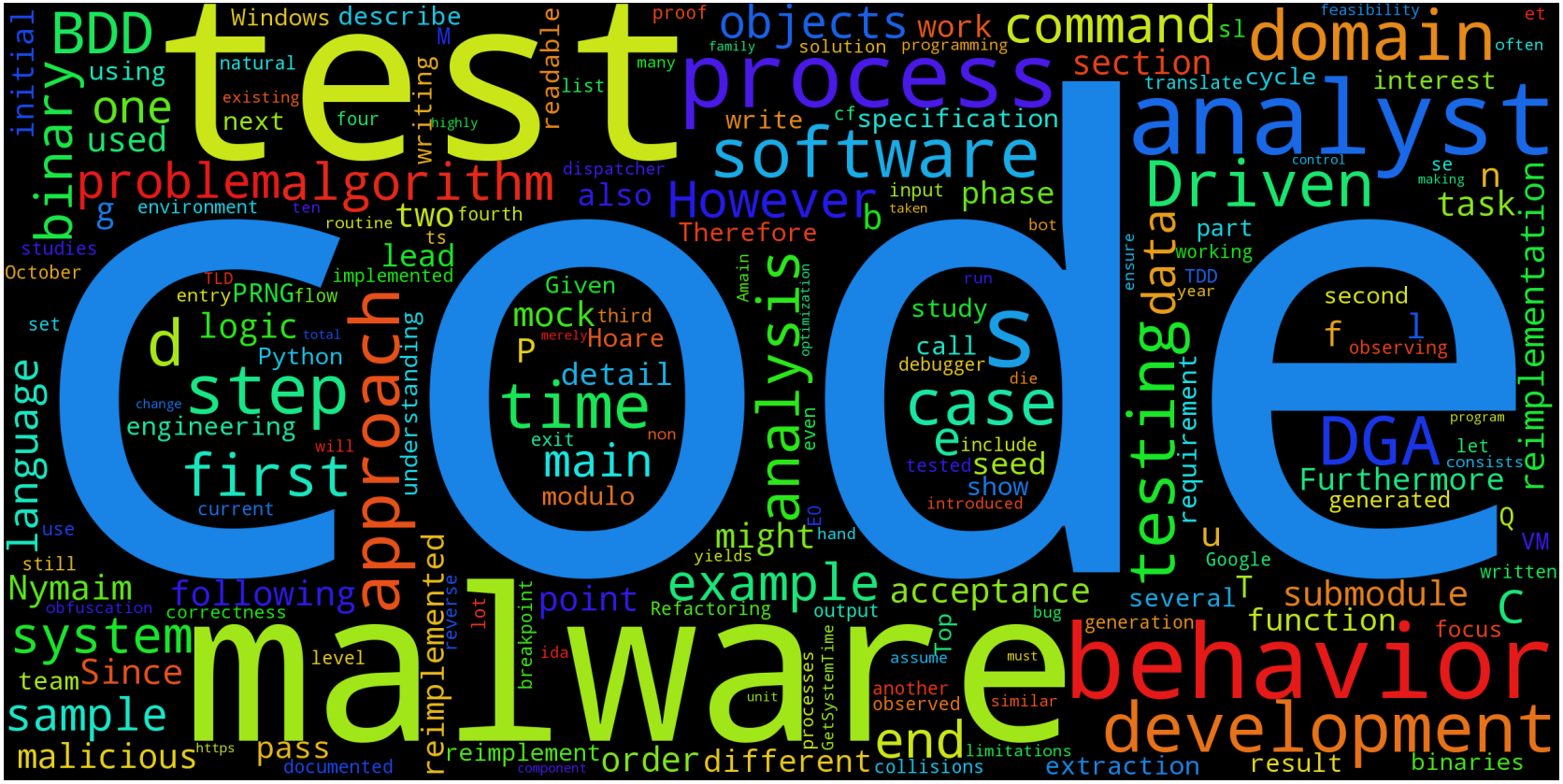
Behavior-Driven Development in Malware Analysis

Thomas Barabosch, Elmar Gerhards-Padilla
firstname.lastname@fkie.fraunhofer.de
Botconf 2015, Paris, France

Fraunhofer FKIE



Fraunhofer FKIE

Motivation

- Malware analysis continues to be a tedious and time consuming task (some might call it *job security…*)
- Extraction of malicious behavior is a daily task
  - Analyze (obfuscated) binary code
  - Reimplement in higher language like Python or C (*Reimplementation task*)
- Code is just "translated" from assembly to higher language
  - Functionality is not ensured
  - Readability is poor
  - No documentation
  - Underlying semantics not clear

Solution: Improve current process

Related Work

- Extraction of malicious behavior
  - [Caballero2010], [Kolbitsch2010], [Barabosch2012]
- Using TDD in RE processes
  - [VanLindberg2008], [DeSousa2010]

However, current state-of-the-art solutions

- are not publicly available
- can not cope with anti-analysis techniques
- can not cope with complex obfuscations
- assume source code and documentation available

Requirements of Solution

1. Allows the analyst to describe concisely and naturally what he observes
2. Ensures that the code works continuously during the implementation
3. Resulting code should be concise, documented and readable
4. Increases the focus of the analyst

Proposed Solution:

Apply Behavior-Driven Development to Malware Analysis

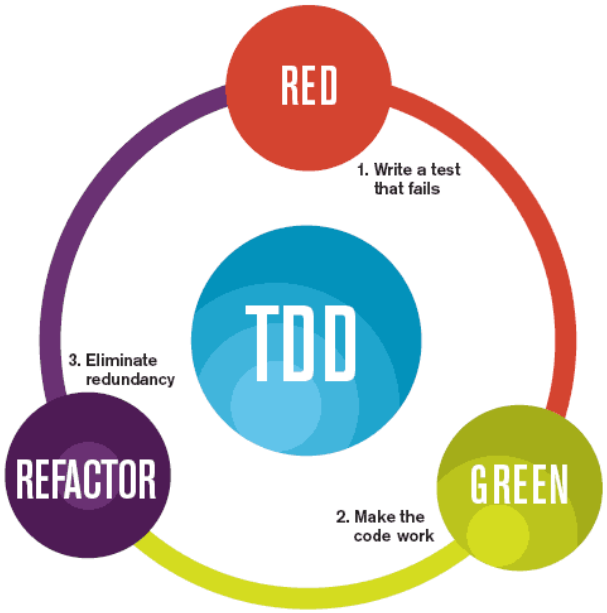
*-DRIVEN DEVELOPMENT

In the Beginning there was Software Testing...

- Tests whether a software does what it is supposed to do
- Shows quality of a software to stake-holders
- Finds defects and failures in a software
- Problems
  - Infrequent testing (e.g. Waterfall model)
  - Code coverage
  - Not efficient if done manually

Test Driven Development (TDD)

- Short development cycle
- 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 than modules, functions, etc.
- BDD emerged from TDD
- Test cases are formulated in natural language
- Strong theoretical foundation (Hoare logic)
  - {P} C {Q} -> Given _ When _ Then _

Behavior Driven Development (BDD)

Scenario: Coffee maker can add sugar to coffee

Given customer chooses sugar

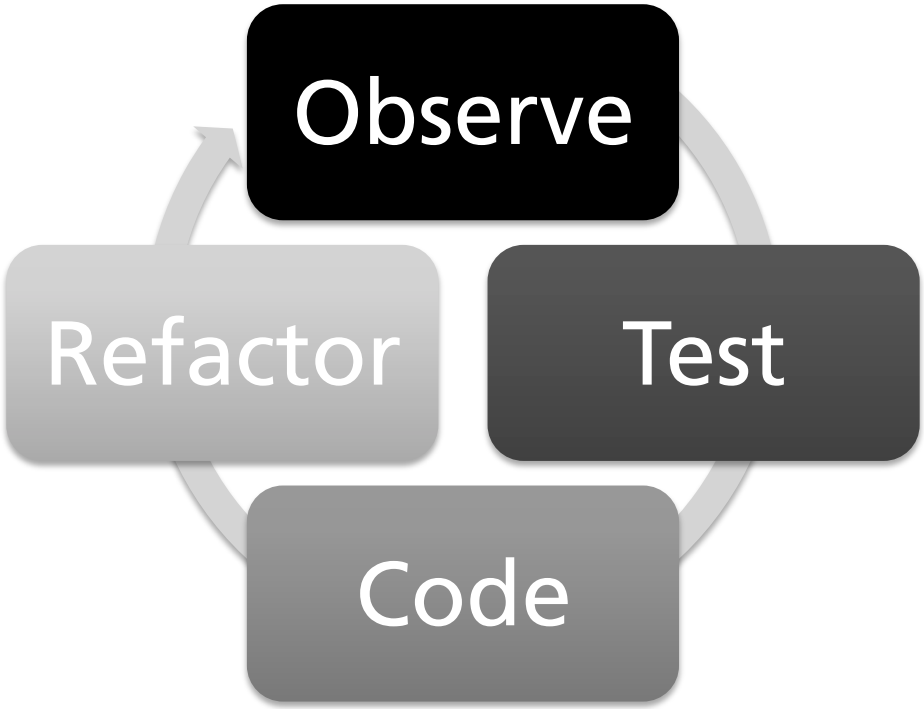
When customer presses OK button

Then coffee maker adds sugar to coffee

BDD IN MALWARE ANALYSIS

Overview of the Process

- Preparation phase
- Implementation phase (Observe – Test – Code – Refactor)



Preparation - Pinpointing the Behavior

- First pinpoint the behavior in the binary
  - Find entry point **S** and exits {$\mathbf{E_1}$, …, $\mathbf{E_n}$}
  - Extract initial test data for acceptance test
  - State acceptance test

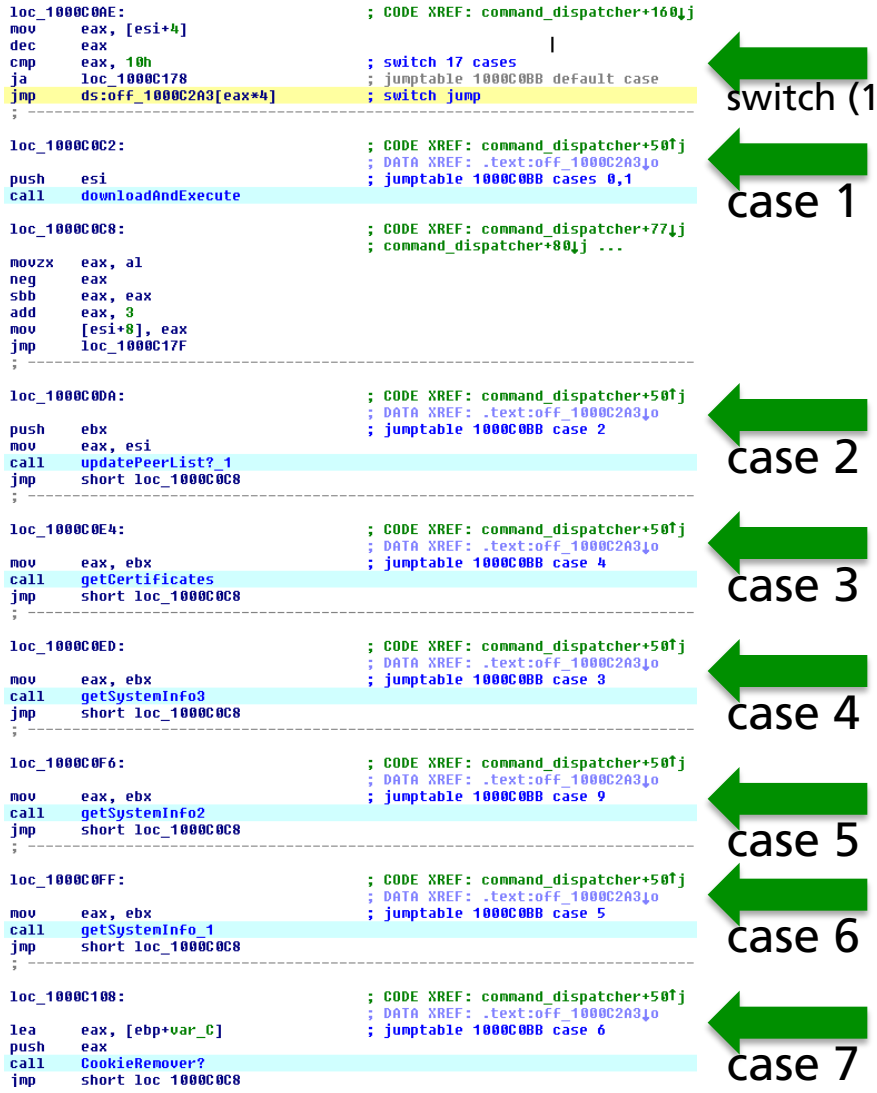
Source: https://trak-1.com/wp-content/uploads/2014/10/haystack.jpg

Pinpointing the Behavior (DGA)

- **D**omain **G**eneration **A**lgorithm
  - See Daniel's talk (*DGArchive – A deep dive into domain generating malware*)
- Several types of DGAs [Barabosch2012]
  - Deterministic/non-deterministic
  - Time-**dependent**/independent
- Naïve approach (forwards): look for timing sources
  - E.g. *GetSystemTime, NtQuerySystemTime, GetLocalTime*
- Naïve approach (backwards): DNS resolution
  - E.g. *gethostbyname*

Pinpointing the Behavior (command dispatcher)

- Bots implement several commands
- Bots receive and process messages of botmaster
  - Command dispatcher
- Naïve approach: follow data flow from network source
  - Monitor networking APIs like *receive*
  - Follow data flow in forwards direction until switch statement

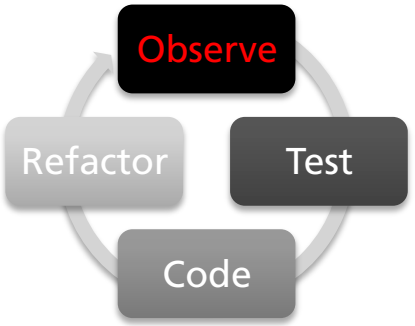


Example: Dridex

Preparation - Initial End-To-End Acceptance Test

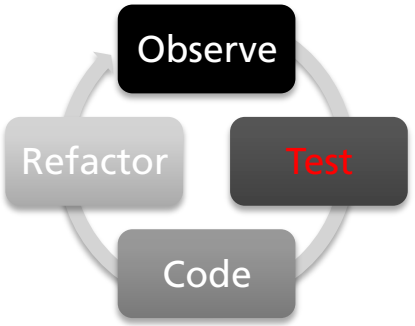
- Serves as guide throughout the implementation phase
- Tests behavior as a black box
- Capture data at **S** and {$\mathbf{E_1}$, …, $\mathbf{E_n}$}
- Once this test passes -> reimplementation successfully

Step 1: Observing the Behavior



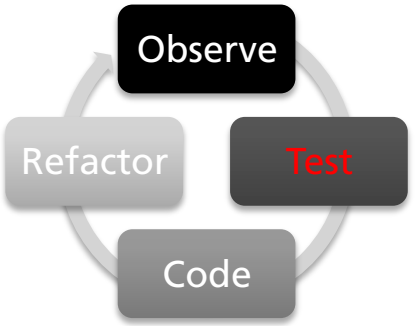
- Top-Down-Approach
  - Getting a rough overview
  - Identifying individual features and their interfaces
- Gather test data at interfaces (input/output)
  - Use this data for mocking later
  - Mock interfaces of submodules at first

Step 2: Writing a Test

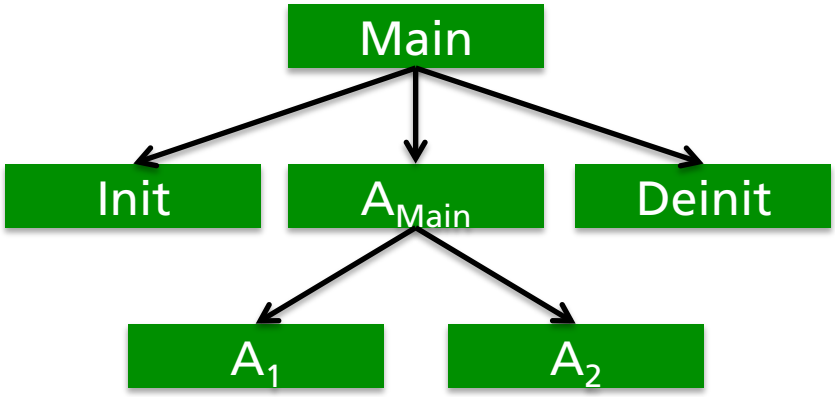


- *Given-Then-When*
- Fundamental: mock objects
  - Mimic the behavior of real objects
  - In software development, they replace, e.g., non-existing objects
  - In our case, they replace modules that are not 100% understood
  - Gather test data at module interfaces

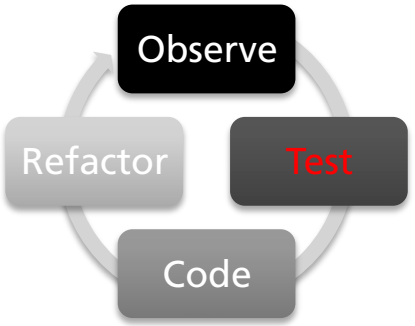
Step 2: Writing a Test



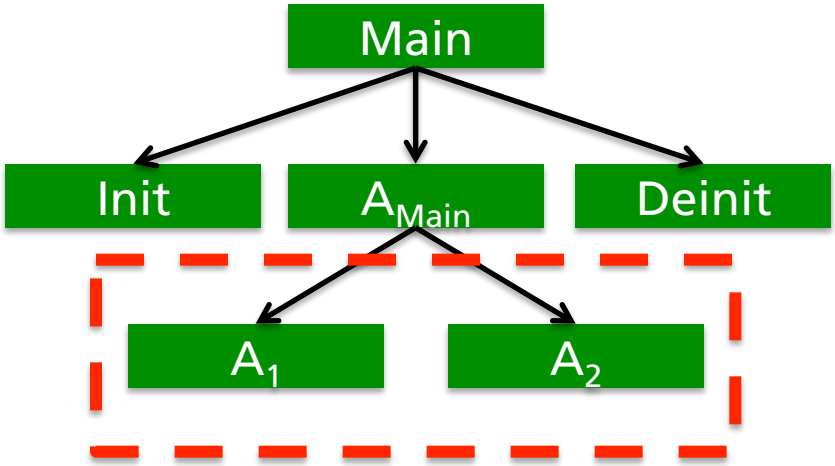
- *Given-Then-When*
- Fundamental: mock objects
  - Mimic the behavior of real objects
  - In software development, they replace, e.g., non-existing objects
  - In our case, they replace modules that are not 100% understood
  - Gather test data at module interfaces



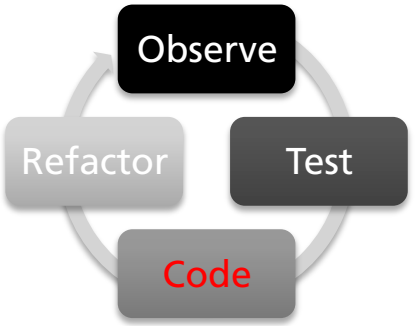
Step 2: Writing a Test



- *Given-Then-When*
- Fundamental: mock objects
  - Mimic the behavior of real objects
  - In software development, they replace, e.g., non-existing objects
  - In our case, they replace modules that are not 100% understood
  - Gather test data at module interfaces

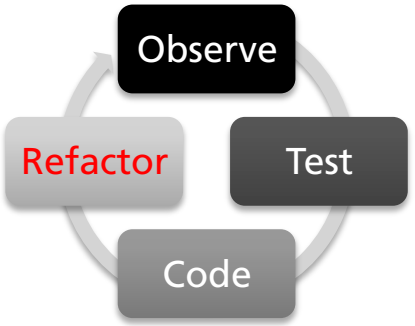


Step 3: Making the Test Pass



- Just write enough code to make the test pass
- Binary serves as valid system specification
- Focus and just implement code to make the test pass
- *"Premature optimization is the root of all evil"*

Step 4: Refactoring the Code



- Altering the syntax without altering the semantics
- Ensures conciseness and readability
- Many refactorings do exist (see also [Fowler1999])
  - Refactoring inlined code (*memcpy*)
  - Break up complex expressions
  - Removing dead expressions
- Does the end-to-end acceptance test pass?

Limitations

- Decrease in time efficiency
  - Extra time pays off due to benefits
  - TDD comes with an overhead of 15% to 35% [Bhat2006]
- TDD/BDD comes from “normal” software development
  - Reusability not needed in malware analysis
  - Long-running projects do exist also in the field of malware analysis

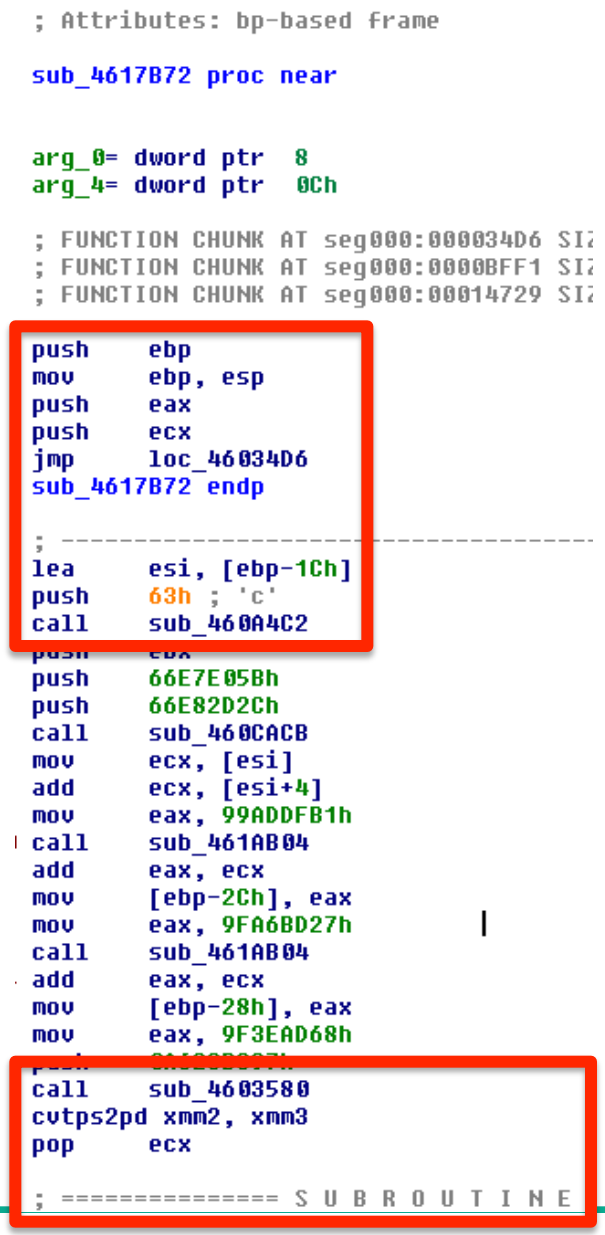
CASE STUDY NYMAIM DGA

Nymaim

- Nymaim is a malware dropper
  - But also credential stealer, SOCKS, etc.
- Heavily obfuscated
  - Decompilers fail to work
  - See *IDApatchwork* presentation of Daniel Plohmann

```
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
```

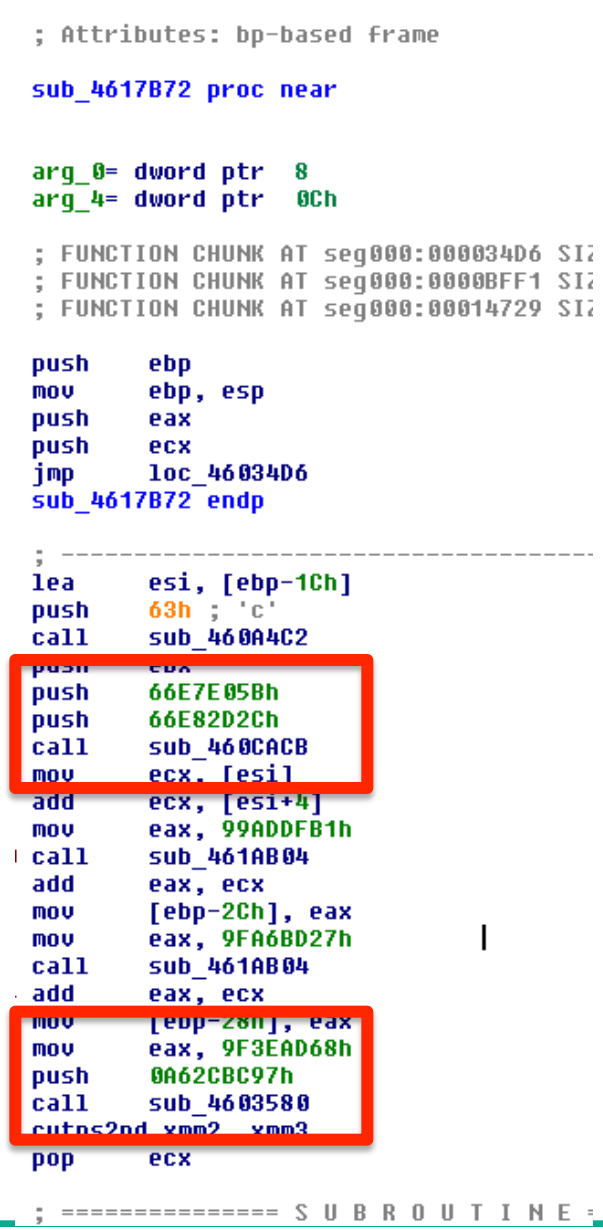
- Unpacked Dridex
- Regular functions
- No strange constants
- Resolved imports
- Reasonable control flow
- …



- 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    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_4603E80
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



- 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_4603E80
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

Nymaim's DGA – Tools of Trade and Resources

- Tools of trade
  - Immunity Debugger 1.85
  - IDA Pro 6.8
  - Mandiant ApateDNS 1.0
  - Python 2.7.9
  - Behave 1.2.5 [Behave2015]
- Source code on *Bitbucket*!
  - https://bitbucket.org/tbarabosch/botconf-2015-bdd-in-mw-analysis

Nymaim's DGA – First Observations

- Black-boxing shows that
  - At first four hard-coded domain are resolved and contacted

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 – 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 date changed

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 – 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 date 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  |
```

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 scenarios:
  features/dga.feature:5  Nymaim DGA computes domains of 2015-06-12

0 features passed, 1 failed, 0 skipped
0 scenarios passed, 1 failed, 0 skipped
2 steps passed, 1 failed, 0 skipped, 0 undefined
Took 0m0.002s
```

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
```

Fraunhofer FKIE

Nymaim's DGA – Two Algorithms

- While stepping over the code we have noticed that there
  - Initialization
  - Main logic
  - PRNG (Xorshift)
- 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_4616288
```

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_4616288
```

```
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

```
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_4616288
```

```
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. choose TLD
- Mock the rest!
- Might require several scenarios

Nymaim's DGA – Main Logic

- Test only the main logic, e.g. choose 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 (Xorshift)

- Next, we have a look at the PRNG (Xorshift)
- 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 (Xorshift)

```
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 – PRNG (Xorshift)

```
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
```

Nymaim's DGA – PRNG (Xorshift)

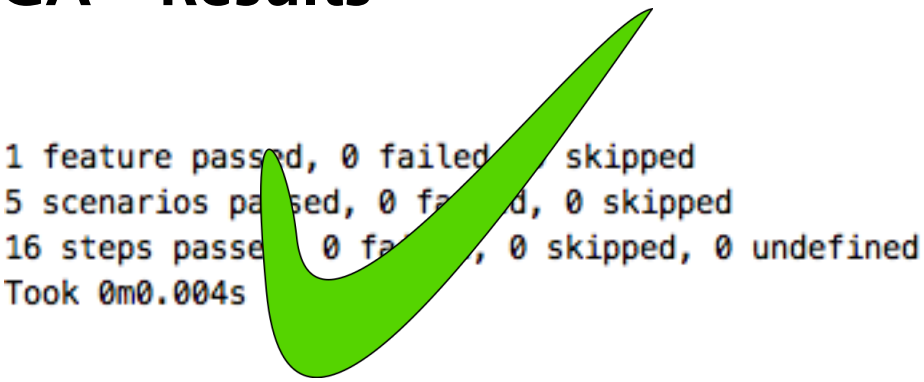
```
xor     ecx, ecx
mov     eax, [ebp+arg_0]
or      eax, eax
setz    cl
or      eax, ecx
mov     esi, [ebp+arg_4]
imul    eax, 64h
or      eax, eax
jz      loc_46118AE
mov     edi, eax
mov     eax, [esi]
shl     eax, 0Bh
xor     eax, [esi]
mov     edx, [esi+4]
add     [esi], edx
mov     ecx, [esi+8]
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
```

```
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
```

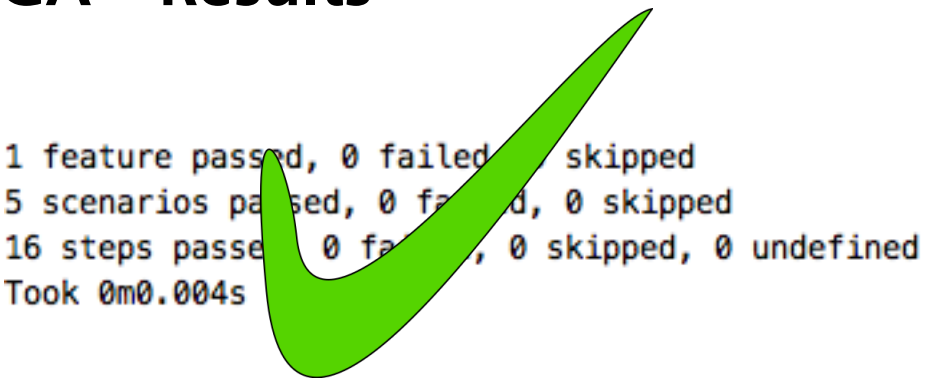
Nymaim's DGA – Results

```
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
```

Nymaim's DGA – Results

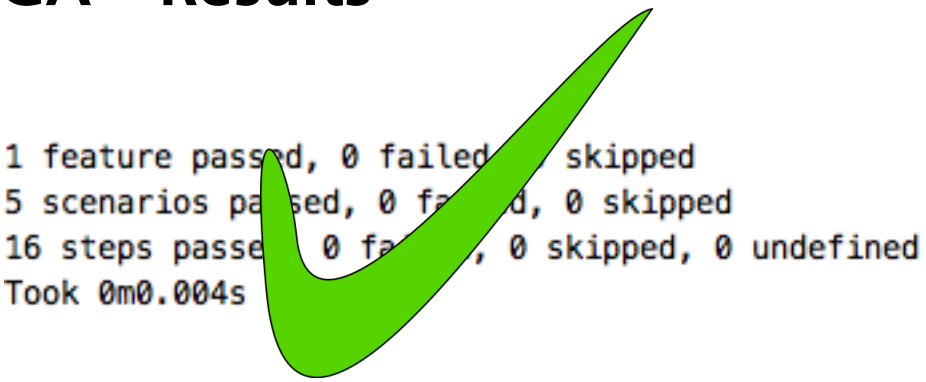


Nymaim's DGA – Results



- Five tests of DGA's features
- One end-to-end acceptance test

Nymaim's DGA – Results



- 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

CONCLUSION & FUTURE WORK

Conclusion & Future Work

- BDD in malware analysis
- Case Study Nymaim
  - Check source code on *Bitbucket*!
  - https://bitbucket.org/tbarabosch/botconf-2015-bdd-in-mw-analysis
- Future work
  - Automatic test case generation
  - Tools for gathering test data in RE context