

Spicy and an RCE



- I am Evan Typanski :)
- Work for Corelight as a software engineer on Zeek
- Background with static code analysis (SAST)



First: What is Spicy?

Then, create a “real”
detection:

- 1) Understand the exploit
- 2) Parse the network traffic
- 3) Create the detection
- 4) Ensure the detection triggers



- Spicy is an incremental parser generator, specialized for network protocols

- Spicy is an **incremental** parser generator, specialized for network protocols
 - Incremental: Parsing occurs chunk-by-chunk as it arrives, rather than with an entire payload

- Spicy is an incremental **parser generator**, specialized for network protocols
 - Incremental: Parsing occurs chunk-by-chunk as it arrives, rather than with an entire payload
 - Parser generator: Spicy generates programs that parse network traffic

- Spicy is an incremental parser generator, specialized for **network protocols**
 - Incremental: Parsing occurs chunk-by-chunk as it arrives, rather than with an entire payload
 - Parser generator: Spicy generates programs that parse network traffic
 - Protocols: Spicy is specialized for network protocols, but it can also parse files (mainly for transmission over the network)
 - Spicy is capable; it can do more. But it is primarily for network protocols

- Spicy parses “units” which describe the protocol
- All of the units together make a “grammar”

```
12 public type RequestLine = unit {  
13     method: Token;  
14     :      WhiteSpace;  
15     uri:    Token;  
16     :      WhiteSpace;  
17     version: Version;  
18     :      NewLine;  
19  
20     on %done {  
21         print self.method, self.uri, self.version.number;  
22     }  
23 };
```

- Depending on what is in a message, you could go down many different routes to parse that message
- Spicy gives multiple ways to switch what you parse based on earlier data

```
3 public type Packet = unit {  
4   opcode: uint16;  
5  
6   switch ( self.opcode ) {  
7     1 → rrq:   ReadRequest;  
8     2 → wrq:   WriteRequest;  
9     3 → data:   Data;  
10    4 → ack:   Acknowledgement;  
11    5 → error: Error;  
12  };  
13  
14  on %done { print self; }  
15 };
```

... and so much more

```
public type Foo = unit {  
  x: bytes &ood &convert=$$.to_uint();  
  on %done { print self; }  
};
```

```
public type Foo = unit {  
  x: int8 &requires=($$ < 5);  
  on %done { print self; }  
};
```

```
type Bar = unit(msg: string, mult int8) {  
  x: int8 &convert=($$ * mult);  
  on %done { print "%s: %d" % (msg, self.x); }  
};
```

```
public type Foo = unit {  
  y: Bar("My multiplied integer", 5);  
};
```

```
type Foo = unit {  
  version: uint32;  
} &byte-order=spicy::ByteOrder::Little;
```

```
module Test;
```

```
public type Foo = unit {  
  a: int8;  
  b: int8 if ( self.a == 1 );  
  c: int8 if ( self.a % 2 == 0 );  
  d: int8;  
  on %done { print self; }  
};
```

```
module Test;
```

```
public type test = unit {  
  foo: Foo &try;  
  bar: Bar;  
  on %done { print self; }  
};
```

```
type Foo = unit {  
  a: int8 {  
    if ( $$ != 1 )  
      self.backtrack();  
  }  
  b: int8;  
};
```

```
type Bar = unit {  
  a: int8;  
  b: int8;  
};
```

```
module Test;
```

```
public type Foo = unit {  
  data: uint8[];  
    : /EOD/;  
  x : int8;  
  on %done { print self; }  
};
```

- Spicy was made for Zeek, but it's fully independent
- Do you want to parse protocols with simple-ish syntax and handling?
Think about making a custom host application

Understanding the Exploit

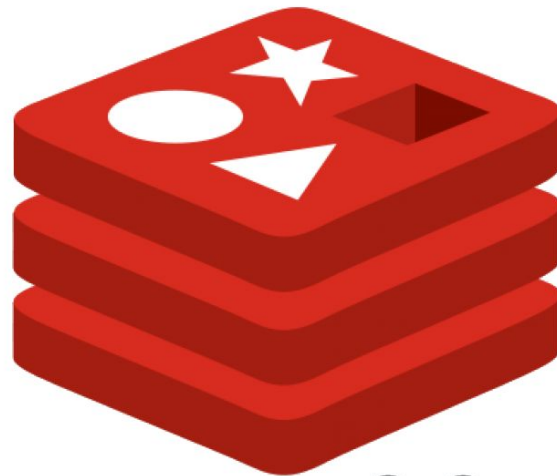


From [Crowdstrike](#):

REMOTE CODE EXECUTION (RCE): PRINCIPLES AND FUNCTION

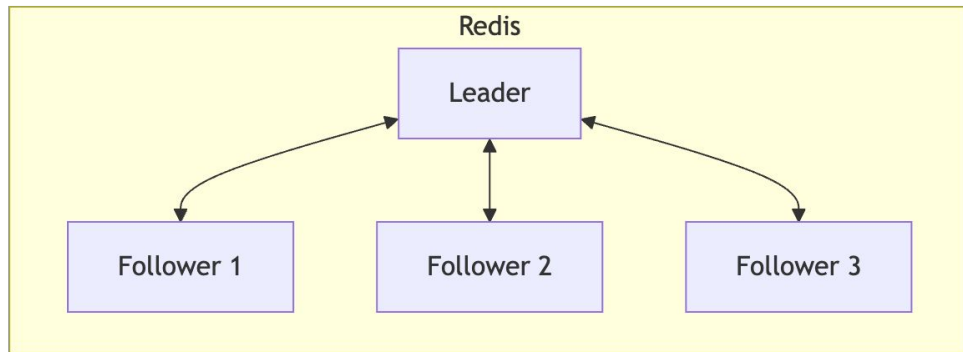
Remote code execution (RCE) refers to a class of cyberattacks in which attackers remotely execute commands to place malware or other malicious code on your computer or network. In an RCE attack, there is no need for user input from you. A remote code execution vulnerability can compromise a user's sensitive data without the hackers needing to gain physical access to your network.

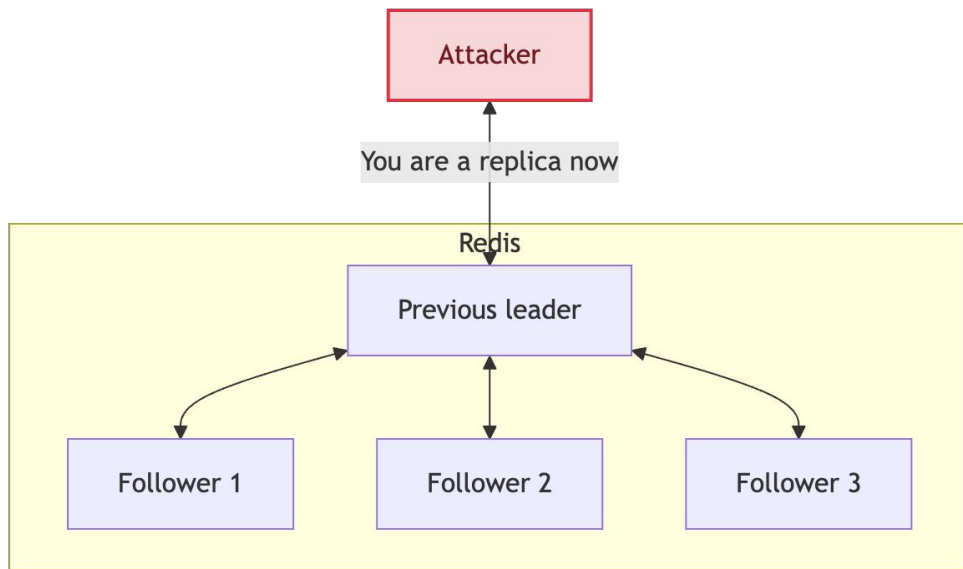
- Redis is a key/value store
- Widely used for many applications:
 - Caching
 - Message broker
 - Generic database
- Generally run on a separate server
- Clients connect to this server to get and set values from keys



redis

- Redis servers can “replicate” a leader





- Redis servers can “replicate” a leader
- The attacker connects to the unauthenticated leader
- The attacker tells the leader that it’s a replica

1) Attacker tells leader it's a replica

[illegible]

- [illegible]

- 1) Attacker tells leader it's a replica
- 2) Replica tries to sync its database; gets sent an ELF file???
- 3) Attacker loads that ELF file
- 4) :(



Parsing

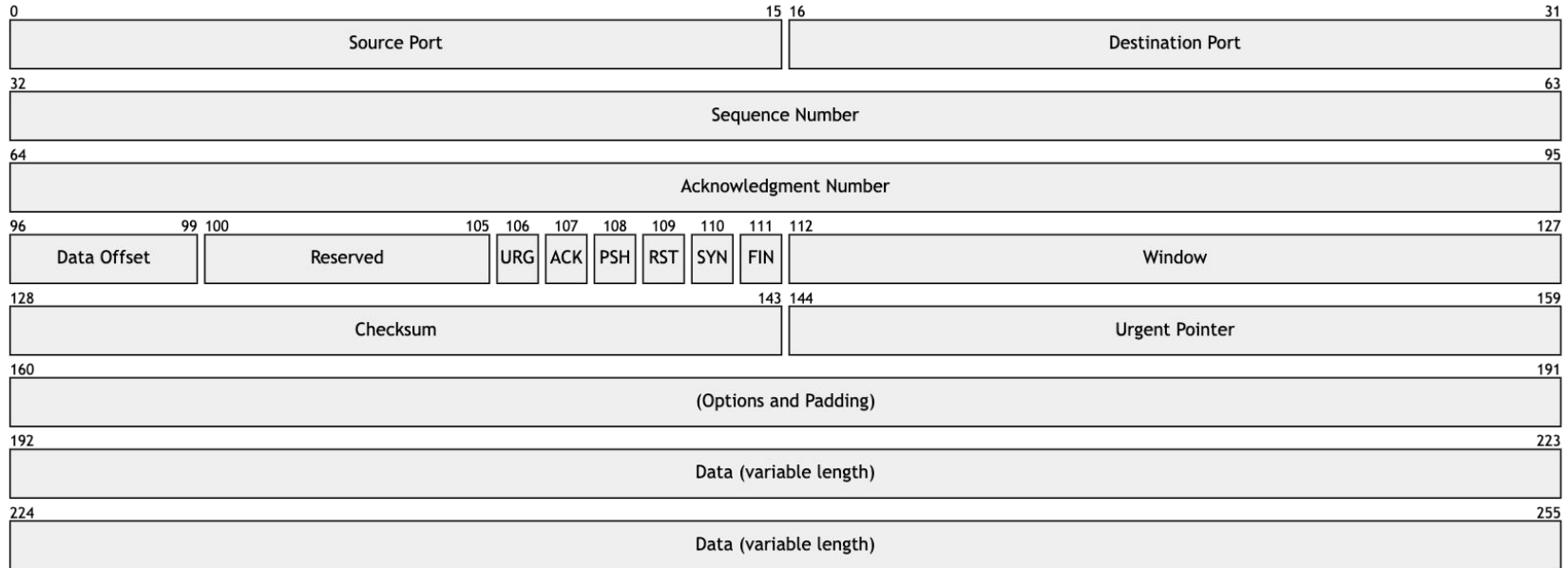


- Parsing is transforming one form of data into data structures

```
00000000: 11010100 11000011 10110010 10100001 00000010 00000000
00000006: 00000100 00000000 00000000 00000000 00000000 00000000
0000000c: 00000000 00000000 00000000 00000000 11111111 11111111
00000012: 00000000 00000000 00000001 00000000 00000000 00000000
00000018: 10101110 00001001 00111001 01010001 11100000 01000011
0000001e: 00001101 00000000 01001110 00000000 00000000 00000000
00000024: 01001110 00000000 00000000 00000000 00000000 00010000
0000002a: 11011011 10001000 11010010 11101111 11001000 10111100
00000030: 11001000 10010110 11010010 10100000 00001000 00000000
00000036: 01000101 00000000 00000000 01000000 11011111 10001110
0000003c: 01000000 00000000 01000000 00000110 01101101 11010011
00000042: 10001101 10001110 11100100 00000101 11000000 10010110
00000048: 10111011 00101011 11101001 11010000 00000000 01010000
0000004e: 11111110 00100001 00110010 00111010 00000000 00000000
00000054: 00000000 00000000 10110000 00000010 11111111 11111111
0000005a: 00111011 11000000 00000000 00000000 00000010 00000100
00000060: 00000101 10110100 00000001 00000011 00000011 00000100
00000066: 00000001 00000001 00001000 00001010 00010110 01001010
0000006c: 11011101 00100000 00000000 00000000 00000000 00000000
00000072: 00000100 00000010 00000000 00000000 10101110 00001001
00000078: 00111001 01010001 01001100 01010100 00001110 00000000
0000007e: 01001010 00000000 00000000 00000000 01001010 00000000
00000084: 00000000 00000000 11001000 10111100 11001000 10010110
0000008a: 11010010 10100000 00000000 00010000 11011011 10001000
00000090: 11010010 11101111 00001000 00000000 01000101 00000000
00000096: 00000000 00000000 00111100 00000000 01000000 00000000
0000009c: 00101111 00000110 01011110 01100110 11000000 10010110
000000a2: 10111011 00101011 10001101 10001110 11100100 00000101
000000a8: 00000000 01010000 11101001 11010000 10100101 10101111
000000ae: 11001110 00111110 11111110 00100001 00110010 00111011
000000b4: 10100000 00010010 00111000 10010000 00101110 01010000
```

```
GET /download/CHANGES.bro-aux.txt HTTP/1.1
User-Agent: Wget/1.14 (darwin12.2.0)
Accept: */*
Host: bro.org
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Thu, 07 Mar 2013 21:43:07 GMT
Server: Apache/2.4.3 (Fedora)
Last-Modified: Wed, 29 Aug 2012 23:49:27 GMT
ETag: "1261-4c870358a6fc0"
Accept-Ranges: bytes
Content-Length: 4705
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/plain; charset=UTF-8
```



TCP Packet

Redis Serialization Protocol (RESP)

RESP data type	Minimal protocol version	Category	First byte
Simple strings	RESP2	Simple	+
Simple Errors	RESP2	Simple	-
Integers	RESP2	Simple	:
Bulk strings	RESP2	Aggregate	\$
Null bulk strings	RESP2	Aggregate	\$_1\r\n
Arrays	RESP2	Aggregate	*
Nulls	RESP3	Simple	_
Booleans	RESP3	Simple	#
Doubles	RESP3	Simple	,
Big numbers	RESP3	Simple	(
Bulk errors	RESP3	Aggregate	!
Verbatim strings	RESP3	Aggregate	=
Maps	RESP3	Aggregate	%
Attributes	RESP3	Aggregate	
Sets	RESP3	Aggregate	~
Pushes	RESP3	Aggregate	>

Redis Serialization Protocol (RESP) - with Spicy

RESP data type	Minimal protocol version	Category	First byte
Simple strings	RESP2	Simple	+
Simple Errors	RESP2	Simple	-
Integers	RESP2	Simple	:
Bulk strings	RESP2	Aggregate	\$
Null bulk strings	RESP2	Aggregate	\$_1\r\n
Arrays	RESP2	Aggregate	*
Nulls	RESP3	Simple	-
Booleans	RESP3	Simple	#
Doubles	RESP3	Simple	,
Big numbers	RESP3	Simple	(
Bulk errors	RESP3	Aggregate	!
Verbatim strings	RESP3	Aggregate	=
Maps	RESP3	Aggregate	%
Attributes	RESP3	Aggregate	
Sets	RESP3	Aggregate	~
Pushes	RESP3	Aggregate	>

```
134 type DataType = enum {  
135     SIMPLE_STRING = '+',  
136     SIMPLE_ERROR = '-',  
137     INTEGER = ':',  
138     BULK_STRING = '$',  
139     ARRAY = '*',  
140     NULL = '_',  
141     BOOLEAN = '#',  
142     DOUBLE = ',',  
143     BIG_NUM = '(',  
144     BULK_ERROR = '!',  
145     VERBATIM_STRING = '=',  
146     MAP = '%',  
147     ATTRIBUTE = '|',  
148     SET = '~',  
149     PUSH = '>',  
150 };
```

- Then we “**just**” parse the data

```
89 type Data = unit {
90   %synchronize-after = b"\x0d\x0a";
91   ty: uint8 &convert=DataType($$);
92
93   switch (self.ty) {
94     DataType::SIMPLE_STRING → simple_string: SimpleString(False);
95     DataType::SIMPLE_ERROR → simple_error: SimpleString(True);
96     DataType::INTEGER → integer: Integer;
97     DataType::BULK_STRING → bulk_string: BulkString(False);
98     DataType::ARRAY → array: Array(depth);
99     DataType::NULL → null: Null_;
100    DataType::BOOLEAN → boolean: Boolean;
101    DataType::DOUBLE → double: Double;
102    DataType::BIG_NUM → big_num: BigNum;
103    DataType::BULK_ERROR → bulk_error: BulkString(True);
104    DataType::VERBATIM_STRING → verbatim_string: BulkString(False);
105    DataType::MAP → map_: Map(depth);
106    DataType::SET → set_: Set(depth);
107    DataType::PUSH → push: Array(depth);
108  };
109};
```

```
134 type SimpleString = unit(is_error: bool) {
135   content: RedisBytes;
136};
```

* 2 \r\n

\$ 5 \r\n

Hello \r\n

\$ 9 \r\n

Berkeley! \r\n

*	2	\r\n
\$	5	\r\n
	Hello	\r\n
\$	9	\r\n
	Berkeley!	\r\n

\r\n - Carriage Return
Line Feed

Terminator - separates
the protocol's parts

*	2
---	---

 \r\n

\$ 5 \r\n

Hello \r\n

\$ 9 \r\n

Berkeley! \r\n

Array

Array Length (2)

```
* 2 \r\n
$ 5 \r\n
  Hello \r\n
$ 9 \r\n
  Berkeley! \r\n
```

Bulk String

String length (5)

String content (Hello)

* 2 \r\n

\$ 5 \r\n

Hello \r\n

\$	9	\r\n
----	---	------

Berkeley!	\r\n
-----------	------

Bulk String

String length (9)

String content
(Berkeley!)

The Detection (tm)



- We have a couple ways to detect the RCE:
 1. The attacker transferred an ELF file instead of a database file
 2. The attacker used some troubling commands (SLAVEOF, MODULE LOAD)
 3. The attacker connected to the Redis server without authentication
- Point 1 may be tough
 - The transferred file could be a non-ELF file
 - The communication between the server and attacker was actually not RESP, it just looks like it
- Point 3 is also the whole reason we're in this mess in the first place!
- So, I will show how to detect point 2: an allow-list of commands

How should we detect this?

- ...
- An allow list of commands!

- Spicy talks with Zeek via events in a .evt file

```
20# All client data is a command
21on RESP::ClientData → event Redis::command($conn, self.command);
22
23on RESP::ServerData if ( Redis::classify(self) = Redis::ReplyType::Reply ) →
24|   event Redis::reply($conn, Redis::make_server_reply(self));
25on RESP::ServerData if ( Redis::classify(self) = Redis::ReplyType::Error ) →
26|   event Redis::error($conn, Redis::make_server_reply(self));
27on RESP::ServerData if ( Redis::classify(self) = Redis::ReplyType::Push ) →
28|   event Redis::server_push($conn, Redis::make_server_reply(self));
```

```
1 @load frameworks/notice
2 @load base/protocols/redis
3
4 export {
5     global allowlist: set[string] = ["ping", "config", "replconf", "psync",
6         "module", ] &redef;
7 }
8
9 redef enum Notice::Type += {
10     Bad_Redis_Command,
11 };
12
13 event Redis::command(c: connection, cmd: Redis::Command)
14 {
15     if ( to_lower(cmd$name) !in allowlist )
16         NOTICE([$note=Bad_Redis_Command, $msg=fmt("Disallowed Redis command: %s",
17             cmd$name), $id=c$id, $identifier=cat(cmd$name, c$id$orig_h,
18             c$id$resp_h)]);
19 }
```

```
1 @load frameworks/notice
2 @load base/protocols/redis
3
4 export {
5     global allowlist: set[string] = ["ping", "config", "replconf", "psync",
6         "module", ] &redef;
7 }
8
9 redef enum Notice::Type += {
10     Bad_Redis_Command,
11 };
12
13 event Redis::command(c: connection, cmd: Redis::Command)
14 {
15     if ( to_lower(cmd$name) !in allowlist )
16         NOTICE([$note=Bad_Redis_Command, $msg=fmt("Disallowed Redis command: %s",
17             cmd$name), $id=c$id, $identifier=cat(cmd$name, c$id$orig_h,
18             c$id$resp_h)]);
19 }
```

Is the command allowed?

```
1 @load frameworks/notice
2 @load base/protocols/redis
3
4 export {
5     global allowlist: set[string] = ["ping", "config", "replconf", "psync",
6         "module", ] &redef;
7 }
8
9 redef enum Notice::Type += {
10     Bad_Redis_Command,
11 };
12
13 event Redis::command(c: connection, cmd: Redis::Command)
14 {
15     if ( to_lower(cmd$name) !in allowlist )
16         NOTICE([ $note=Bad_Redis_Command, $msg=fmt("Disallowed Redis command: %s",
17             cmd$name), $id=c$id, $identifier=cat(cmd$name, c$id$orig_h,
18             c$id$resp_h) ]);
19 }
```

If not, raise a notice

```
1 @load frameworks/notice
2 @load base/protocols/redis
3
4 export {
5     global allowlist: set[string] = ["ping", "config", "replconf", "psync",
6         "module", ] &redef;
7 }
8
9 redef enum Notice::Type += {
10     Bad_Redis_Command,
11 };
12
13 event Redis::command(c: connection, cmd: Redis::Command)
14 {
15     if ( to_lower(cmd$name) !in allowlist )
16         NOTICE([$note=Bad_Redis_Command, $msg=fmt("Disallowed Redis command: %s",
17             cmd$name), $id=c$id, $identifier=cat(cmd$name, c$id$orig_h,
18             c$id$resp_h)]);
19 }
```

The logged notice

```

c:\Users\otyp\src\redis-rce\tests\jj:(r:xtv) $ cat notice.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path notice
#open 2026-05-18-15-44-57
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p fvid file_mime_type file_desc proto note msg sub src dst p n peer_de
scr actions email_dest suppress_for remote_location.country_code remote_location.region remote_location.city remote_location.latitude remote_location.longitude
#types time string addr port addr port string string string enum enum string string addr addr port count string set[enum] set[string] interval s
tring string string double double
1776094434.392233 - 172.17.0.1 61332 172.17.0.2 6379 - - - tcp Bad_Redis_Command Disallowed Redis command: SLAVEOF - 172.17.0.1 1
72.17.0.2 6379 - Notice::ACTION_LOG (empty) 3600.000000 - - - - - - - - -
1776094441.173486 - 172.17.0.1 61332 172.17.0.2 6379 - - - tcp Bad_Redis_Command Disallowed Redis command: system.exec - 172.17.0.1 1
72.17.0.2 6379 - Notice::ACTION_LOG (empty) 3600.000000 - - - - - - - - -
#close 2026-05-18-15-44-57

```


Using zeek-cut:

```
etyp:/Users/etyp/src/redis-rce/tests jj:(r:xtv) $ cat notice.log | zeek-cut -m id.orig_h id.resp_h note msg
id.orig_h      id.resp_h      note           msg
172.17.0.1      172.17.0.2     Bad_Redis_Command Disallowed Redis command: SLAVEOF
172.17.0.1      172.17.0.2     Bad_Redis_Command Disallowed Redis command: system.exec
```

Detected



“Vanilla” Zeek logs also work!

- Zeek also has logs for each protocol
- You can get the details to detect this RCE just from “vanilla” Zeek logs

[illegible]

Red flag 1: system.exec

system.exec is not a standard Redis command

```
etyp:/Users/etyp/src/redis-rce/tests jj:(r:xtv) $ cat redis.log | zeek-cut -m cmd.name  
cmd.name  
SLAVEOF  
CONFIG  
PING  
REPLCONF  
REPLCONF  
PSYNC  
-  
MODULE  
SLAVEOF  
system.exec  
system.exec  
system.exec  
system.exec  
CONFIG  
system.exec  
MODULE
```

ELF files have known “magic bytes” — That’s not a database!

[illegible]



Spicy Tips and Tricks!



```
1 module StateBomb;
2
3 const MiBPerByte = 1024 * 1024;
4
5 type ByteBomb = unit {
6     byte: uint8;
7     var extra_crap: vector<uint8>;
8     on %init { self.extra_crap.resize(MiBPerByte); }
9 };
10
11 public type Input = unit {
12     bombs: ByteBomb[] &eod;
13 };
```

```
11 public type Input = unit {
12     bombs: ByteBomb[] &eod &max-size=5000;
13 };
```

ParseError

```
[1] 41825 killed spicy-driver
```

Bound State Growth

```
local limit = zeek::get_count("Redis::max_aggregate_elements");
```

```
[error] resp.spicy:9:1-9:12: cannot import module 'zeek': cannot find file  
[error] spicyc: aborting after errors
```

```
12 @if HAVE_ZEEK  
13 import zeek;  
14 global _max_aggregate_elements: optional<uint64>; # appease spicy-format with a comment  
15 public function max_aggregate_elements(): uint64 {  
16     if (!_max_aggregate_elements)  
17         _max_aggregate_elements = zeek::get_count("Redis::max_aggregate_elements");  
18  
19     return *_max_aggregate_elements;  
20 }  
21 @else  
22 public function max_aggregate_elements(): uint64 {  
23     return 20000;  
24 }  
25 @endif
```

Go faster!

155.4 μ s -> 52.8 μ s

```
1 module SlowNestedUnits;
2
3 public type EncodedOctet = unit {
4     value: uint8;
5 };
6
7 public type Pad1 = unit {
8     value: EncodedOctet;
9 };
10
11 public type Flags = unit {
12     value: EncodedOctet;
13 };
14
15 public type HopLimit = unit {
16     value: EncodedOctet;
17 };
18
19 public type QueueClass = unit {
20     value: EncodedOctet;
21 };
22
23 public type Header = unit {
24     kind: uint8;
25
26     switch (self.kind) {
27         0 → pad: Pad1;
28         1 → flags: Flags;
29         2 → hop_limit: HopLimit;
30         3 → queue_class: QueueClass;
31     };
32 };
33
34 public type Slow = unit {
35     headers: Header[] &eod &max-size=4096;
36 };
```

```
1 module FastNestedUnits;
2
3 type Header = unit {
4     kind: uint8;
5
6     switch (self.kind) {
7         0 → pad: uint8;
8         1 → flags: uint8;
9         2 → hop_limit: uint8;
10        3 → queue_class: uint8;
11    };
12 };
13
14 public type Fast = unit {
15     : Header[] &eod &max-size=4096;
16 };
```

Thank You!

etyp@zeek.org

WordArt generator (no affiliation):

www.makewordart.com