will be the size of window. We have simulated this scenario and consider the size of window is one. If the address matches to updated or older *vIP* address then packet will be forwarded to respective address otherwise packet will be dropped. Second, we have also considered another scenario that MTG does not save older *vIP* address then MTG forwards packets to new *vIP* address only. In this case, MTG should not drop the packets. If MT host has old *vIP* address then MTG ask to obtain the new *vIP* address. In this case, if IP mutation is very frequent then it will not affect the reachability and also makes the MT host untraceable. The results presented in this section belong the second scenario.

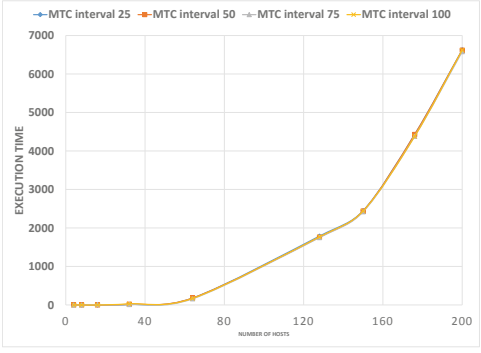

Fig. 6. Reachability analysis over MTC intervals

The analysis figures are based on numerous parameters over various sizes of subnet and networks. Figure 6 describes the reachability analysis of RHM over various mutation intervals of MTC. It shows the execution time of reachability property which is in milliseconds. Figure 6 is based on mutation intervals. For this simulation, the mutation intervals is 25 time units. The size of the subnet

is set to ten. The RHM still satisfies the reachability. The RHM protocol satisfies the reachability property that every packet is delivered to its destination host and there is no packet loss even we all these parameters. The maximum time consumed is 6700 ms for a network size of two hundred hosts. The time consumption is very low which makes verification of RHM scalable. We have also analyzed the memory usage for properties verification. We have observed that the UPPAAL model checker uses up to 1.4 GB memory for verification of a network with 200 hosts and 100 MTGs. Figure 7 presents the memory analysis of reachability property over MTC intervals.

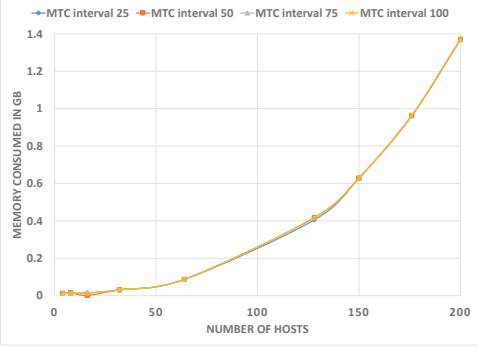

Fig. 7. Memory analysis of reachability property over MTC interval

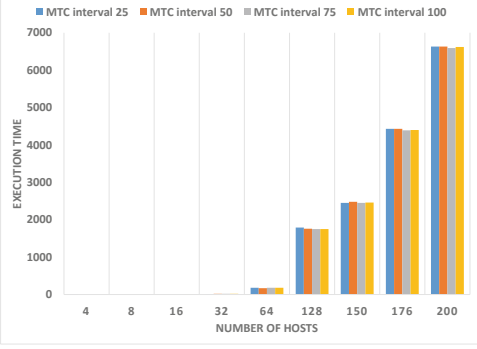

Fig. 8. Liveness analysis over MTC interval

Figure 8 presents the analysis of liveness property over MTC intervals. We have used the same parameters values as shown in Fig. 6. Although, the execution time is different, however, the trend of time consuming is almost same. The

maximum time consumed is 6800 milliseconds for network of 200 hosts and 20 MTGs. Figure 9 shows the memory analysis of liveness property over MTC intervals. The parameters for 9 are same as shown in Fig. 7. The maximum memory consumed is 1.23 GB for a network of 200 hosts with 10 MTGs. The trend of memory consumption in both figures (Fig. 7 and Fig. 9) is almost same.

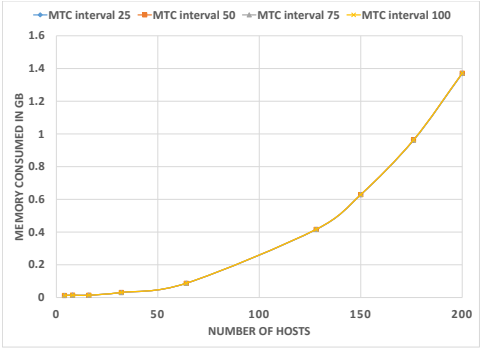

Fig. 9. Memory analysis of liveness property over MTC interval

We have also applied the methodology for formal specification and analysis of random routing mutation (RRM) which is a technique of MTD. In RRM, the router changes the routing path frequently. UPPAAL is useful for the selection of random and un-deterministic routing paths. We use UPPAAL for modeling, simulation, and verification of RRM. The design properties used for verification of RHM are used for validation of RRM.

7 Related Work

In this literature review we have evaluated few techniques for specification and verification. Pedroza et al. [21] propose a SysML profile for designing of an embedded system and perform verification for safety and security of communication links. Perroza uses UPPAAL and ProVerif toolkits for verification of embedded system. Shen et al. [27], Fang et al. [14], and Lugou et al. [18] graphically model the security protocols using unified modeling language (UML), convert it to Spi calculus to perform verification using Profier. Dadeau et al. [9] propose a set of mutation operators for HLPSL models that aim at introducing leaks in the security protocols. The model is analyzed by a tool AVISPA that produce counter-example traces leading to the leaks. Fu et al. [15] propose Security Objectives to Protocol Security Testing, to generate the test cases on-the-fly. He propose the algorithm for the protocol verification. However, the verification of randomized and dynamic system is really challenging especially when size of the network is too large. The researchers of all these proposals use the SysML

as a foundation for modeling of their case studies. Then different tools are used to perform the validation of static systems.

Duration Calculus is a specification language for embedded and real-time systems. Basit Ur Rahim et al. [6] and Olderburg [13] use the DC for specification and perform verification using UPPAAL model checker. Schwammberger [26] extended DC and named it as multi-lane spatial logic. The researcher used UPPAAL for verification of traffic control system. The UPPAAL is a model checking tool for modeling, validation and verification of real-time systems [25]. UPPAAL models the system as network of timed automata and performs analysis of the system. UPPAAL is most appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks [25]. The processes communicate through channels or shared variables [8]. UPPAAL is suitable for applications that include communication protocols and real-time controllers, in particular, those where timing aspects are critical [25]. Basit Ur Rahim et al. [6,7,23,24] has performed verification of various dynamic real-time systems using UPPAAL which helped to find design error in an earlier design phase. Sidra Sultana et al [28] verified the dynamic traffic light system using UPPAAL. As the RHM [3] is a dynamic and randomized, therefore, UPPAAL is suitable to model the framework to validate its correctness, efficiency and reliability.

The notion of mutable networks as a frequently randomized changing of network addresses and responses was initially proposed in [1]. The idea was later extended as part of the MUTE network which implemented the moving target through random address hopping and random fingerprinting [2]. Randomization is a common technique for security. Examples of information randomization include instruction set randomization [5,10], memory address randomization [11], and compiler-generated software diversity [12].

8 Conclusions

In this paper, we present a verification methodology for the correctness and integrity of MTD techniques. We have formalized, modeled, and verified the techniques of MTD over numerous configuration-based mutation parameters. As a case study, we have simulated a class of MTD for conformance of procedures and performed an analysis to ensure the design correctness. The RHM model is validated against reachability, liveness, fairness, and deadlock-freeness properties. The properties ensure that the mutation process does not affect the MTD technique. The mutation process is also evaluated over non-repeating and conflict-free constraints. The mutation process ensures that every MT host address mutates over a certain interval. The results present that dynamic and randomized RHM is correct, effective, and reliable. The verification of RHM does not utilize enough resources that make verification more scalable. The verification of RHM has consumed up to 1.4 GB memory and 6700 ms for verification of TCTL properties for a network with (up to) 200 MT hosts, 100 MTGs, and 40 MT hosts over a subnet. We have designed the generalized model of RHM that also helps to design, simulate, and verify a scalable network.

Acknowledgement. This research was supported in part by the United States Army Research Office (ARO). Any opinions, findings, conclusions or recommendations stated in this material are those of the authors and do not necessarily reflect the views of the funding sources.

Appendix: DC Implementable with Definition

DC implementables	Pattern	Description
Initialisation	$\lceil\rceil \longrightarrow \lceil x \rceil; true,$	Each control automaton is either empty or in initial state
Sequence	$\lceil x \rceil \longrightarrow \lceil x \vee x_1 \vee x_2 \vee .. \vee x_n \rceil$	if the control automaton is in state x it subsequently stays in x or moves to one of the state $x_1, ..., x_n$. e.g. all transition among the states of a component are specified using sequence implementable
Progress	$\lceil x \rceil \xrightarrow{\varepsilon} \lceil \neg x \rceil$	The control automaton stays for ε seconds in state x, it leaves this state and progresses accordingly. e.g. This will be used to define the computational overhead of mutation
Synchronization	$\lceil x \wedge \alpha \rceil \xrightarrow{\varepsilon} \lceil \neg x \rceil$	The control automaton stays for ε second in state x with condition α being true. The time unit ε is a clock that can be either discrete or continuous type. e.g. This will be used to allow the transition iff the mutation criterion is satisfied
Bounded Stability	$\lceil \neg x \rceil; \lceil x \wedge \alpha \rceil \xrightarrow{\leq \varepsilon} \lceil x \vee x_1 \vee x_2 \vee .. \vee x_n \rceil$	When the control automaton changes its state to x with the condition α being true and the time does not exceed ε seconds, it stays in x or it moves to one of states $x_1, ..., x_n$. e.g. Mutate *vIP* after a certain time interval
Unbounded Stability	$\lceil \neg x \rceil; \lceil x \wedge \alpha \rceil \longrightarrow \lceil x \vee x_1 \vee x_2 \vee .. \vee x_n \rceil$	when the control automaton changes its state to x with the condition α being true, it stays in x or it moves to one of states $x_1, ..., x_n$. e.g. Select new *vIP* till all MT hosts get a new *vIP*
Bounded Initial Stability	$\lceil x \wedge \alpha \rceil \xrightarrow{\leq \varepsilon}_0 \lceil x \vee x_1 \vee x_2 \vee .. \vee x_n \rceil$	When the control automaton initially is in state x with the condition α being true and the time does not exceed ε seconds, it stays in x or it moves to one of states $x_1, .., x_n$. e.g. Set values of all parameters in initial phase before mutation starts
Unbounded Initial Stability	$\lceil x \wedge \alpha \rceil \longrightarrow_0 \lceil x \vee x_1 \vee x_2 \vee .. \vee x_n \rceil$	When the control automaton initially is in phase x with the condition α being true, it stays in x or it moves to one of states $x_1, .., x_n$. e.g. Initialise the configuration parameters in initial phase

References

1. Al-Shaer, E.: Mutable networks, National cyber leap year summit 2009 participants ideas report. Technical report, Networking and Information Technology Research and Development (NTIRD) (2009)
2. Al-Shaer, E.: Toward network configuration randomization for moving target defense. In: Jajodia, S., Ghosh, A.K., Swarup, V., Wang, C., Wang, X.S. (eds.) Moving Target Defense, Advances in Information Security, vol. 54, pp. 153–159. Springer, New York (2011). https://doi.org/10.1007/978-1-4614-0977-9_9
3. Al-Shaer, E., Duan, Q., Jafarian, J.H.: Random host mutation for moving target defense. In: Keromytis, A.D., Di Pietro, R. (eds.) SecureComm 2012. LNICST, vol. 106, pp. 310–327. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36883-7_19
4. An, J., Zhan, N., Li, X., Zhang, M., Yi, W.: Model checking bounded continuous-time extended linear duration invariants. In: Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week), HSCC 2018 pp. 81–90. ACM, New York (2018)
5. Atighetchi, M., Pal, P., Webber, F., Jones, C.: Adaptive use of network-centric mechanisms in cyber-defense. In: Second IEEE International Symposium on Network Computing and Applications, NCA 2003, pp. 179–188 (2003)
6. Basit-Ur-Rahim, M.A., Ahmad, J., Arif, F.: Parallel verification of UML using divine tool. In: 2013 5th International Conference on Computer Science and Information Technology, pp. 49–53 (2013)
7. Basit-Ur-Rahim, M.A., Arif, F., Ahmad, J.: Modeling of real-time embedded systems using SysML and its verification using UPPAAL and DiVinE. In: 2014 IEEE 5th International Conference on Software Engineering and Service Science, pp. 132–136 (2014)
8. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30080-9_7
9. Dadeau, F., Héam, P., Kheddam, R.: Mutation-based test generation from security protocols in hlpsl. In: 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation, pp. 240–248 (2011)
10. Droms, R., Bound, J., Volz, B., Lemon, T., Perkins, C., Carney, M.: Dynamic host configuration protocol for IPv6 (2003). https://tools.ietf.org/html/rfc3315
11. Dunlop, M., Groat, S., Marchany, R., Tront, J.: IPv6: now you see me, now you don't. In: The Tenth International Conference on Networks, ICN 2011 (2011)
12. Dunlop, M., Groat, S., Urbanski, W., Marchany, R., Tront, J.: Mt6d: a moving target IPv6 defense. In: 2011 - MILCOM 2011 Military Communications Conference, pp. 1321–1326 (2011)
13. Olderog, E.R., Dierks, H.: Real-Time Systems: Formal Specification and Automatic Verification. Cambridge University Press, Cambridge (2008)
14. Fang, K., Li, X., Hao, J., Feng, Z.: Formal modeling and verification of security protocols on cloud computing systems based on UML 2.3. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 852–859 (2016)
15. Fu, Y., Koné, O.: Validation of security protocol implementations from security objectives. Comput. Secur. **36**, 27–39 (2013)
16. Goranko, V., Montanari, A., Sciavicco, G.: A road map of interval temporal logics and duration calculi. J. Appl. Non-Class. Logics **14**(1–2), 9–54 (2004)

17. Guelev, D.P., Wang, S., Zhan, N.: Compositional hoare-style reasoning about hybrid CSP in the duration calculus. In: Larsen, K.G., Sokolsky, O., Wang, J. (eds.) SETTA 2017. LNCS, vol. 10606, pp. 110–127. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69483-2_7
18. Lugou, F., Li, L.W., Apvrille, L., Ameur-Boulifa, R.: SYSML models and model transformation for security. In: 2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 331–338 (2016)
19. Meyer, R., Faber, J., Rybalchenko, A.: Model checking duration calculus: a practical approach. In: Barkaoui, K., Cavalcanti, A., Cerone, A. (eds.) ICTAC 2006. LNCS, vol. 4281, pp. 332–346. Springer, Heidelberg (2006). https://doi.org/10.1007/11921240_23
20. Page, R.L.: Engineering software correctness. J. Funct. Program. **17**(6), 675–686 (2007)
21. Pedroza, G., Apvrille, L., Knorreck, D.: Avatar: A SYSML environment for the formal verification of safety and security properties. In: 2011 11th Annual International Conference on New Technologies of Distributed Systems, pp. 1–10 (2011)
22. Rahim, M.A.B.U., Duan, Q., Al-Shaer, E.: A formal analysis of moving target defense (2020)
23. Basit ur Rahim, M.A., Arif, F.: Translating activity diagram from duration calculus for modeling of real-time systems and its formal verification using UPPAAL and DiVinE. Mehran Univ. Res. J. Eng. Technol. **35**(1), 139–154 (2016)
24. Basit Ur Rahim, M.A., Arif, F., Ahmad, J.: Modeling of embedded system using SysML and its parallel verification using DiVinE tool. In: Murgante, B., et al. (eds.) ICCSA 2014. LNCS, vol. 8583, pp. 541–555. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09156-3_38
25. Ravn, A.P., Srba, J., Vighio, S.: Modelling and verification of web services business activity protocol. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 357–371. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_32
26. Schwammberger, M.: Introducing liveness into multi-lane spatial logic lane change controllers using UPPAAL, pp. 17–31. CoRR abs/1804.04346 (2018)
27. Shen, G., Li, X., Feng, R., Xu, G., Hu, J., Feng, Z.: An extended UML method for the verification of security protocols. In: 2014 19th International Conference on Engineering of Complex Computer Systems, pp. 19–28 (2014)
28. Sultana, S., Arif, F.: Computational conversion via translation rules for transforming C++ code into UPPAAL's automata. IEEE Access **5**, 14455–14467 (2017)
29. Wang, H., Zhou, X., Dong, Y., Tang, L.: Modeling timing behavior for cyber-physical systems. In: 2009 International Conference on Computational Intelligence and Software Engineering, pp. 1–4 (2009)
30. Zhang, M., Liu, Z., Zhan, N.: Model checking linear duration invariants of networks of automata. In: Arbab, F., Sirjani, M. (eds.) FSEN 2009. LNCS, vol. 5961, pp. 244–259. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11623-0_14