RESEARCH ARTICLE
doi: 10.2306/scienceasia1513-1874.2016.42S.049

ScienceAsia 42S (2016): **49–55**

Model checking the DNS under DNS cache-poisoning attacks using SPIN

Wei Zhang[a,b,*], **Meihong Yang**[a], **Xinchang Zhang**[a], **Huiling Shi**[a]

[a] Shandong Key Laboratory of Computer Networks, Shandong Computer Science Centre (National Supercomputer Centre in Jinan), Jinan 250000, China

[b] College of Information Science and Engineering, Shandong University of Science and Technology, Qingdao 266510, China

*Corresponding author, e-mail: wzhang@sdas.org

Received 31 Aug 2014
Accepted 20 Jul 2016

ABSTRACT: Domain name system (DNS) security has garnered substantial interest due to DNS cache-poisoning attacks. In this article, a model checking method is employed to verify the security of the DNS protocol, especially when it is under DNS cache-poisoning attacks. The DNS protocol is first translated into a simpler model that preserves all the attack behaviour to be verified. Extended finite state machine models are given and represented in PROMELA which can be identified by SPIN; and then the whole process of model checking is proposed. The initial results on verification of the DNS under DNS cache-poisoning attacks using SPIN are also proposed. From the experimental results it can be seen that the security of DNS should be carefully considered.

KEYWORDS: protocol verification, extended finite state machine, PROMELA

INTRODUCTION

Domain Name System (DNS) is a hierarchical distributed naming system that translates alphanumeric domain names into IP addresses. DNS makes it possible to use a Uniform Resource Locator (URL) to address a host in the internet. DNS forms the logical backbone of the World Wide Web, and the service it provides is used on the order of a trillion times a day[1]. That the administration of the system can be distributed among several name servers is an important feature of the design of DNS. Zone is a contiguous part of the domain name space which is managed by many authoritative name servers. Then the distribution is achieved by delegating part of the zone administration to several subzones. DNS was designed to be a public database with no intentions to restrict access to information. Nowadays, many distributed applications make use of DNS. Confidence on the working of those applications depends critically on the use of trusted data: fake information inside the system has been shown to lead to unexpected and potentially dangerous problems.

DNS security has garnered substantial interest, due to the highly publicized DNS cache-poisoning vulnerability discovered by Kaminsky[2]. The attack targets DNSs URL-resolution mechanism so that an infected DNS server gives an incorrect IP address for a URL. An intruder can exploit this mechanism to hijack an internet domain. Specifically, a corrupted DNS server will reply with the IP address of a malicious web server when asked to resolve URLs within a non-malicious domain such as *www.qq.com*. This would direct many unsuspecting clients (ordinary desktop machines) to the malicious web site when they wanted to visit a web site within the domain *www.qq.com*.

In this article, we use the model checker SPIN. In general, this approach consists of constructing a formal model of the protocol design, and then using an automatic analysis technique to prove or to check the satisfaction of a given set of critical properties[3]. Probably, the most promising formal methods to ensure a prior reliability is model checking[4,5], Analysing the DNS protocol with model checking comprises the following steps: (a) construction of a protocol model; (b) specification of the reliability properties with a property-oriented language; and (c) production of a reachability graph including all the execution paths for the model in order to check whether these paths satisfy the properties. This technique has been integrated in many academic and industrial oriented tools.

This article also discusses how to employ SPIN[6–8], one of the most powerful and well-known model checkers, in order to specify the correctness of DNS protocol, and analyse the DNS under cache-poisoning attacks. The rest of the article is orga-

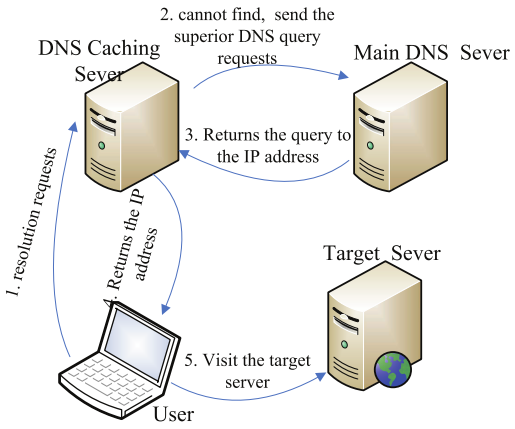

Fig. 1 Domain name resolution in DNS.

nized as follows. The Background Section provides background material on DNS, DNS cache-poisoning attacks and the model checking technology. Then we presents the modelling process of DNS protocol and DNS under attack. The process of the model checking is also given. The results of some experiments of the verification are discussed. we give some conclusions of the works we have done, and point out some further works we would do in the last section.

BACKGROUND

DNS protocol

Domain Name System is a hierarchical naming system for the internet based on underlying client-server architecture, which is also hierarchical in nature. The primary function of a DNS server is to perform URL-resolution: the process of translating a URL or domain name, such as *www.qq.com* into a physical IP address, such as 119.188.89.220. DNS servers are organized hierarchically in terms of top-level domains and subordinate, lower-level domains, respectively, *com*, *qq* and *www* in the example.

The DNS protocol is shown in Fig. 1 and the details of steps are as follows:

(1) In client, a URL (e.g., *www.qq.com*) is filled in into the address bar of the browser, the client must know the IP address corresponding to the URL. The client then sends a parse request for temporary DNS server; the request is something like: "could you tell me which IP address *www.qq.com* is?"
(2) If the IP address of *www.qq.com* cannot be found in the temporary DNS server, the DNS server will send the parse request to the main DNS server. The main DNS server on the preservation of *www.qq.com* corresponds to the IP address.
(3) The main DNS server replies to temporary DNS server request, the response is something like: "the IP address of *www.qq.com* is 119.188.89.220".
(4) After the temporary DNS server received the response from the main DNS server, it sends the response message to the client.
(5) The client uses the correct IP address to access the *qq* homepage.

DNS cache-poisoning attacks

The attack targets URL-resolution mechanism of DNS so that an infected DNS server gives an incorrect IP address for a URL. An intruder can exploit this mechanism to hijack an internet domain. Specifically, a corrupted DNS server will reply with the IP address of a malicious web server when asked to resolve URLs within a non-malicious domain such as www.qq.com. To understand how DNS cache-poisoning attack works, consider the following steps. A user wants to visit the target domain, and asks to resolve the URL within the DNS. Meanwhile, an intruder substitutes a wrong IP address of the DNS server. So the user may be receiving the wrong IP address, and visit the server which has been set up by the intruder. In the event of a successful attack, the intruder will reply to the users URL-resolution request for a URL within the target domain with the IP address of the malicious domain.

The DNS cache-poisoning attack is shown in Fig. 2, in which the steps marked in red are the steps of the attack process. The details of steps are as follows:

(1) The user sends a parse request to temporary DNS server for the IP address of *www.qq.com*.
(2) If the IP address of *www.qq.com* cannot be found in the temporary DNS server, the DNS server will send the parse request to the main DNS server. The main DNS server on the preservation of *www.qq.com* corresponds to the IP address.
(3) When the main DNS server has not returned to the temporary DNS server, the attacker sends a wrong IP address to the temporary DNS server. The temporary DNS server does not know that it has been given a DNS cache poison.
(4) The temporary DNS server replies to the client with the false IP address of the domain name.
(5) The user receives the false IP address, and visits

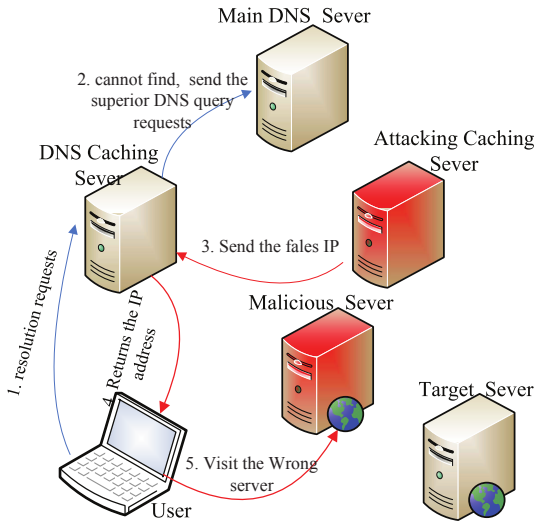

Fig. 2 DNS cache-poisoning attacks.

the malicious server which has been set up by the attacker.

Model checking and SPIN

Model checking is a technique that relies on building a finite model of a system and checking that a desired property holds in the model or not. As specified in Ref. 9, model checking has been used primarily in hardware and protocol verification. There are two methods for model checking. The first one is modelled as finite transition system and specifications are expressed in a temporal logic. Another is that, the specification is given as an automaton and the system is also modelled as automaton, and is compared to the specification to determine whether its behaviour conforms to the specification [10].

There are many kinds of model checking tools, such as SPIN [7], this NUSMV2, and etc. Among them, the model checker SPIN is one of the most popular, providing a friendly user interface and acceptance model specification written in PROMELA (PROcess MEta LAnguage) [11]. PROMELA is used for building verification models which represents an abstract of a protocol, which contains the aspects relevant to the properties one wants to verify [12]. A PROMELA program consists of processes, message channels, and variables. Processes are defined globally; while message channels and variables can be declared either globally or locally within a process. Processes are used to specify system types of behaviour, and channels and global variables are used to define the environment in which the processes run. Examples and further details about the PROMELA language can be found in Ref. 7.

MODELLING THE DNS UNDER DNS CACHE-POISONING ATTACKS

Formal modelling is the first and crucial step in model checking. Constructing a model for a protocol in PROMELA requires a previous abstraction process of the original source code. Usually, this process eliminates details that are not necessary for debugging purposes. Models will therefore be as small as possible, making sure that they represent the exact details needed for the properties to be analysed.

Extended finite state machine model

Extended finite state machine (EFSM) model is extended from the finite state machine (FSM) model. Compared with FSM, there are environmental variables and the migration of pre-conditions in EFSM. So EFSM model has a stronger ability to describe the dynamic behaviour of the system. For these reasons, we use EFSM to model the process, which is a formula in the area of model checking, and can be described in PROMELA easily.

Definition 1 EFSM M is defined as the tuple $(S, s_0, V, M_V, P, M_P, I, O, T)$ in which S is a finite set of states; s_0 is the initial state, $s_0 \in S$; V is the finite set of the internal variables (environment variables), and the range of the internal variable is D_V; M_V is the set of the initial (or default) values of variables in V, in which any element can be expressed as a tuple (s, v), $s \in S$, $v \in D_V$; P is the set of the input and output parameters; M_P is the set of the initial (or default) values of variables in P, in which any element can be expressed as a tuple (p, u), $p \in I \cup O$, $u \in D_p$, D_p is the range of the input and output parameters; I is the set of the input symbols; O is the set of the output symbols; T is the finite set of state transition.

A state transition t ($t \in T$) is defined as the tuple $(s, x, y, g_P, g_E, op, e)$, where: s and t are the start (head) state and the end (tail) state; g_P is the input and output conditions to determine; g_E is the conditions to determine of the variable required for migration; x and y are the input and output symbols; op is the output operations.

Modelling the DNS protocol by EFSM

It requires a previous abstraction process of the original source code that constructing a model for a protocol in PROMELA. Usually, details that are not

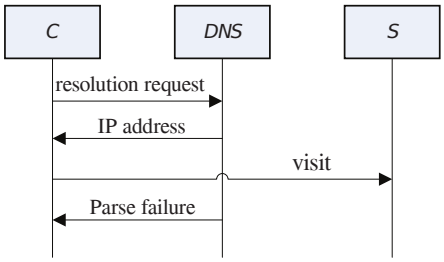

Fig. 3 Message flow in the DNS protocol.

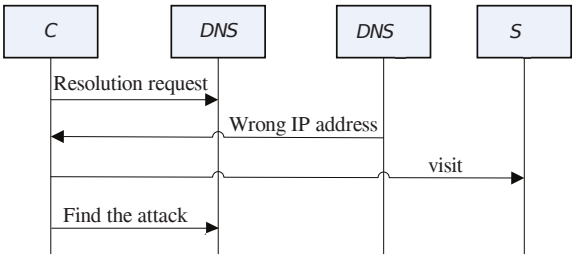

Fig. 5 Message flow under DNS cache-poisoning attacks.

necessary for debugging purposes will be eliminated in this process. In addition, the models should be as small as possible to make sure that they represent the exact details which are needed to be analysed. As the analysis, we focus on the most important attributes, ignore the less important attribute. Hence we abstract the DNS protocol. In the formal modelling process, we should ignore the participants which are independent of the desired characteristics of the protocol. To obtain an accurate model of the DNS protocol, here we only pay attention to the three objects, the client (C), the DNS, and the target server (S). We use the notation "A ⇒ B: *message*" to indicate that A sends the *message* to B. The basic DNS protocol consists of the following messages:

(1) C ⇒ DNS: *Domain name resolution requests*
(2) DNS ⇒ C: *IP Address*
(3) C ⇒ S: *Visit*
(4) DNS ⇒ C: *Parse failure*

The message flow is shown in Fig. 3.

Fig. 4 shows the EFSM model of the DNS protocol, containing three-state and five-state transitions. The abstracted model is small but can describe all the properties which we care about. The label R-OK/IP means that, when the input symbol of the state RESOLUTION satisfying R-OK, the state will be converted to the state INIT, and it will output the symbol IP.

Modelling the DNS cache-poisoning attacks

The whole process of attack is given in the last section. Combined with DNS protocol model abstracted in the last section, we will make further abstraction and description of the message flow in the process of attack. Here we only consider the three objects: the client (C), the DNS, the bogus DNS (DNS′) and the bogus server (S′). The detailed message flow is shown in Fig. 5.

According to the EFSM model of DNS protocol discussed in last section, the EFSM model of the DNS under DNS cache-poisoning attacks can be shown in Fig. 6. There are five-state and nine-state transitions in the EFSM model. The label IP =*wip/attack ok* means that, when the input symbol of the state INIT satisfying IP =*wip*, the state will be converted to the state W-VISIT, and it will output the symbol *attack ok*.

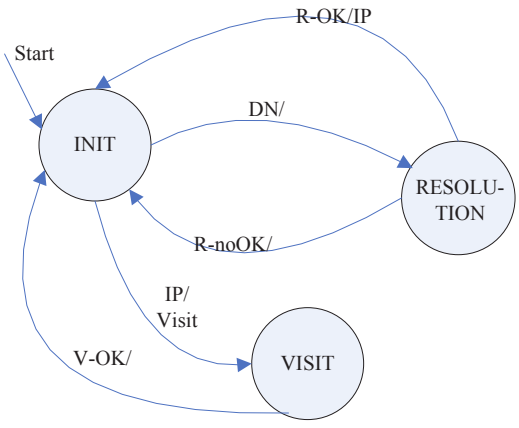

Fig. 4 EFSM of the DNS protocol.

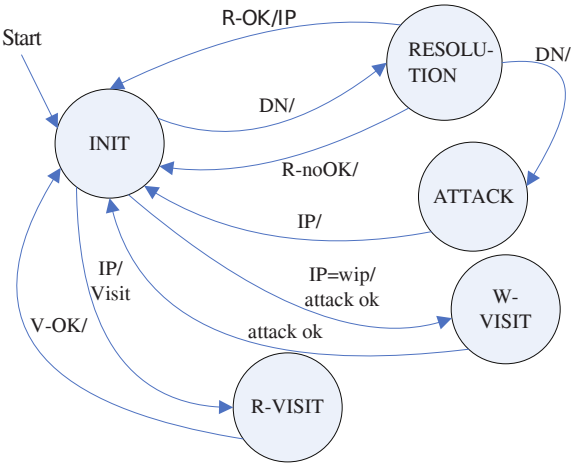

Fig. 6 EFSM of DNS under cache-poisoning attacks.

THE PROCESS OF MODEL CHECKING

According to the analysis of message flow above, we have abstracted two models. The EFSM models can be identified by SPIN while they have been translated by the PROMELA. The fragment of PROMELA code that represents EFSM is presented in Listing 1 as the DNS Process. In this fragment, each label like `DNS_init` or `DNS_wait` represents every state of the process. Notation `:` denotes the possible nondeterministic choices of further execution options inside bodies of `if` and `do` operations. The option can be selected if its guard (the first statement follows right after `:`) is enabled — that is executable. If more than one option is enabled, one will be selected at random. Notation `goto` means the process turns from one state to another state.

Channels are used to model data flow between processes and can be either globally scoped or locally scoped within a single process. The fragment presented in Listing 2 shows how to define the channels.

SPIN supports two kinds of analysis for the modelled protocols. The first one consists of checking deadlocks and other safety properties by generating the execution paths in the model. The second kind of analysis consists of checking temporal properties specified with temporal logic. Here we describe the correctness criteria we are interested in, and show how they can be defined in SPIN. To formalize both desired and undesired properties of the DNS, we use linear temporal logic notation (LTL) to describe them. The SPIN model checker supports specification of system properties using LTL, which has been proven to have good expressibility and more natural language like statements for verification[13,14]. LTL consists of only a few logic operators, such as [] (always), <> (eventually), U (until), W (unless, or weak until), and O (next). Combining with Boolean operators, i.e., && (and), || (or), ! (Negation), → (logical implication), and ↔ (logical equivalence), LTL is capable of describing many key properties of a concurrent software system.

Listing 1 Fragment of DNS Process in PROMELA.

```
active proctype DNS()
{records=100;
 DNS_init:
 if
  ::C_DNS?RS ->
    if
      ::IP=wip -> DNS_C!R-OK;
                  goto DNS_wait
      ::IP!=wip -> DNS_C!R-noOK;
                   goto DNS_init
    fi;
 fi;
 DNS_wait:
 ...
}
```

Listing 2 Fragment of the definition of channels in PROMELA.

```
{chan DNS_C=[0] of {mtype};
 chan DNS_C=[0] of {int};
 chan C_DNS=[0] of {mtype};
 chan S_DNS=[0] of {mtype};
 chan C_S=[0] of {mtype};
 chan S_C=[0] of {mtype};
...
}
```

EXPERIMENTAL RESULTS

SPIN will search the whole state space of the model, if the model has been given in PROMELA. SPIN can also identify the unreachable state or deadlock in model. In addition, SPIN can construct a verifier, which can check several claims on the execution of the model. We have established the model of DNS under cache-poisoning attacks. In this section, we present various kinds of verifications that can be performed on a PROMELA model described in the sections above. Using SPIN and the PROMELA specification presented above, several properties of the execution of the models were verified. For each experiment, the size of the model constructed by SPIN, and the time for verification were measured. The experiments were carried out on a 2.7 GHz Pentium dual-core machine with 8 GB of memory.

First, we let SPIN perform a full state space search for invalid end states, which is SPIN formalization of deadlock states, in case of 10 attackers. The results are shown in Fig. 7. In line 2, the + sign indicates that the default simple algorithms have been used; otherwise, the − sign indicates that it is fully compiled without simplification. Line 3 represents the type of search. The − sign in line 4 means that it did not use a never claim or an LTL formula. Line 5 indicates that the detection process does not violate the user-defined statements (assert). Line 6 indicates that the process does not detect the current user-defined acceptable cycle or no progress marked circle. The + sign in line 7 represents the correct end

```
(Spin Version 6.1.0 -- 4 May 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (not selected)
        invalid end states      +

State-vector 100 byte, depth reached 3162, errors: 0
 94540894 states, stored
 337456453 states, matched
 431997347 transitions (= stored+matched)
      0 atomic steps
hash conflicts: 126864754 (resolved)

 3616.172   memory usage (Mbyte)
.....
.....
pan: elapsed time 251.8 seconds
pan: rate 375460.26states/second
```

Fig. 7 Results of verification using SPIN.

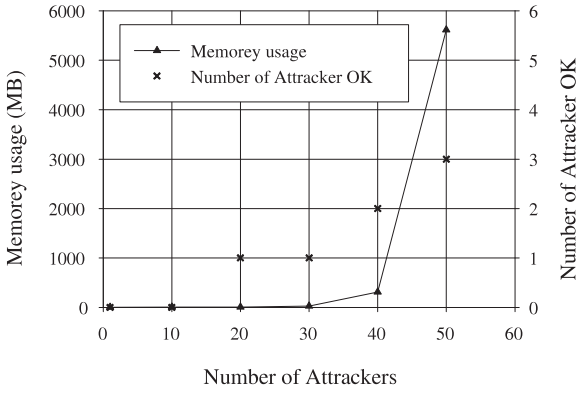

Fig. 8 Results of the experiments.

state, which means that there is no deadlock.

We have employed SPIN to perform a full state space search in cases of 1, 10, 20, 30, 40, and 50 attackers. The results are summarized in Fig. 8. The exponential increase in the number of memory usage seems to be not manageable when checking more than 50 attackers. In Fig. 8, we can also see the number of the output *attack ok* is in the growth with the increasing of the number of attackers. In other words, this means the DNS cache-poisoning attacks are successful.

CONCLUSIONS

In this paper, we introduced a model checking approach to verify the DNS under DNS cache-poisoning attacks. First we analysed the DNS protocol, the participants of the DNS under DNS cache-poisoning attacks, and the message flow in them. We proposed the EFSM models, and translated them in PROMELA. Then we analysed the whole process of model checking using SPIN. We did some experiments, which can prove that our model can simulate the actual attacks. The initial results on verification of the DNS under DNS cache-poisoning attacks using SPIN are also proposed in this paper. Through the experimental results, it can be seen that the security of DNS should be paid more attention to. For our future work, we will try to combine our approach with more details of the DNS protocol. Also we will perform model checking on the DNSEC protocol to compare whether it is more secure than the DNS protocol.

Acknowledgements: This work was supported by Shandong Provincial Natural Science Foundation of China (Grant no. ZR2015YL019) and National Natural Science Foundation of China (Grant no. 61272433 and no. 61472230).

REFERENCES

1. Alexiou N, Basagiannis S, Katsaros P, Dashpande T, Smolka SA (2010) Formal analysis of the Kaminsky DNS cache-poisoning attack using probabilistic model checking. In: *Proceedings of the 2010 IEEE 12th International Symposium on High-Assurance Systems Engineering (HASE)*, pp 94–103.
2. Kaminsky D (2008) It's the end of the cache as we know it, Black Hat USA 2008, [talk].
3. Gallardo MM, Martínez J, Merino P (2005) Model checking active networks with SPIN. *Comput Comm* **28**, 609–22.
4. Clarke EM, Emerson EA, Sistla AP (1986) Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans Program Lang Syst* **8**, 244–63.
5. Clarke EM, Grumberg O, Peled D (1999) *Model Checking*, MIT Press.
6. Holzmann GJ (1991) *Design and Validation of Computer Protocols*, Prentice-Hall.
7. Holzmann GJ (1997) The model checker SPIN. *IEEE Trans Software Eng* **23**, 279–95.
8. Eker S, Meseguer J, Sridharanarayanan A (2004) The Maude LTL model checker. *Electron Notes Theor Comput Sci* **71**, 162–87.
9. Clarke EM, Wing JM (1996) Formal methods: State of the art and future directions. *ACM Comput Surv* **28**, 626–43.
10. Shaikh R, Devane S (2010) Formal verification of payment protocol using AVISPA. *Int J Infonomics* **3**, 326–37.

11. Holzmann GJ (2004) *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley, pp 17–9.
12. Xu H, Cheng YT (2007) Model checking bidding behaviors in internet concurrent auctions. *Int J Comput Syst Sci Eng* **22**, 179–91.
13. Manna Z, Pnueli A (1992) *The Temporal Logic of Reactive and Concurrent Systems: Specifications*, Springer Science & Business Media.
14. Clarke EM, Grumberg O, Hamaguchi K (1997) Another look at LTL model checking. *Formal Meth Syst Des* **10**, 47–71.