

CERT-Conix tools: Machoke and BTG

CERT-Conix tools: Machoke and BTG

Robin Marsollier
CONIX CyberSecurity

-

10 April 2018

CERT-Conix tools: Machoke and BTG

Who am I?

Robin Marsollier

Twitter: [@rbnctl](#)

robin.marsollier@conix.fr

CERT-Conix

Machoke: CFG-based malware classification

—

BTG: OSINT tool

Machoke: CFG-based malware classification

CERT-Conix tools: Machoke and BTG

Machoke: definitions

- ➊ fuzzy hash

CERT-Conix tools: Machoke and BTG

Machoke: definitions

- ① fuzzy hash
- ② CFG : control flow graph

CERT-Conix tools: Machoke and BTG

Machoke: definitions

- ① fuzzy hash
- ② CFG : control flow graph
- ③ clusterisation

CERT-Conix tools: Machoke and BTG

Machoke: CFG-based classification

- ① Apparently used by AV
- ② Used by academics
- ③ Used by other actors

CERT-Conix tools: Machoke and BTG

Machoke: CFG-based classification

- ① Apparently used by AV
- ② Used by academics
- ③ Used by other actors
- ④ few public implementations ...

CERT-Conix tools: Machoke and BTG

Machoke: Objectives

- ① Get something better than md5/sha* (resistant to small changes inside samples notably, etc.)
- ② A fuzzy hash better than good old ssdeep
- ③ Get a small and independent tool easy to use and deploy at large
- ④ Let other tools do the clustering

CERT-Conix tools: Machoke and BTG

Machoke: Machoc

- ① Designed by ANSSI, published with Polichombr (<https://github.com/ANSSI-FR/polichombr>)
- ② CFG-based fuzzy hash
- ③ 2 implementations: Ruby/miasm || Python/IDAPython
(Machoc lost in lots of ruby/python/whatever code)

CERT-Conix tools: Machoke and BTG

Machoke: Naming



CERT-Conix tools: Machoke and BTG

Machoke: Core

- ① Radare2 + r2pipe
- ② Python

CERT-Conix tools: Machoke and BTG

Machoke: algorithm

```
[0x660] ;[gb]
;-- main:
fcn main 54
main ();
; var int local_14h @ rbp-0x14
; var int local_4h @ rbp-0x4
;     ; CALL XREF from 0x000006a6 (sym.functionl)
;     ; DATA XREF from 0x0000054d (entry0)
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [local_14h], edi
mov dword [local_4h], 0
mov dword [local_4h], 0
jmp 0x689;[ge]
```

v

```
0x689 ;[ga]
; JMP XREF from 0x00000679 (main)
; [0x4:4]=0x10102
cmp dword [local_4h], 4
jle 0x67b;[gd]
```

t

f

```
0x67b ;[gd]
; JMP XREF from 0x0000068d (main)
mov eax, dword [local_4h]
mov edi, eax
call sym.functionl;[gc]
add dword [local_4h], 1
```

```
0x68f ;[ge]
mov eax, 0
leave
ret
```

CERT-Conix tools: Machoke and BTG

Machoke: algorithm

The diagram illustrates the flow of control between four labeled blocks (1, 2, 3, 4) in the Machoke algorithm:

- Block 1:** Contains assembly code for the main function, including pushes, moves, and a jump to address 0x689.
- Block 2:** Contains a comparison (cmp) instruction followed by a jump if less than or equal (jle). It also contains a JMP XREF from address 0x689.
- Block 3:** Contains a call to the symbol `sym.function1`, which is circled in red.
- Block 4:** Contains the standard function prologue (mov eax, 0; leave; ret).

Control flow is indicated by dashed arrows: one arrow points from Block 1 to Block 2, another from Block 2 to Block 3, and a third from Block 3 to Block 4. A red dashed arrow also points from Block 3 to Block 4.

```
[0x660] ;[gb]
;-- main:
fcn main 54
main ();
; var int local_14h @ rbp-0x14
; var int local_4h @ rbp-0x4
; CALL XREF from 0x000006a6 (sym.function1)
; DATA XREF from 0x0000054d (entry0)
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [local_14h], edi
mov dword [local_4h], 0
mov dword [local_4h], 0
jmp 0x689;[ga]

v
|
0x689 ;[ga]
; JMP XREF from 0x00000679 (main)
; [0x4:4]=0x10102
cmp dword [local_4h], 4
jle 0x67b;[gd]

t f
|
0x67b ;[gd]
; JMP XREF from 0x0000068d (main)
mov eax, dword [local_4h]
mov edi, eax
call sym.function1;[gc]
add dword [local_4h], 1

0x68f ;[ge]
mov eax, 0
leave
ret
```

1

2

3

4

1

Blocks and call labelling

CERT-Conix tools: Machoke and BTG

Machoke: algorithm

The screenshot shows the Machoke debugger interface with assembly code. The code is divided into four numbered blocks:

- Block 1: [0x660] ;[gb]
;-- main:
(fcn) main 54
main ();
; var int local_14h @ rbp-0x14
; var int local_4h @ rbp-0x4
; CALL XREF from 0x000006a6 (sym.functionl)
; DATA XREF from 0x0000054d (entry0)
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [local_14h], edi
mov dword [local_4h], 0
mov dword [local_4h], 0
jmp 0x689;[ge]
- Block 2: 0x689 ;[ga]
; JMP XREF from 0x00000679 (main)
; [0x4:4]=0x10102
cmp dword [local_4h], 4
jle 0x67b;[gd]
- Block 3: 0x67b ;[gd]
; JMP XREF from 0x0000068d (main)
mov eax, dword [local_4h]
mov edi, eax
call sym.functionl;[gc]
add dword [local_4h], 1
- Block 4: 0x68f ;[ge]
mov eax, 0
leave
ret

- ➊ Blocks and call labelling
- ➋ Translate to text:
1:2;

CERT-Conix tools: Machoke and BTG

Machoke: algorithm

The screenshot shows assembly code with four numbered regions (1, 2, 3, 4) indicating different blocks:

- Region 1:** [0x660] ;[gb]
;-- main:
(fcn) main 54
main ();
; var int local_14h @ rbp-0x14
; var int local_4h @ rbp-0x4
; CALL XREF from 0x000006a6 (sym.functionl)
; DATA XREF from 0x0000054d (entry0)
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [local_14h], edi
mov dword [local_4h], 0
mov dword [local_4h], 0
jmp 0x689;[ga]
- Region 2:** 0x689 ;[ga]
; JMP XREF from 0x00000679 (main)
; [0x4:4]=0x10102
cmp dword [local_4h], 4
jle 0x67b;[gd]
- Region 3:** 0x67b ;[gd]
; JMP XREF from 0x0000068d (main)
mov eax, dword [local_4h]
mov edi, eax
call sym.functionl;[gc]
add dword [local_4h], 1
- Region 4:** 0x68f ;[ge]
mov eax, 0
leave
ret

- 1 Blocks and call labelling
- 2 Translate to text:
1:2;2:3,4;

CERT-Conix tools: Machoke and BTG

Machoke: algorithm

The screenshot shows assembly code from the Machoke debugger. The code is organized into four numbered blocks:

- Block 1:** The first few instructions set up local variables and push the current stack pointer (rbp) onto the stack. Red numbers '1' and 'v' are overlaid on this block.
- Block 2:** This block contains a conditional jump (jle) based on the value of a local variable. Red number '2' is overlaid on this block.
- Block 3:** This block contains a call instruction to a symbol named 'sym.function1'. A red circle highlights this call instruction. Red number '3' is overlaid on this block.
- Block 4:** This block contains the function prologue (mov eax, 0; leave; ret) and the function epilogue (add dword [local_4h], 1). Red number '4' is overlaid on this block.

The assembly code itself includes comments like ;[gb] and ; CALL XREF from 0x000006a6 (sym.function1).

```
[0x660] ;[gb]
;-- main:
fcn main 54
main ();
; var int local_14h @ rbp-0x14
; var int local_4h @ rbp-0x4
;     ; CALL XREF from 0x000006a6 (sym.function1)
;     ; DATA XREF from 0x0000054d (entry0)
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [local_14h], edi
mov dword [local_4h], 0
mov dword [local_4h], 0
jmp 0x689;[ga] 1

v

0x689 ;[ga]
; JMP XREF from 0x00000679 (main)
; [0x4:4]=0x10102
cmp dword [local_4h], 4 2
jle 0x67b;[gd]

t f

0x67b ;[gd]
; JMP XREF from 0x0000068d (main)
mov eax, dword [local_4h]
mov edi, eax
call sym.function1;[gc] 3
add dword [local_4h], 1

0x68f ;[ge]
mov eax, 0
leave
ret 4
```

- ➊ Blocks and call labelling
- ➋ Translate to text:
1:2;2:2;3,4;3:c,2;

CERT-Conix tools: Machoke and BTG

Machoke: algorithm

The screenshot shows the Machoke debugger interface with assembly code. The code is divided into four numbered blocks:

- Block 1:** [0x660] ;[gb]
;-- main:
(fcn) main 54
main ();
; var int local_14h @ rbp-0x14
; var int local_4h @ rbp-0x4
; ; CALL XREF from 0x000006a6 (sym.functionl)
; ; DATA XREF from 0x0000054d (entry0)
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [local_14h], edi
mov dword [local_4h], 0
mov dword [local_4h], 0
jmp 0x689;[ga]
- Block 2:** 0x689 ;[ga]
; JMP XREF from 0x00000679 (main)
; [0x4:4]=0x10102
cmp dword [local_4h], 4
jle 0x67b;[gd]
- Block 3:** 0x67b ;[gd]
; JMP XREF from 0x0000068d (main)
mov eax, dword [local_4h]
mov edi, eax
call sym.functionl;[gc]
add dword [local_4h], 1
- Block 4:** 0x68f ;[ge]
mov eax, 0
leave
ret

Annotations include:

- A red circle around the 'call' instruction in Block 3.
- Red numbers 1 through 4 next to each block.
- Red letters 'v' and 't f' above the first two blocks.

- 1 Blocks and call labelling
- 2 Translate to text:
1:2;2:2;3,4;3:c,2;4::

CERT-Conix tools: Machoke and BTG

Machoke: algorithm

The screenshot shows the Machoke debugger interface with assembly code. The code is divided into four numbered blocks:

- Block 1:** [0x660] ;[gb]
;-- main:
(fcn) main 54
main ();
; var int local_14h @ rbp-0x14
; var int local_4h @ rbp-0x4
; ; CALL XREF from 0x000006a6 (sym.functionl)
; ; DATA XREF from 0x0000054d (entry0)
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [local_14h], edi
mov dword [local_4h], 0
mov dword [local_4h], 0
jmp 0x689;[ge]
- Block 2:** 0x689 ;[ga]
; JMP XREF from 0x00000679 (main)
; [0x4:4]=0x10102
cmp dword [local_4h], 4
jle 0x67b;[gd]
- Block 3:** 0x67b ;[gd]
; JMP XREF from 0x0000068d (main)
mov eax, dword [local_4h]
mov edi, eax
call sym.functionl;[gc]
add dword [local_4h], 1
- Block 4:** 0x68f ;[ge]
mov eax, 0
leave
ret

- 1 Blocks and call labelling
- 2 Translate to text:
1:2;2:3,4;3:c,2;4::
- 3 Murmurhash3: e38a5cbb

CERT-Conix tools: Machoke and BTG

Machoke: algorithm

The screenshot shows the Machoke tool interface with assembly code for the 'main' function. The code includes variable declarations, function calls, and a loop structure. Four numbered callouts point to specific parts of the assembly:

- 1**: Points to the beginning of the main function, showing the prologue code: push rbp, mov rbp, rsp, sub rsp, mov dword [local_14h], edi, mov dword [local_4h], 0, mov dword [local_4h], 0, and jmp 0x689;[ga].
- 2**: Points to the start of a conditional jump block: cmp dword [local_4h], 4, jle 0x67b;[gd].
- 3**: Points to the body of the loop, which contains a call to 'sym.function1'. This call is circled in red.
- 4**: Points to the end of the loop, showing the epilogue code: mov eax, 0, leave, ret.

```
[0x660] ;[gb]
    ;-- main:
fcn main 54
main ();
; var int local_14h @ rbp-0x14
; var int local_4h @ rbp-0x4
;     ; CALL XREF from 0x000006a6 (sym.function1)
;     ; DATA XREF from 0x0000054d (entry0)
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [local_14h], edi
mov dword [local_4h], 0
mov dword [local_4h], 0
jmp 0x689;[ga]

v
|
|_
0x689 ;[ga]
; JMP XREF from 0x00000679 (main)
; [0x4:4]=0x10102
cmp dword [local_4h], 4
jle 0x67b;[gd]

t f
|
|_
0x67b ;[gd]
; JMP XREF from 0x0000068d (main)
mov eax, dword [local_4h]
mov edi, eax
call sym.function1;[gc]
add dword [local_4h], 1

0x68f ;[ge]
mov eax, 0
leave
ret
```

- 1 Blocks and call labelling
- 2 Translate to text:
1:2;2:3,4;3:c,2;4::
- 3 Murmurhash3: e38a5cbb
- 4 Repeat for each function in sample, concatenate hashes

CERT-Conix tools: Machoke and BTG

Machoke: r2 commands used

- ① aa
- ② ilj
- ③ aflj
- ④ agj

CERT-Conix tools: Machoke and BTG

Machoke: Results

Analysis on a (small) collection of 21915 samples:

- ① 21915 unique MD5/SHA256 (as expected)

CERT-Conix tools: Machoke and BTG

Machoke: Results

Analysis on a (small) collection of 21915 samples:

- ① 21915 unique MD5/SHA256 (as expected)
- ② 10691 unique ssdeep

CERT-Conix tools: Machoke and BTG

Machoke: Results

Analysis on a (small) collection of 21915 samples:

- ① 21915 unique MD5/SHA256 (as expected)
- ② 10691 unique ssdeep
- ③ Only 4674 unique machoke hashes

CERT-Conix tools: Machoke and BTG

Machoke: demonstration

Demonstration

BTG: OSINT tool

CERT-Conix tools: Machoke and BTG

BTG: typical qualification workflow

The SOC/CERT analyst stumbles upon an suspicious domain/IP
(our observables) and then :

CERT-Conix tools: Machoke and BTG

BTG: typical qualification workflow

The SOC/CERT analyst stumbles upon an suspicious domain/IP (our observables) and then :

- ① Go on VT and search for the observables

CERT-Conix tools: Machoke and BTG

BTG: typical qualification workflow

The SOC/CERT analyst stumbles upon an suspicious domain/IP (our observables) and then :

- ① Go on VT and search for the observables
- ② Go on MalwareShare and search for the observables

CERT-Conix tools: Machoke and BTG

BTG: typical qualification workflow

The SOC/CERT analyst stumbles upon an suspicious domain/IP (our observables) and then :

- ① Go on VT and search for the observables
- ② Go on MalwareShare and search for the observables
- ③ Go on MISP and search for the observables
- ④ Go on Cuckoo and search for the observables

CERT-Conix tools: Machoke and BTG

BTG: typical qualification workflow

The SOC/CERT analyst stumbles upon an suspicious domain/IP (our observables) and then :

- ① Go on VT and search for the observables
- ② Go on MalwareShare and search for the observables
- ③ Go on MISP and search for the observables
- ④ Go on Cuckoo and search for the observables
- ⑤ Go on PassiveTotal and search for the observables
- ⑥ Go on OTX and search for the observables
- ⑦ Go on some malware trackers and search for the observables

CERT-Conix tools: Machoke and BTG

BTG: typical qualification workflow

The SOC/CERT analyst stumbles upon an suspicious domain/IP (our observables) and then :

- ① Go on VT and search for the observables
- ② Go on MalwareShare and search for the observables
- ③ Go on MISP and search for the observables
- ④ Go on Cuckoo and search for the observables
- ⑤ Go on PassiveTotal and search for the observables
- ⑥ Go on OTX and search for the observables
- ⑦ Go on some malware trackers and search for the observables

So many sources of informations.

CERT-Conix tools: Machoke and BTG

BTG: purposes

Help the analyst to get to the websites that contains info/intel about this observable quickly.

CERT-Conix tools: Machoke and BTG

BTG: principles

- ① Ergonomy (less output is better)
- ② Give the right informations to the analyst

CERT-Conix tools: Machoke and BTG

BTG: invocation

- ① python3 BTG.py [your observable]

CERT-Conix tools: Machoke and BTG

BTG: invocation

- ① python3 BTG.py [your observable]
- ② python3 BTG.py [your observables]

CERT-Conix tools: Machoke and BTG

BTG: invocation

- ① python3 BTG.py [your observable]
- ② python3 BTG.py [your observables]
- ③ python3 BTG.py [name of file containing your observables]

CERT-Conix tools: Machoke and BTG

BTG: output formating

[OSINT Source][Status]{Observable} Link to the ressource

CERT-Conix tools: Machoke and BTG

BTG: output formating

[OSINT Source][Status]{Observable} Link to the ressource

Example :

[malwareteks][FOUND]{m|314.com} <http://hosts-file.malwareteks.com/hosts.txt>

CERT-Conix tools: Machoke and BTG

BTG

- ① python3
- ② 25 modules for 25 OSINT services
- ③ actively used and maintained @ Conix
- ④ observables handled: URL, MD5, SHA1, SHA256, SHA512, IPv4, IPv6, domain

CERT-Conix tools: Machoke and BTG

BTG: demonstration

Demonstration

Github Conix-security :

- ① <https://github.com/conix-security/machoke>
- ② <https://github.com/conix-security/BTG>

Q/A

—

Robin Marsollier

robin.marsollier@conix.fr

CERT-Conix