DNSSEC Protocol Parser:
A case study

Fatema Bannat Wala
Security Engineer
University of Delaware
Fatema.bannatwala@gmail.com

About Me

- A big fan of Zeek IDS
- Have been working with Zeek for past 3.5 years
- Joined UD's IT Security Operations (IT-SecOps Team) in 2015
- Passionate about Cyber-Security
- Also a part-time Ph.D. student

Agenda

- Motivation - How it all started??
- Why DNSSEC Protocol parser?
- Getting started …
- Deep dive into code – The top-down approach!
- Summary

UD's Traffic Profile

- 10 Gbps network links
- Average ~5Gbps network traffic
- Peak ~8Gbps
- 4 Zeek sensors, each getting 25% of total traffic
- Each sensor getting ~300,000pps
- Overall triggered Weird notices logged in weird.log were ~21M/day

Motivation – How it All Started?

- Weird logs analysis – Almost ~200 unique weird notices in weird.log
- Investigating the condition triggering the weird notices logging
- But where to find them?
  - In core layer - source code of BRO IDS (in .cc files)
  - In script land - in base/ policy/ folders (in various .bro scripts)

Motivation – How it All Started?

- The noisiest one was “DNS_RR_unknown_type”

```
$ zcat weird.*.log.gz | awk -F'\t' '{print $7}' | sort | uniq -c | sort -rn
    2,603,914 DNS_RR_unknown_type
    2,160,812 possible_split_routing
    2,092,811 inappropriate_FIN
```

- Ok, so what DNS RR types are unknown to Zeek?

```
$ zcat weird.*.log.gz | grep “DNS_RR_unknown”| awk -F'\t' '{print $7,$8}' | sort | uniq -c | sort -rn
    1,049,729 DNS_RR_unknown_type 46
      234,393 DNS_RR_unknown_type 50
       99,914 DNS_RR_unknown_type 43
       44,346 DNS_RR_unknown_type 47
       26,592 DNS_RR_unknown_type 48
```

Analysis - Why DNSSEC Protocol parser?

After looking up those RR numbers, looks like they belong to DNSSEC protocol types:

RR Type ID	RR Type Name	Description
46	RRSIG	Resource Record Signature
50	NSEC3	Next Secure Record v3
43	DS	Delegation Signer
47	NSEC	Next Secure Record
48	DNSKEY	Public Key record of the key pair used for Signing RRSIG Records

Getting started –Step by Step!

Find the condition that triggers the weird notice – ‘DNS_RR_unknown_type’

- Script Land (/usr/local/bro/2.5.1/share) –

```
[user@sensor ~]$ grep   -rnw   '/usr/local/bro/default/share‘  -e   'DNS_RR_unknown_type‘
/usr/local/bro/default/share/bro/base/frameworks/notice/weird.bro:122:
["DNS_RR_unknown_type"]   = ACTION_LOG,
```

…. controls logging of different weirds..

Triggering condition not there.. hmm

```
.....
## A table specifying default/recommended
actions per weird type.

const actions: table[string] of Action = {
 ......
 ["DNS_RR_bad_length"] = ACTION_LOG,
 ["DNS_RR_length_mismatch"] = ACTION_LOG,
 ["DNS_RR_unknown_type"] = ACTION_LOG,
 ......
} &default=ACTION_LOG &redef;
 ......
 .....
```

Getting started –Step by Step!

Find the condition that triggers that weird notice – ‘DNS_RR_unknown_type’

- Core Layer (Source Code)–

```
[user@sensor bro-2.5.1]$ grep   -rnw   ‘src/‘  -e   'DNS_RR_unknown_type‘
src/analyzer/protocol/dns/DNS.cc:326:   analyzer->Weird("DNS_RR_unknown_type",
fmt("%d", msg->atype));
```

Looks like DNS.cc defines the condition that triggers Weird notices for “DNS_RR_unknown_type” weird notice.

Getting started – So Far

What we know so far:

- 'DNS_RR_unknown_type' is the noisiest weird we are trying to investigate.
- It isn't defined in Script land
- It is defined in the source code, specifically in file DNS.cc

Some Tips for deep dive into the code..

- A good text/code editor - Sublime is a great text editor tool to effectively navigate between the code files.
  - Use the tool you are comfortable with!
- Need the source code of Zeek, so if not build from source, get the source code from the website/Github.

Deep Dive into the Code

Let's take a look at DNS.cc - the condition that triggers 'DNS_RR_unknown_type'

```
306
307         case TYPE_EDNS:
308             status = ParseRR_EDNS(msg, data, len, rdlength, msg_start);
309             break;
310
311         case TYPE_TSIG:
312             status = ParseRR_TSIG(msg, data, len, rdlength, msg_start);
313             break;
314
315         default:
316
317             if ( dns_unknown_reply && ! msg->skip_event )
318                 {
319                 val_list* vl = new val_list;
320                 vl->append(analyzer->BuildConnVal());
321                 vl->append(msg->BuildHdrVal());
322                 vl->append(msg->BuildAnswerVal());
323                 analyzer->ConnectionEvent(dns_unknown_reply, vl);
324                 }
325
326             analyzer->Weird("DNS_RR_unknown_type", fmt("%d", msg->atype));
327             data += rdlength;
328             len -= rdlength;
329             status = 1;
330             break;
331     }
```

Deep Dive into the Code

Let's take a look at DNS.h

```
35  typedef enum {
36      TYPE_A = 1,      ///< host address
37      TYPE_NS = 2,         ///< authoritative name server
38      TYPE_CNAME = 5,      ///< canonical name
39      TYPE_SOA = 6,        ///< start of authority
40      TYPE_WKS = 11,       ///< well known service
41      TYPE_PTR = 12,       ///< domain name pointer
42      TYPE_HINFO = 13,     ///< host information
43      TYPE_MX = 15,        ///< mail routing information
44      TYPE_TXT = 16,       ///< text strings
45      TYPE_SIG = 24,       ///< digital signature (RFC 2535)
46      TYPE_KEY = 25,       ///< public key (RFC 2535)
47      TYPE_PX = 26,        ///< pointer to X.400/RFC822 mapping info (RFC 1664)
48      TYPE_AAAA = 28,      ///< IPv6 address (RFC 1886
49      TYPE_NBS = 32,       ///< Netbios name (RFC 1002)
50      TYPE_SRV = 33,       ///< service location (RFC 2052)
51      TYPE_NAPTR = 35,     ///< naming authority pointer (RFC 2168)
52      TYPE_KX = 36,        ///< Key Exchange (RFC 2230)
53      TYPE_CERT = 37,      ///< Certificate (RFC 2538)
54      TYPE_A6 = 38,        ///< IPv6 address with indirection (RFC 2874)
55      TYPE_DNAME = 39,     ///< Non-terminal DNS name redirection (RFC 2672)
56      TYPE_EDNS = 41,      ///< OPT pseudo-RR (RFC 2671)
57      TYPE_TKEY = 249,     ///< Transaction Key (RFC 2930)
58      TYPE_TSIG = 250,     ///< Transaction Signature (RFC 2845)
59      TYPE_CAA = 257,      ///< Certification Authority Authorization (RFC 6844)
60
61      // The following are only valid in queries.
62      TYPE_AXFR = 252,
63      TYPE_ALL = 255,
64      TYPE_WINS = 65281,  ///< Microsoft's WINS RR
65      TYPE_WINSR = 65282, ///< Microsoft's WINS-R RR
66  } RR_Type;
```

Top Down Approach

So there are MANY RR Types parsers already written in DNS.cc

- Analysis:
  - Can extend the code to include the parsing of unknown RR types corresponding to DNSSEC
    - Find if there is a parsed RR type closest to any of the DNSSEC types?
      - After quick look at the *DNS.h RR_Type enum* & corresponding *DNS.cc:*

```
58      TYPE_TSIG = 250,    ///< Transaction Signature (RFC 2845)
59      TYPE_CAA = 257,     ///< Certification Authority Authorization (RFC 6844)
60
61      // The following are only valid in queries.
62      TYPE_AXFR = 252,
63      TYPE_ALL = 255,
64      TYPE_WINS = 65281,  ///< Microsoft's WINS RR
65      TYPE_WINSR = 65282, ///< Microsoft's WINS-R RR
66  } RR_Type;
```

```
311         case TYPE_TSIG:
312             status = ParseRR_TSIG(msg, data, len, rdlength, msg_start);
313             break;
314
315         default:
```

Top Down Approach

I chose to follow TYPE_TSIG….

- Analysis:
  - Let's start from DNS.cc – main code file

```
311         case TYPE_TSIG:
312             status = ParseRR_TSIG(msg, data, len, rdlength, msg_start);
313             break;
314
```

Function call that parses TSIG RR

```
727 int DNS_Interpreter::ParseRR_TSIG(DNS_MsgInfo* msg,
728                     const u_char*& data, int& len, int rdlength,
729                     const u_char* msg_start)
730     {
```

Function that parses TSIG RR

```
761     val_list* vl = new val_list;
762
763     vl->append(analyzer->BuildConnVal());
764     vl->append(msg->BuildHdrVal());
765     vl->append(msg->BuildTSIG_Val());
766
767     analyzer->ConnectionEvent(dns_TSIG_addl, vl);
768
769     return 1;
770     }
```

New Conn Event 'dns_TSIG_addl' for TSIG RR

New Record Type 'dns_tsig_additional' for TSIG RR

```
1084 Val* DNS_MsgInfo::BuildTSIG_Val()
1085     {
1086     RecordVal* r = new RecordVal(dns_tsig_additional);
1087     double rtime = tsig->time_s + tsig->time_ms / 1000.0;
1088
```

Top Down Approach - Analysis So Far:

From DNS.cc -

- We chose already parsed DNS RR Type – “TYPE_TSIG”
- We found the function that parses the TSIG RR – “ParseRR_TSIG”
- A new connection event used for TSIG RR – “dns_TSIG_addl”
- A New Record Type used for TSIG RR – “dns_tsig_additional”

Unknown -

- What other code files would need to be modified?
  - Where is the new Event declared and handled?
  - Where is the new Record Type declared/defined and what/how many values it has?

Top Down Approach - Finding the Unknown:

Lets do a quick search on “dns_tsig” in the src code:

```
Searching 6282 files for "dns_tsig"

/Users/fatemabw/bro-2.5.1/scripts/base/init-bare.bro:
 ....
 3100  ## An additional DNS TSIG record.
 3101  ##
 3102: ## .. bro:see:: dns_TSIG_addl
 3103: type dns_tsig_additional: record {
 3104   query: string;  ##< Query.
 3105   qtype: count;   ##< Query type.
 ....

/Users/fatemabw/bro-2.5.1/scripts/base/protocols/dns/main.bro:
  478  #
  479: #event dns_TSIG_addl(c: connection, msg: dns_msg, ans: dns_tsig_additional)
  480  #    {

/Users/fatemabw/bro-2.5.1/src/analyzer/protocol/dns/events.bif:
  ...
  492  ##    dns_rejected dns_request non_dns_request dns_max_queries dns_session_timeout
  493  ##    dns_skip_addl dns_skip_all_addl dns_skip_all_auth dns_skip_auth
  494: event dns_TSIG_addl%(c: connection, msg: dns_msg, ans: dns_tsig_additional%);

/Users/fatemabw/bro-2.5.1/src/NetVar.cc:
  120: RecordType* dns_tsig_additional;
  ...
  439:  dns_tsig_additional =
  440:      internal_type("dns_tsig_additional")->AsRecordType();

/Users/fatemabw/bro-2.5.1/src/NetVar.h:
  123: extern RecordType* dns_tsig_additional;
```

Top Down Approach - Summary

We chose “TYPE_TSIG” as a reference to code DNSSEC RR TYPES

Dry Analysis of the code showed following files would need extensions:

- src/analyzer/protocol/dns/DNS.h -> New Type/function declarations
- src/analyzer/protocol/dns/DNS.cc -> New Parser code
- src/NetVar.h & src/NetVar.cc -> New Record Type declaration/ definition
- scripts/base/init-bare.bro -> New Record Type definition
- src/analyzer/protocol/dns/events.bif -> New event declaration
- scripts/base/protocols/dns/main.bro -> Handles the event

Top Down Approach - Summary

Coding new RR types parsing (RRSIG, NSEC, NSEC3, DNSKEY, DS) support:

- Capture the sample traffic using tcpdump
- Analyze the packets captured in Wireshark
- Read and understand the RFCs for the Protocols & standard packet structure
- Code re-use -> no need to reinvent the wheel
  - All the basic functions to extract the bytes from the packet are already available in code
- Start from something simple-
  - code -> test -> troubleshoot -> repeat

Finally…. Few Take Aways

If you see something, do something!

Contribute back to the Zeek open source project –

- It’s fun to learn more about everyone's favorite and the most widely used NSM in the industry ☺
- You don’t have to be Pro in coding – smart techniques of code reference and code re-use will get you a long way!
- & more importantly you will develop a new skill set to show off!
  - (You might become one of few who speak hex!)

Acknowledgements ☺

- *Thanks to the Awesome Zeek Team for the support, and providing answers/solutions to all the Zeek related questions. [@zeek.org mailing list]*

- *Thanks for the opportunity to be a part of Zeek Workshop Europe 2019 !!*

Questions???