# Behavior-Driven Development in Malware Analysis: Can it Improve the Malware Analysis Process?

**Thomas Barabosch**
**thomas.barabosch@fkie.fraunhofer.de**
**SPRING 2015, Neubiberg bei München**

# REIMPLEMENTATION TASK

# Problem Statement: REimplementation task

- REimplentation of certain algorithms
  - DGA, e.g. network detection, anticipation of CC-servers
  - Crypto, e.g. for opening network traffic
- Fundamental part of the malware analysis process
- System specifications given by malware sample
- Hypothesize and corroborate hypotheses until system specifications derived and implemented

# State-of-the-art

- Slicing, e.g. Inspector Gadget [Kolbitsch2010]
    - Still needs manual intervention
    - Cannot cope with obfuscated code
- Iterative Reengineering Process (Smalltalk to Java, documentation available) [Durelli2010]
- Modern malware analysis processes are already agile (SCRUM) [Plohmann2013]

# Current situation

- Scientific state-of-the-art solutions
  - are not publicly available
  - do not work with current malware
    - at least without preparations like deobfuscation
- Most Analysts merely translate from machine code to higher language
  - Code's correctness is not ensured
  - Code's readability is often very poor
  - Colleagues have a hard time during integration

## What do we need in order to improve the malware analysis process?

- Inspector Gadget on steroids!
  - Unlikely: too many unresolved problems
- Change the way how we think about the REimplementation task
  - describing observations in clear, spoken language
  - continuously ensuring the correctness of the code during reimplementation
  - writing code documentation on the go

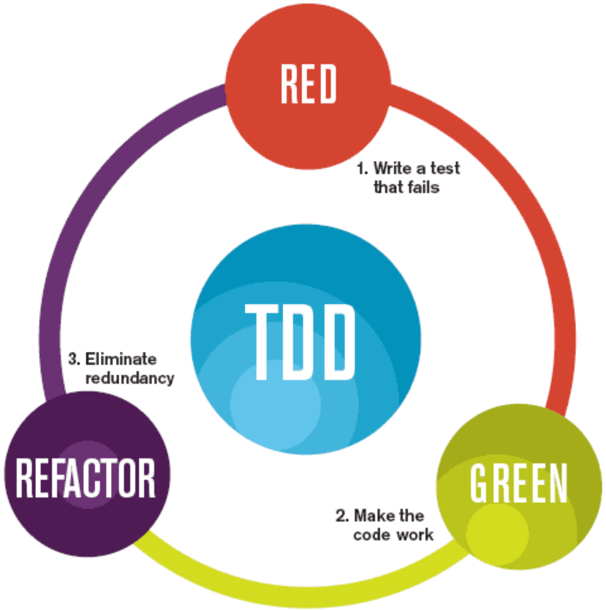
# *-DRIVEN-DEVELOPMENT

# In the beginning there was Software Testing...

- Main objective: showing quality of a software to stake-holders
- Test whether a software does what it is supposed to do
- Find defects and failures in a software
  - Input space is at least very large…
- But also test non-functional requirements
  - Performance, Scalability, Usability, Reliability, …
- Problems
  - Infrequent testing due to long testing circles (e.g. Waterfall model)
  - Code coverage

# Test Driven Development (TDD)

- Short development cycle
  - Write a failing test
  - Write code to make the test pass
  - Refactor the code
- Ideally ensures 100% coverage
- Small and comprehensive code base due to frequent refactoring
- Tests serve as a documentation of the code




The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Source: http://luizricardo.org/wordpress/wp-content/upload-files/2014/05/tdd_flow.gif

# Behavior Driven Development (BDD)

- BDD focuses on a clear understanding of the software’s behavior rather then modules, functions, etc.
- Test cases are formulated in natural language
- Hoare logic -> {P} C {Q}
- BDD community still discusses… [North2015]

# Behavior Driven Development Example

```
Story: Returns go to stock

In order to keep track of stock
As a store owner
I want to add items back to stock when they're returned

Scenario 1: Refunded items should be returned to stock
Given a customer previously bought a black sweater from me
And I currently have three black sweaters left in stock
When he returns the sweater for a refund
Then I should have four black sweaters in stock

Scenario 2: Replaced items should be returned to stock
Given that a customer buys a blue garment
And I have two blue garments in stock
And three black garments in stock.
When he returns the garment for a replacement in black,
Then I should have three blue garments in stock
And two black garments in stock
```

https://en.wikipedia.org/wiki/Behavior-driven_development

# BDD in the malware analysis process

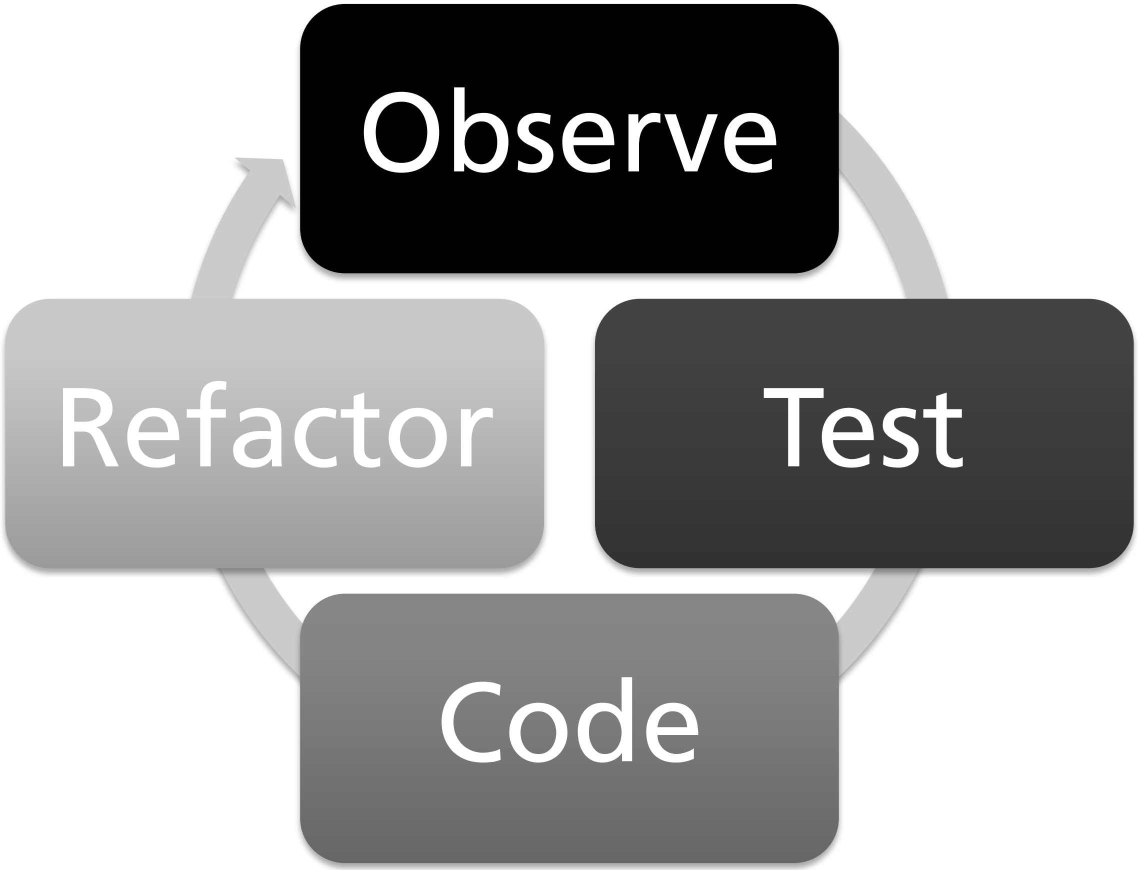
- First pinpoint the algorithm in the binary
  - Find entry point and exits
  - Extract initial test data for acceptance test and state acceptance test

Source: https://trak-1.com/wp-content/uploads/2014/10/haystack.jpg

# BDD in the malware analysis process

- Then we enter a cycle consisting of four steps
  - (1) Observe behavior statically/dynamically and gather test data
  - (2) Write a failing test that expresses clearly the observations in natural language
  - (3) Write code that satisfies the observations and passes the test
  - (4) Refactor code

# BDD in the malware analysis process



# Putting the first step under the microscope

- Top-Down-Approach
  - Getting a rough overview
  - Identifying individual features and their interfaces (e.g. function calls)
- Gather test data at interfaces (input/output)
  - Use this data for mocking in the next step
  - Mock interfaces of submodules at first

# Benefits of BDD in malware analysis

- Writing an observation down in simple words
  - reflect, understand, explain
  - “If you can't **explain** it simply, you don't **understand** it well enough.” (attributed to Albert Einstein)
- Delivery of concise code that comes with examples
- Insurance that the code works continuously

# Possible pitfalls

- Getting started
  - Identify the interfaces
    - Guess related API calls…
  - Then write first end-to-end acceptance test
- Getting lost in details
  - Gathering to much irrelevant test data
  - Writing to many unnecessary tests

# CASE STUDY: NYMAIM

# Nymaim

- Nymaim is a malware dropper
  - But also credential stealer, SOCKS, etc.
- Heavily obfuscated -> Won’t decompile
  - See Spring 2014 presentation of [Plohmann2014]

```
optval= byte ptr -4

push    ebp
mov     ebp, esp
push    ecx
push    ebx
push    edi
mov     edi, eax
mov     eax, esi
call    sub_1001E837
mov     [esi+18h], edi
imul    edi, 3E8h
push    4                           ; optlen
lea     eax, [ebp+optval]
push    eax                         ; optval
push    1005h                       ; optname
mov     ebx, 0FFFFh
push    ebx                         ; level
push    dword ptr [esi]             ; s
mov     dword ptr [ebp+optval], edi
mov     edi, ds:setsockopt
call    edi ; setsockopt
test    eax, eax
jz      short loc_1001E1DA

loc_1001E1C8:                       ; CODE XREF: sub_1001E18E+5E↓j
mov     dword ptr [esi+10h], 3
call    ds:WSAGetLastError
mov     [esi+14h], eax
jmp     short loc_1001E1EE
; ---------------------------------------------------------------------

loc_1001E1DA:                       ; CODE XREF: sub_1001E18E+38↑j
push    4                           ; optlen
lea     eax, [ebp+optval]
push    eax                         ; optval
push    1006h                       ; optname
push    ebx                         ; level
push    dword ptr [esi]             ; s
call    edi ; setsockopt
test    eax, eax
jnz     short loc_1001E1C8
```

- Unpacked Dridex
- Regular functions
- No strange constants
- Resolved imports
- Reasonable control flow
- ...

```
; Attributes: bp-based frame

sub_4617B72 proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch

; FUNCTION CHUNK AT seg000:000034D6 SI;
; FUNCTION CHUNK AT seg000:0000BFF1 SI;
; FUNCTION CHUNK AT seg000:00014729 SI;

push     ebp
mov      ebp, esp
push     eax
push     ecx
jmp      loc_46034D6
sub_4617B72 endp

; ---------------------------------------
lea      esi, [ebp-1Ch]
push     63h ; 'c'
call     sub_460A4C2
push     66E7E05Bh
push     66E82D2Ch
call     sub_460CACB
mov      ecx, [esi]
add      ecx, [esi+4]
mov      eax, 99ADDFB1h
call     sub_461AB04
add      eax, ecx
mov      [ebp-2Ch], eax
mov      eax, 9FA6BD27h
call     sub_461AB04
add      eax, ecx
mov      [ebp-28h], eax
mov      eax, 9F3EAD68h
call     sub_4603580
cvtps2pd xmm2, xmm3
pop      ecx

; ============== S U B R O U T I N E
```

- Unpacked Nymaim
- Irregular functions
  - Function entries
  - Function ends

```
; Attributes: bp-based frame

sub_4617B72 proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch

; FUNCTION CHUNK AT seg000:000034D6 SI;
; FUNCTION CHUNK AT seg000:0000BFF1 SI;
; FUNCTION CHUNK AT seg000:00014729 SI;

push    ebp
mov     ebp, esp
push    eax
push    ecx
jmp     loc_46034D6
sub_4617B72 endp

; ---------------------------------------
lea     esi, [ebp-1Ch]
push    63h ; 'c'
call    sub_460A4C2
push    66E7E05Bh
push    66E82D2Ch
call    sub_460CACB
mov     ecx, [esi]
add     ecx, [esi+4]
mov     eax, 99ADDFB1h
call    sub_461AB04
add     eax, ecx
mov     [ebp-2Ch], eax
mov     eax, 9FA6BD27h
call    sub_461AB04
add     eax, ecx
mov     [ebp-28h], eax
mov     eax, 9F3EAD68h
push    0A62CBC97h
cvtps2pd xmm2, xmm3
pop     ecx

; =============== S U B R O U T I N E =
```

- Unpacked Nymaim
- Irregular functions
  - Function entries
  - Function ends
- Strange constants

```
; Attributes: bp-based frame

sub_4617B72 proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch

; FUNCTION CHUNK AT seg000:000034D6 SI;
; FUNCTION CHUNK AT seg000:0000BFF1 SI;
; FUNCTION CHUNK AT seg000:00014729 SI;

push    ebp
mov     ebp, esp
push    eax
push    ecx
jmp     loc_46034D6
sub_4617B72 endp

; ------------------------------------
lea     esi, [ebp-1Ch]
push    63h ; 'c'
call    sub_460A4C2
push    66E7E05Bh
push    66E82D2Ch
call    sub_460CACB
mov     ecx, [esi]
add     ecx, [esi+4]
mov     eax, 99ADDFB1h
call    sub_461AB04
add     eax, ecx
mov     [ebp-2Ch], eax
mov     eax, 9FA6BD27h
call    sub_461AB04
add     eax, ecx
mov     [ebp-28h], eax
mov     eax, 9F3EAD68h
push    0A62CBC97h
call    sub_4603580
cvtps2pd xmm2, xmm3
pop     ecx

; ============== S U B R O U T I N E
```

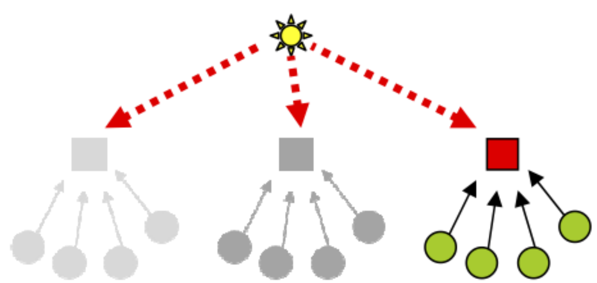
- Unpacked Nymaim
- Irregular functions
  - Function entries
  - Function ends
- Strange constants
- Control flow computed dynamically

```
; Attributes: bp-based frame

sub_4617B72 proc near


arg_0= dword ptr  8
arg_4= dword ptr  0Ch

; FUNCTION CHUNK AT seg000:000034D6 SI
; FUNCTION CHUNK AT seg000:0000BFF1 SI
; FUNCTION CHUNK AT seg000:00014729 SI

push    ebp
mov     ebp, esp
push    eax
push    ecx
jmp     loc_46034D6
sub_4617B72 endp

; ------------------------------------
lea     esi, [ebp-1Ch]
push    63h ; 'c'
call    sub_460A4C2
push    ebx
push    66E7E05Bh
push    66E82D2Ch
call    sub_460CACB
mov     ecx, [esi]
add     ecx, [esi+4]
mov     eax, 99ADDFB1h
call    sub_461AB04
add     eax, ecx
mov     [ebp-2Ch], eax
mov     eax, 9FA6BD27h
call    sub_461AB04
add     eax, ecx
mov     [ebp-28h], eax
mov     eax, 9F3EAD68h
push    0A62CBC97h
call    sub_46[illegible]
cvtps2pd xmm2, xmm3
pop     ecx

; =============== S U B R O U T I N E
```

- Unpacked Nymaim
- Irregular functions
  - Function entries
  - Function ends
- Strange constants
- Control flow computed dynamically
- Confuses disassembler

# Recap: What is a Domain Generation Algorithm (DGA)?

- Locomotive botnets
- There are four classes of DGAs [Barabosch2012]
  - Time-dependent/time-independent
  - Deterministic/non-deterministic



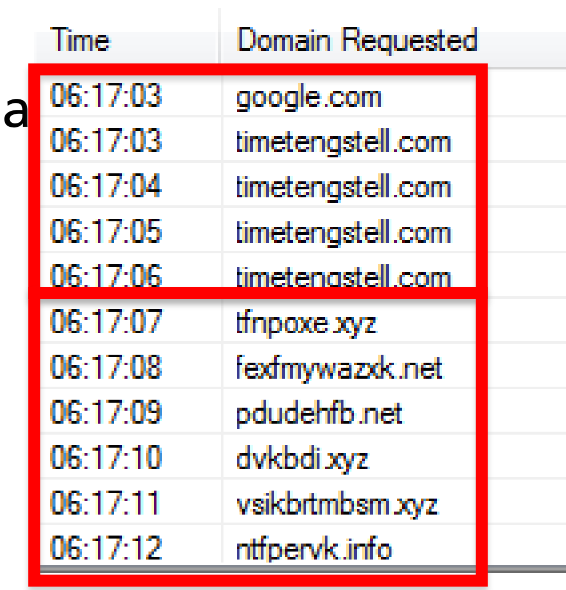
[Leder2009]

# Nymaim's DGA – Tools of trade and resources

- Tools of trade
  - Immunity Debugger 1.85
  - Mandiant ApateDNS 1.0
  - IDA Pro 6.8
  - Python 2.7.9
  - Behave 1.2.5 [Behave2015]
- Send me an email for source code + IDB

# Nymaim's DGA – First observations

- Black-boxing shows that
  - At first four hard-coded domain are resolved and contacted
  - In case of failure domains are generated and resolved
  - Deterministic: same results in two different VMs
  - Time-dependent: different results when da changed
- Pinpointing the algorithm
  - Breaking on GetSystemTime -> Bingo!
  - Input: time
  - Output: 30 domain names



| Time | Domain Requested |
|---|---|
| 06:17:03 | google.com |
| 06:17:03 | timetengstell.com |
| 06:17:04 | timetengstell.com |
| 06:17:05 | timetengstell.com |
| 06:17:06 | timetengstell.com |
| 06:17:07 | tfnpoxe.xyz |
| 06:17:08 | fexfmywazxk.net |
| 06:17:09 | pdudehfb.net |
| 06:17:10 | dvkbdi.xyz |
| 06:17:11 | vsikbrtmbsm.xyz |
| 06:17:12 | ntfpervk.info |

# Nymaim's DGA – Our first test: Acceptance test

- We know already many important parameters
  - Interfaces of algorithm
- Also we have gathered a first set of test data
  - Time information and list of generated domains
- We write our first end-to-end acceptance test
  - It does not pass
  - However, once it passes we are done!

# Nymaim's DGA – Our first test: Acceptance test

```
Scenario: Nymaim DGA computes domains of 2015-06-12
Given the day is "2015-06-12"
When DGA computes domains for this date
Then the domais for this date are
| domains       |
| dmjdfotcy.in  |
| yjcmub.info  |
| uiismpexr.info  |
| rszsgpzivi.info  |
```

```
Failing scen
  features/d                                    ains of 2015-06-12

0 features p
0 scenarios
2 steps pass
Took 0m0.002
```

# Nymaim‘s DGA – Two algorithms

- While stepping over the code we have noticed that there
  - is an initialization
  - are two algorithms
    - main logic
    - PRNG
- For now, we focus on one component at a time
  - Reverse the main logic, mock the rest!

# Nymaim's DGA – Main logic

```
push    dword ptr [ebp-30h]
push    6
push    edx
push    9169F53Dh
push    6E9591F2h
call    sub_4601335
lea     ecx, [eax+6]
lea     ebx, [esi+4]

loc_46162A8:
call    sub_46031ED
push    dword ptr [ebp-30h]
push    5Dh ; ']'
call    obfuscateRegisterPush
push    edx
push    984951E2h
push    67B63528h
call    sub_46029EF
mov     [ebx], al
call    sub_4613862
add     [ebx], al
inc     ebx
dec     ecx
jnz     short loc_46162A8
call    sub_460D912
mov     [ebx], al
inc     ebx
push    dword ptr [ebp-30h]
push    6
push    esi
push    56D194D2h
push    56D20DF2h
call    sub_4614592
inc     eax
dec     eax
jz      tld_ru
dec     eax
jz      tld_net
dec     eax
jz      tld_in
dec     eax
jz      tld_com
dec     eax
jz      tld_xyz
call    deobfuscateString
mov     [ebx], eax
mov     dword ptr [ebx+4], 0
add     ebx, 5

loc_4616326:

lea     eax, [esi+4]
sub     ebx, esi
mov     [esi+2], bx
add     esi, ebx
dec     dword ptr [ebp-8]
jnz     loc_46162A8
```

```python
def generateDomains(self):
    domains = []
    for i in range(0, DOMAIN_COUNT):
        domain = self.generateDomain()
        domains.append(domain)
    return domains

def generateDomain(self):
    lenDomain = self.computeLengthOfDomain()
    domain = ""
    for j in range(lenDomain):
        domain += self.computeChar()
    domain += "."
    tld = self.computeTld()
    domain += tld
    return domain
```

# Nymaim's DGA – Main logic

- Test only the main logic, e.g. choosing of the TLD
- Mock the rest!
- Might require several scenarios

```
Scenario: Nymaim DGA chooses correct TLD from set of
          possible TLDs ["ru","net","in","com","xyz", "info"]
Given the seeds
  | seed     |
  | 78670654 |
  | 44370352 |
  | 35461477 |
  | 97912344 |
When DGA computes TLD
Then the TLD is ru
```

# Nymaim's DGA – PRNG

- Next we have a look at the PRNG
- Still we do not want to deal with the seeds
- Input: five integers (4* seed + modulo)
- Output:  integer [0, modulo - 1]
- Has side effects on the seeds !

# Nymaim's DGA – PRNG

```
xor     ecx, ecx
mov     eax, [ebp+arg_0]
or      eax, eax
setz    cl
or      eax, ecx
mov     esi, [ebp+arg_4]
imul    eax, 64h
or      eax, eax
jz      loc_46118AE
mov     edi, eax
mov     eax, [esi]
shl     eax, 0Bh
xor     eax, [esi]
mov     edx, [esi+4]
add     [esi], edx
mov     ecx, [esi+8]
add     [esi+4], ecx
mov     ebx, [esi+0Ch]
add     [esi+8], ebx
shr     ebx, 13h
xor     ebx, [esi+0Ch]
xor     ebx, eax
shr     eax, 8
xor     ebx, eax
mov     [esi+0Ch], ebx
mov     eax, ebx
add     eax, ecx
xor     edx, edx
div     edi
xchg    eax, edx
xor     edx, edx
mov     edi, 64h ; 'd'
div     edi
```

```python
def execute(self, seeds, modulo):

    a = cutTo32bits(seeds.seeds[0] << 11) ^ seeds.seeds[0]
    b = cutTo32bits(seeds.seeds[3] >> 19) ^ seeds.seeds[3]
    a = b ^ a ^ cutTo32bits(a >> 8)
    c = seeds.seeds[2]

    self._updateSeeds(seeds, a)

    return (cutTo32bits(a + c) % modulo) / 100

def _updateSeeds(self, s, a):
    s.seeds[0] = cutTo32bits(s.seeds[0] + s.seeds[1])
    s.seeds[1] = cutTo32bits(s.seeds[1] + s.seeds[2])
    s.seeds[2] = cutTo32bits(s.seeds[2] + s.seeds[3])
    s.seeds[3] = a
```

# Nymaim's DGA – PRNG

```
Scenario: PRNG works correctly
          for given seeds and modulo
Given the modulo 600
And the seeds
  | seed       |
  | 123172080  |
  | 79962903   |
  | 133504895  |
  | 2326822159 |
When PRNG executes
Then the output is 1
```

# Nymaim‘s DGA – Initialization

- Now we can focus on the initialization and seeds
  - Seeds are initialized (homework)
  - Seeds are updated every time the PRG is called (trivia

```
Scenario: Nymaim DGA is properly initialized on 2015-06-11
Given the day is "2015-06-11"
When initialize DGA on this date
Then the seeds are
  | seed     |
  | 78670654 |
  | 44370352 |
  | 35461477 |
  | 97912344 |
```

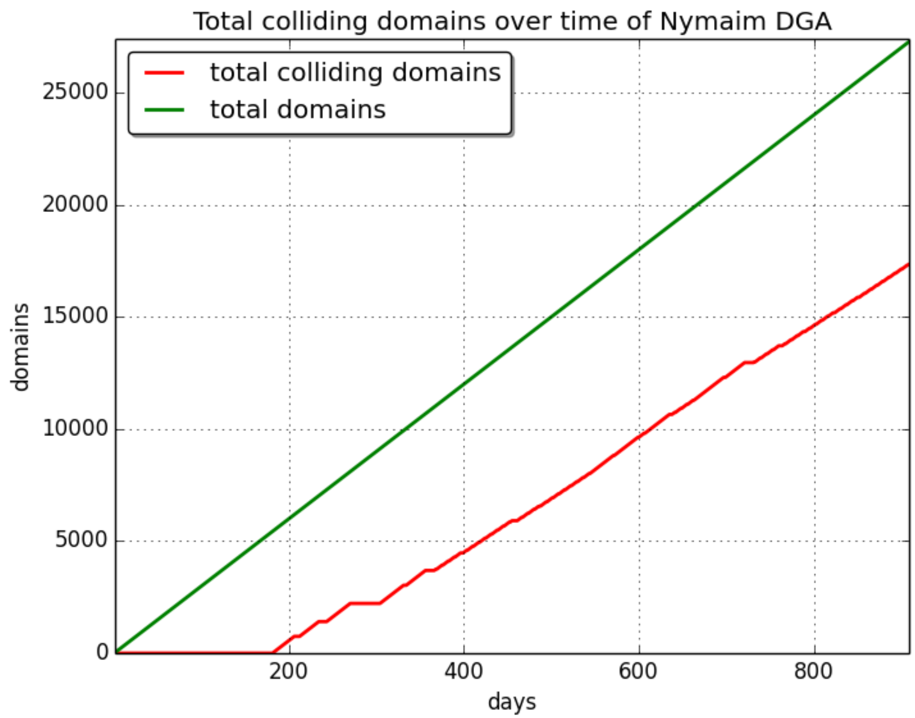
# Nymaim's DGA – Results

```
1 feature passed, 0 failed, 0 skipped
5 scenarios passed, 0 failed, 0 skipped
16 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.004s
```

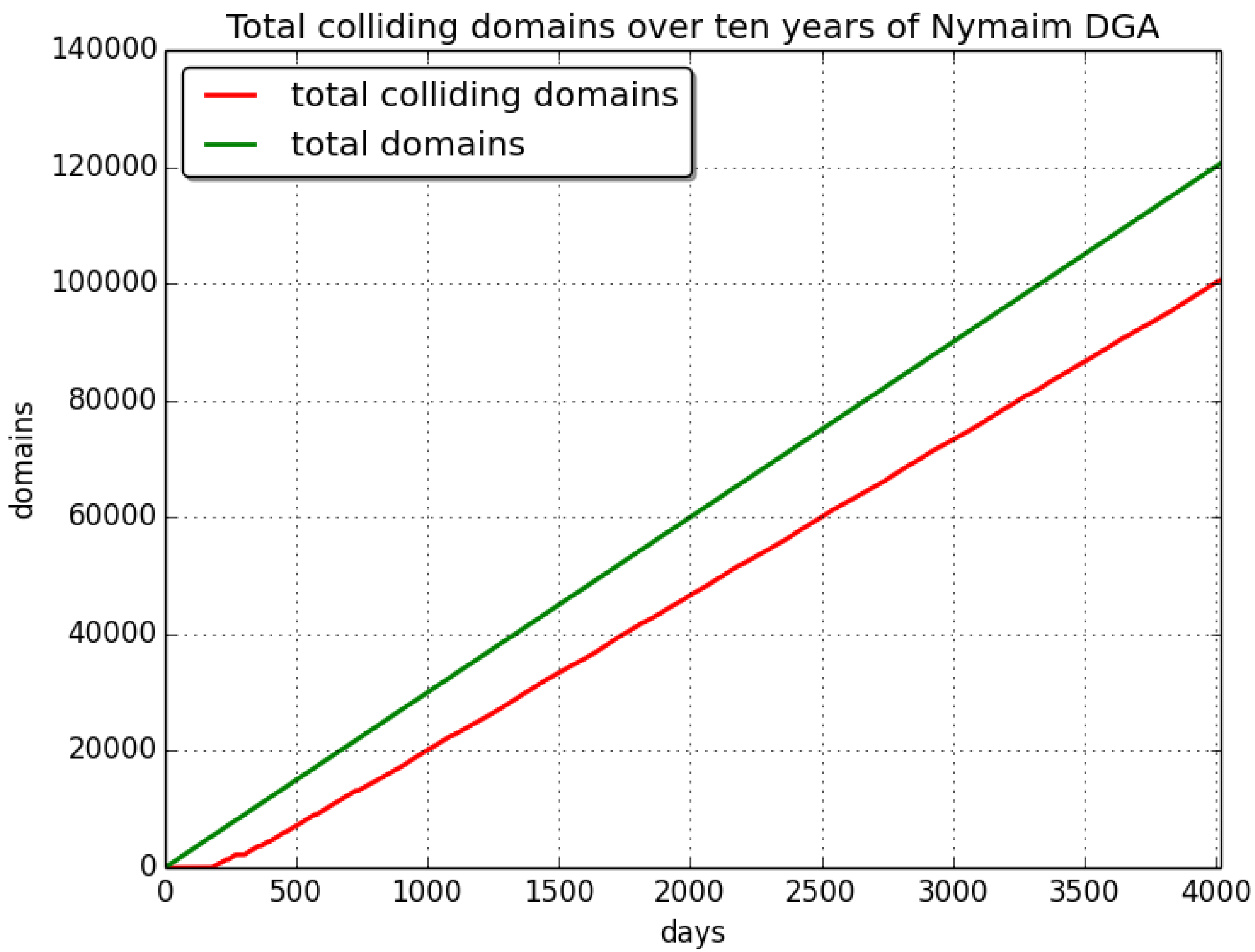
- Five tests of DGA's features
- One end-to-end acceptance test
- Readable code
  - One class implementing the main logic
  - One class implementing the PRNG (strategy pattern)
  - One class serving as data structure

# Nymaim's DGA – Collisions

- Algorithm results in a lot of collisions
- Based on 27300 generated domains (2013-01-01 - 2015-06-30)



# Nymaim's DGA – Collisions



# FUTURE WORK & CONCLUSION

# Future Work

- Towards Inspector Gadget on Steroids…
  - Deobfuscation
  - Feature detection
- More practical
  - Try out other testing processes
  - Automatic test case generation
  - Tools for gathering test data in RE context

# Conclusion

- Unfortunately, profound malware analysis continues to be highly manual work
- The result and efficiency of the REimplementation task can be improved by using BDD
- We showed the feasibility of BDD in a case study on the highly obfuscated DGA of Nymaim

# References


- [Barabosch2012] Barabosch et. al., Automatic Extraction of Domain Name Generation Algorithms from Current Malware, STO-MP-IST-111 2012
- [Behave2015] behave project, http://pythonhosted.org/behave/
- [Cucumber2015] cucumber project, Gherkin https://github.com/cucumber/cucumber/wiki/Gherkin
- [Durelli2010] Durelli et al., An Iterative Reengineering Process Applying Test-Driven Development and Reverse Engineering Patterns, 2010
- [Lott2012] Lott, TDRE - Test Driven Reverse Engineering Case Study, http://slott-softwarearchitect.blogspot.de/2012/02/tdre-test-driven-reverse-engineering.html
- [North2015] Dan North, BDD by Example, http://dannorth.net/2015/04/03/bdd-by-example/, 2015
- [Plohmann2013] Plohmann et al., Patterns of a Cooperative Malware Analysis Workflow, Cycon 2013
- [Plohmann2014] Plohmann, Patchwork: Stitching against malware families with IDA Pro, Spring 2014
- [Kolbitsch2010] Kolbitsch et al., Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries, S&P 2010




Fraunhofer FKIE