Wikiprint Book

Title: Model for DNS Cache Poisoning (Kaminsky)

Subject: ThirdEye - ExampleDNSKaminsky

Version: 21

Date: 09/15/14 11:39:25

Table of Contents

Model for DNS Cache Poisoning (Kaminsky)

Running experiments on a networking testbed, such as DETER, is challenging since it is hard to ascertain the validity of the experiment manually. With SAF, a model can be used to capture the "definition of validity" which includes possible successful and failed behaviors for an experiment and then confirmatory analysis can verify if it was met. Such a model can also be easily shared with other experimenters promoting sharing and reuse of experiments.

Our experiment here is Dan Kaminsky?s popular DNS attack which was simulated on the DETER testbed using the metasploit framework. Referring to setup shown in Figure 1, the attackers objective is to poison the cache of the *victimns* so that any requests to *bofa.com* are redirected to a fake nameserver (*fakens*) instead of the real nameserver (*realns*).

Figure 1. Experiment Setup

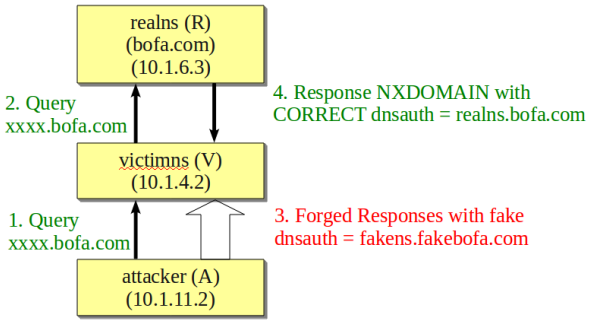
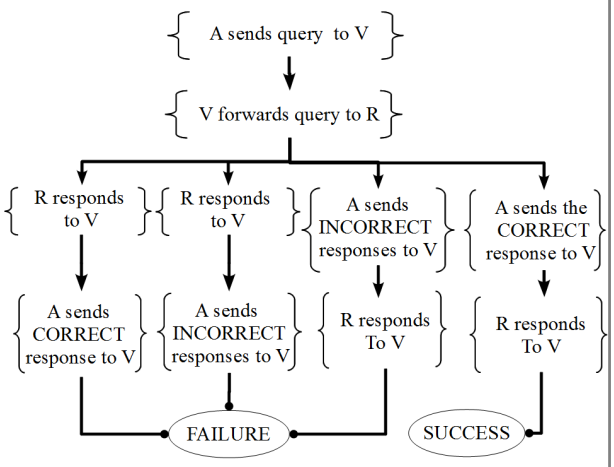


Figure 2. Conceptual model of experiment behavior



Also shown in Figure 1 any single attack attempt comprises of 4 basic steps.

1. Attacker sends a DNS query for a randomly generated name under the *bofa.com* domain to the victim nameserver. The attacker uses a randomly generated name to ensure that the victim nameserver always forwards the query upstream.
2. The victim nameserver forwards it to the real nameserver for the domain.
3. The attacker sends a series of forged DNS responses (containing the fake nameserver as authority) to the victim and spoofing as the real nameserver.
4. The real nameserver responds with NXDOMAIN.

Note that steps 3 and 4 can occur in any order.

There are two conditions to be satisfied for the attack to work:

1. The attackers forged responses have to match the responses from the real nameserver in atleast the following attributes:
    1. source IP : attacker knows this.
    2. destination IP : attacker knows this.
    3. destination port : This needs to be same as the source port used in the forwarded victimns to realns query.
    4. source port: 53
    5. dnsid : A random 16-bit number which the attacker has to guess.
1. In addition to above, the attacker has to beat the actual response from the real nameserver response.

The attacker keeps repeating the basic steps for as many number of times until the attack is successful. Figure 2 captures the high-level conceptual model of possible experiment behaviors in terms of the above steps. The curly braces correspond to a single step in an attack. Paths connecting the steps a capture an entire attack behavior. The single steps are treated as states and the paths are complex behaviors. In all there are three possible behaviors that can lead to failures and one behavior that can lead to success.

Analysis using SAF

Model Script

The model script encodes the high-level experiment behavior captured using the modeling language. Download the model.

```
##################################################################################
# Script Name
#   DNSKAMINSKY
#
# Description
#   Model of the DNS Kaminsky cache poisoning attack.
#   A detailed explanation of the attack and the corresponding model is
#   can be found at http://thirdeye.deterlab.net/trac/wiki/ExampleDNSKaminsky.
#
# Input Requirements
#       Event Type: PACKET_DNS
#       Event Attributes: sipaddr,dipaddr,sport,dport, protocol,
#                         dnsqrflag, dnsauth, dnsid, dnsquesname
#
# Output
#   Events satisfying either the SUCCESS or FAILURE behaviors.
#
# Example Dataset(s)
#   http://thirdeye.deterlab.net/trac/browser/trunk/saf-data/db/nsdipaper_casestudy2_data.sqlite
#
# SAF compatibility
#   0.2a and later
#
# Depends On
#   NET.APP_PROTO.DNSREQRES
#
# References
#   http://thirdeye.deterlab.net/trac/wiki/ExampleDNSKaminsky
#       Contains a more detailed explanation of the model, input and output.
#
# Model Author(s)
#   Arun Viswanthan (aviswana@usc.edu)
#
# Additional Notes
#   This model uses constant values for the 'dnsauth' variable.  Make sure
#   to replace those constant values by appropriate values while using with
#   other datasets.  A future version of the model will remove this
#   limitation.
#
# # Leave these lines unchanged.
# $URL: http://thirdeye.deterlab.net/svn/trunk/SAF/knowbase/net/attacks/dnskaminsky.b $
# $LastChangedDate: 2011-07-05 11:52:34 -0700 (Tue, 05 Jul 2011) $
##################################################################################

[header]
NAMESPACE = NET.ATTACKS
NAME = DNSKAMINSKY
QUALIFIER = {eventtype='PACKET_DNS'}

# Import the DNSREQRES model that already defines states and behaviors relevant
# to the DNS protocol.
IMPORT = NET.APP_PROTO.DNSREQRES

[states]
#---------------------------------------------------------------------------
# There are 5 possible states in the experiment that need to be captured.
# A, V and R are used to refer to Attacker, Victim nameserver and Real nameserver
```

```
# respectively
#-------------------------------------------------------------------------

#-------------------------------------------------------------------------
# State: Attacker sends a DNS query for a randomly generated name under the
#        'domain' in question.
#
# This is done by directly importing the predefined state dns_req()
#-------------------------------------------------------------------------
AtoV_query = DNSREQRES.dns_req()

#-------------------------------------------------------------------------
# State: The victim nameserver forwards it to the real nameserver for the domain.
#
# The forwarded query is captured by importing dns_req() with the
# following additional dependencies
# (a) the sipaddr of this request must be the same as the dipaddr of the previous
#      query,
# (b) the dnsquesname must be same as the dnsquesname of the request.
#-------------------------------------------------------------------------
VtoR_query = DNSREQRES.dns_req(sipaddr=$AtoV_query.dipaddr,
                               dnsquesname=$AtoV_query.dnsquesname)

#-------------------------------------------------------------------------
# State: The attacker sends a series of CORRECTLY forged DNS responses
#        (containing the fake nameserver as authority) to the victim and
#         spoofing  as the real nameserver.
#
# To capture this, import the predefined DNS response with the additional
# constraints:
# (a) We make all values of this query dependent upon the values in the
#     previous query since the attacker is spoofing the response.
# (b) We indicate that the only change in the response is the fake nameserver
#     by setting a constant value for the dnsauth variable.

# We use bcount to group all attacker events matching AtoV_resp into a single
# instance.
#-------------------------------------------------------------------------
AtoV_resp = DNSREQRES.dns_res($VtoR_query, dnsauth='fakens.fakebofa.com')[bcount>=1]


#-------------------------------------------------------------------------
# State: The attacker sends a series of INCORRECTLY forged DNS responses (containing
#        the fake nameserver as authority) to the victim and spoofing  as the real
#        nameserver.
#
# This is captured by adding the constraint that dnsid guessed by the attacker
# does not match the correct dnsid in the forwarded query from VtoR.
#-------------------------------------------------------------------------
AtoV_noresp = DNSREQRES.dns_res($VtoR_query, dnsid != $VtoR_query.dnsid)[bcount>1]

#-------------------------------------------------------------------------
# State: The real nameserver response.
#
# This is captured by importing the predefined DNS response with the additional
# constraints:
# (a) Make all values of this query dependent upon the values in the
#     VtoR_query since this is a response to that request and all values would
#     be appropriately dependent upon that state.
# (b) Define a constant value for the dnsauth variable
#     indicating the correct value of dnsauth.
```

```
#-------------------------------------------------------------------------
RtoV_resp = DNSREQRES.dns_res($VtoR_query, dnsauth='realns.bofa.com')

[behavior]
#-------------------------------------------------------------------------
# There are four behaviors (3 failures and 1 success) we consider in the
# model. Each behavior is a combination of the above states which correspond to
# the 4 basic steps of the attack.
#
# For all possible behaviors, the first two states are same.
#-------------------------------------------------------------------------

#-------------------------------------------------------------------------
# Failure 1: Attacker responds with CORRECT responses but after the real
#            nameserver responds
#-------------------------------------------------------------------------
b_1 = (AtoV_query ~> VtoR_query ~> RtoV_resp ~> AtoV_resp)

#-------------------------------------------------------------------------
# Failure 2: Attacker responds with INCORRECT responses during the correct
#            responses coming from the attacker.
#
# The 'dur' (during) parallel operator captures the parallel responses of
# attacker and victim.
#-------------------------------------------------------------------------
b_2 = ((AtoV_query ~> VtoR_query) ~> (RtoV_resp dur AtoV_noresp))

#-------------------------------------------------------------------------
# Failure 3: Attacker responds with INCORRECT responses before the real
#            nameserver responds
#-------------------------------------------------------------------------
b_3 = (AtoV_query ~> VtoR_query ~> AtoV_noresp ~> RtoV_resp)

#-------------------------------------------------------------------------
# SUCCESS: Attacker responds with CORRECT response before the real
#          nameserver responds
#-------------------------------------------------------------------------
b_4 = (AtoV_query ~> VtoR_query ~> AtoV_resp ~> RtoV_resp)

#-------------------------------------------------------------------------
# Group the behaviors explicity into FAILURE and SUCCESS.
#
# Note that this grouping is useful to increase the readability of the model
# and the output.
#-------------------------------------------------------------------------
FAILURE = (b_1 or b_2 or b_3)
SUCCESS = (b_4)
M = (FAILURE or SUCCESS)

[model]
DNSKAMINSKY(eventno,timestamp,timestampusec,sipaddr,dipaddr,sport,dport,dnsquesname, dnsid,dnsauth, dnsqrflag) = M
```

Using the model with SAF

The experiment is run and packets are captured at *victimns*. The experiment stops once the cache is successfully poisoned.

Normalization of captured traces

The captured traces are normalized using the p2db tool included with the framework. Normalization results in generation of PACKET_DNS events.

Sample event database

Download the following sample event databases.

Dataset	Description
nsdipaper_casestudy2_data.sqlite	Traces were collected at the *victimns* and contain a successful attack instance.

Command line

```
./saf.py --db nsdipaper_casestudy2_data.sqlite  --model knowbase/net/attacks/dnskaminsky.b  --pretty
```

Output

Only relevant output containing events satisfying the FAILURE behavior and SUCCESS behavior are shown below. Detailed output can be downloaded from here.

```
Reading input event database '../saf-data/db/nsdipaper_casestudy2_data.sqlite' ..
Found 9138 events in database
       PACKET_DNS - 9138 events [ Wed Jun  2 21:51:25 2010 (1275515485) to Wed Jun  2 21:51:28 2010 (1
Creating temporary directory for storing state /tmp/temp
Initializing global symbol table..
Reading and initializing from the knowledge base 'knowbase'..
Parsing specified model : 'knowbase/net/attacks/dnskaminsky.b'..
Processing model DNSKAMINSKY
   QUALIFIER matched 9138 instances
   QUALIFIER matched 9138 instances
   QUALIFIER matched 9138 instances
   State AtoV_query .. found 1308 instances
   State VtoR_query .. found 651 instances
   State RtoV_resp .. found 623 instances
   State AtoV_resp .. found 1 instances
 Behavior b_1 .. found 0 instances
   QUALIFIER matched 9138 instances
   State AtoV_query .. found 1308 instances
   State VtoR_query .. found 651 instances
   State RtoV_resp .. found 623 instances
   State AtoV_noresp .. found 622 instances
 Behavior b_2 .. found 619 instances
   QUALIFIER matched 9138 instances
   State AtoV_query .. found 1308 instances
   State VtoR_query .. found 651 instances
   State AtoV_noresp .. found 622 instances
   State RtoV_resp .. found 621 instances
 Behavior b_3 .. found 0 instances
 Behavior FAILURE .. found 619 instances
   QUALIFIER matched 9138 instances
   QUALIFIER matched 9138 instances
   State AtoV_query .. found 1308 instances
   State VtoR_query .. found 651 instances
```

```
   State AtoV_resp .. found 1 instances
   State RtoV_resp .. found 1 instances
 Behavior b_4 .. found 1 instances
 Behavior SUCCESS .. found 1 instances
 Behavior M .. found 620 instances
Model DNSKAMINSKY satisfied by 620 instances



================================
Instances satisfying DNSKAMINSKY
================================

Total Matching Instances: 620
```

eventno	timestamp	timestampusec	sipaddr	dipaddr	sp
					Behavior: FAIL
3	1275515485	1169	10.1.11.2	10.1.4.2	19
4	1275515485	1253	10.1.4.2	10.1.6.3	32
					Behavior: FAIL
5	1275515485	1668	10.1.6.3	10.1.4.2	
6	1275515485	2168	10.1.6.3	10.1.4.2	
7	1275515485	2418	10.1.6.3	10.1.4.2	
8	1275515485	2668	10.1.6.3	10.1.4.2	
11	1275515485	2919	10.1.6.3	10.1.4.2	
12	1275515485	3167	10.1.6.3	10.1.4.2	
13	1275515485	3667	10.1.6.3	10.1.4.2	
14	1275515485	3917	10.1.6.3	10.1.4.2	
15	1275515485	4167	10.1.6.3	10.1.4.2	
16	1275515485	4417	10.1.6.3	10.1.4.2	
					Behavior: FAIL
9	1275515485	2672	10.1.6.3	10.1.4.2	

```
//Output Truncated//
```

					Behavior: SUCC
8715	1275515488	573824	10.1.11.2	10.1.4.2	38
8716	1275515488	573917	10.1.4.2	10.1.6.3	32
					Behavior: SUCC
8717	1275515488	574324	10.1.6.3	10.1.4.2	
8719	1275515488	574824	10.1.6.3	10.1.4.2	
8720	1275515488	575075	10.1.6.3	10.1.4.2	
8722	1275515488	575329	10.1.6.3	10.1.4.2	
8723	1275515488	575574	10.1.6.3	10.1.4.2	
8724	1275515488	576074	10.1.6.3	10.1.4.2	
8725	1275515488	576323	10.1.6.3	10.1.4.2	
8726	1275515488	576573	10.1.6.3	10.1.4.2	
8727	1275515488	576823	10.1.6.3	10.1.4.2	

```
      8728        |     1275515488     |       577323       |      10.1.6.3      |      10.1.4.2      |
      8721        |     1275515488     |       575324       |      10.1.6.3      |      10.1.4.2      |
-------------------------------------------------------------------------------------------------------
```

References

1. DETER
2. An illustrated guide to Dan Kaminsky?s popular DNS attack