Recognition of binary patterns by Morphological Analysis

Aurélien Thierry (aurelien.thierry@inria.fr), Guillaume Bonfante, Joan Calvet, Jean-Yves Marion, Fabrice Sabatier

Recon 2012-06-16

Introduction

- Binary analysis

```
.text:00452C1B
.text:00452C1B sub_452C1B      proc near               ; CODE XREF: sub
.text:00452C1B                                         ; sub_45317E+16↓
.text:00452C1B
.text:00452C1B var_4           = dword ptr -4
.text:00452C1B arg_0           = dword ptr  4
.text:00452C1B arg_4           = dword ptr  8
.text:00452C1B arg_8           = dword ptr  0Ch
.text:00452C1B
.text:00452C1B                 push    4
.text:00452C1D                 pop     eax
.text:00452C1E                 call    __alloca_probe
.text:00452C23                 push    ebx
.text:00452C24                 push    ebp
.text:00452C25                 push    esi
.text:00452C26                 push    edi
.text:00452C27                 mov     edi, [esp+14h+arg_0]
.text:00452C2B                 test    edi, edi
.text:00452C2D                 jz      loc_453175
.text:00452C33                 mov     esi, [esp+14h+arg_8]
.text:00452C37                 test    esi, esi
.text:00452C39                 jz      loc_453175
.text:00452C3F                 mov     eax, [esp+14h+arg_4]
.text:00452C43                 mov     ecx, 80h
.text:00452C48                 cmp     eax, ecx
.text:00452C4A                 jz      short loc_452C66
.text:00452C4C                 cmp     eax, 0C0h
.text:00452C51                 jz      short loc_452C62
.text:00452C53                 cmp     eax, 100h
.text:00452C58                 jz      short loc_452C62
.text:00452C5A                 push    0FFFFFFFEh
.text:00452C5C                 pop     eax
.text:00452C5D                 jmp     loc_453178
```

Introduction

- Binary analysis

- Identify librairies that do not need to be reversed

Introduction

- Binary analysis

- Identify librairies that do not need to be reversed

Our approach :

- Control flow graph comparison
- Import results in IDA

Waledac malware and OpenSSL

- Spamming botnet
- Use of cryptography for communication : RSA and AES

Waledac malware and OpenSSL

- Spamming botnet
- Use of cryptography for communication : RSA and AES

```
aurelien:~/R$ strings Waledac\ v48\ unpacked.int | grep OpenSSL
EC part of OpenSSL 0.9.8e 23 Feb 2007
ECDSA part of OpenSSL 0.9.8e 23 Feb 2007
```

- OpenSSL 0.9.8e (Feb 2007) used for cryptography

Waledac malware and OpenSSL

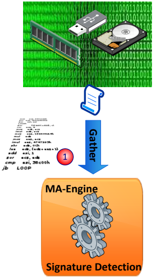
- Spamming botnet
- Use of cryptography for communication : RSA and AES

```
aurelien:~/R$ strings Waledac\ v48\ unpacked.int | grep OpenSSL
EC part of OpenSSL 0.9.8e 23 Feb 2007
ECDSA part of OpenSSL 0.9.8e 23 Feb 2007
```

- OpenSSL 0.9.8e (Feb 2007) used for cryptography
- Which functions are specifically used ?

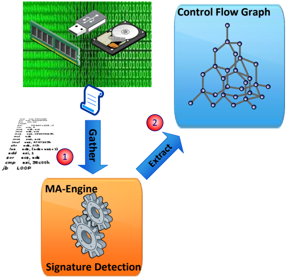
Morphological Analysis : Learning a file

Step 1 : Learn



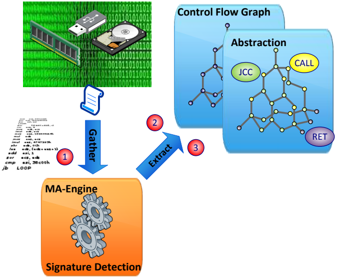
Morphological Analysis : Learning a file

Step 1 : Learn



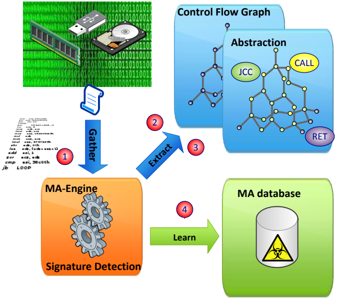
Morphological Analysis : Learning a file

Step 1 : Learn



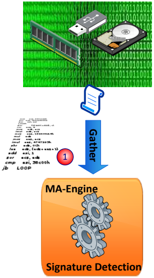
Morphological Analysis : Learning a file

Step 1 : Learn



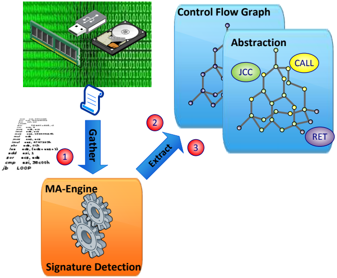
Morphological Analysis : Scanning a file

Step 2 : Scan



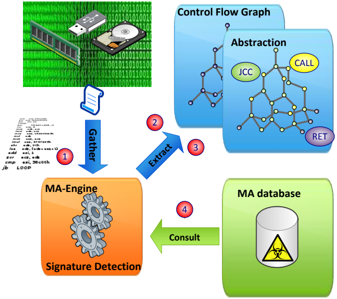
Morphological Analysis : Scanning a file

Step 2 : Scan



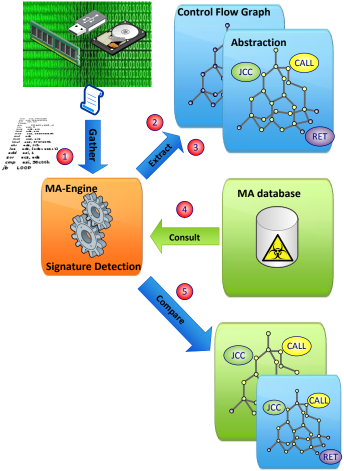
Morphological Analysis : Scanning a file

Step 2 : Scan



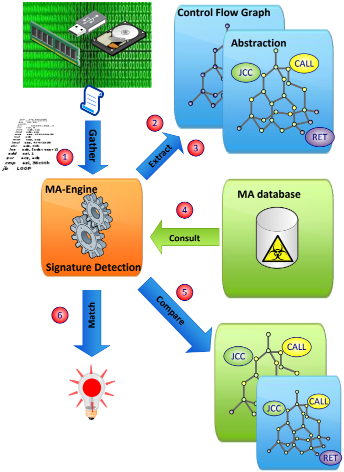
Morphological Analysis : Scanning a file

Step 2 : Scan



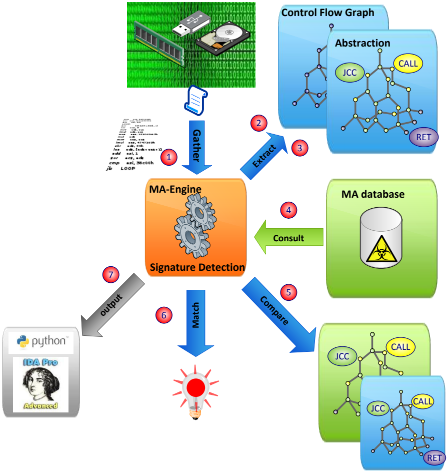
Morphological Analysis : Scanning a file

Step 2 : Scan



Morphological Analysis : Scanning a file

Step 2 : Scan



Control flow graph recovery

Control Flow Graph (CFG) : oriented graph in which nodes are instruction addresses and edges represent all paths that might be traversed during execution

Control flow graph recovery

Control Flow Graph (CFG) : oriented graph in which nodes are instruction addresses and edges represent all paths that might be traversed during execution

ASM code

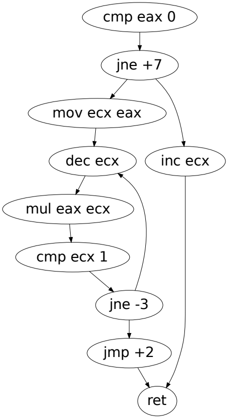
```
cmp eax 0
jne +7
mov ecx eax
dec ecx
mul eax ecx
cmp ecx 1
jne -3
jmp +2
inc ecx
ret
```

Control flow graph recovery

Control Flow Graph (CFG) : oriented graph in which nodes are instruction addresses and edges represent all paths that might be traversed during execution

ASM code

```
cmp eax 0
jne +7
mov ecx eax
dec ecx
mul eax ecx
cmp ecx 1
jne -3
jmp +2
inc ecx
ret
```

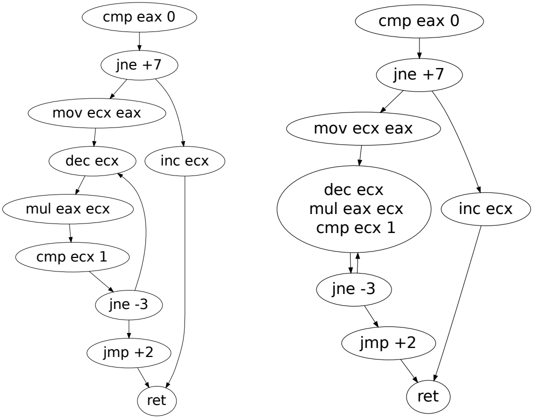


Control flow graph recovery

Control Flow Graph (CFG) : oriented graph in which nodes are instruction addresses and edges represent all paths that might be traversed during execution

ASM code

```
cmp eax 0
jne +7
mov ecx eax
dec ecx
mul eax ecx
cmp ecx 1
jne -3
jmp +2
inc ecx
ret
```



Control flow graph recovery

Extraction of the control flow graph from a binary :

- Static analysis from entrypoints when possible (BeaEngine)
- Dynamic analysis otherwise (Intel's Pintools)

Control flow graph recovery

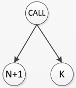
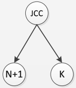
Extraction of the control flow graph from a binary :

- Static analysis from entrypoints when possible (BeaEngine)
- Dynamic analysis otherwise (Intel's Pintools)

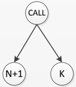
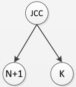
Nodes of the control flow graph :

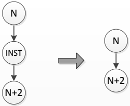
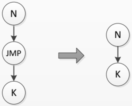
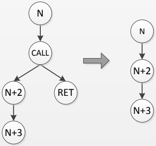
- Sequential instructions do not modify the control flow
- 4 types of instructions have an impact on the CFG (*jmp*, *call*, *jcc*, et *ret*)

Control flow graph construction & reduction

N_{th} instruction	Control flow graph
Sequential instruction	
jmp K	
call K	
jcc K	
ret	

Control flow graph construction & reduction

N_{th} instruction	Control flow graph
Sequential instruction	
jmp K	
call K	
jcc K	
ret	

Remove sequential instructions	
Realign jumps	
Remove false calls	

Reduction of the control flow graph

The CFG is reducted :

- Reduce the size of the graph (fastens the algorithms)
- More abstract form : detect slight changes (junk code insertion, code re-ordering)

Reduction of the control flow graph

The CFG is reducted :

- Reduce the size of the graph (fastens the algorithms)
- More abstract form : detect slight changes (junk code insertion, code re-ordering)

Waledac with static analysis :

- 38236 nodes before reduction
- 14626 nodes after reduction

Reduction on Waledac

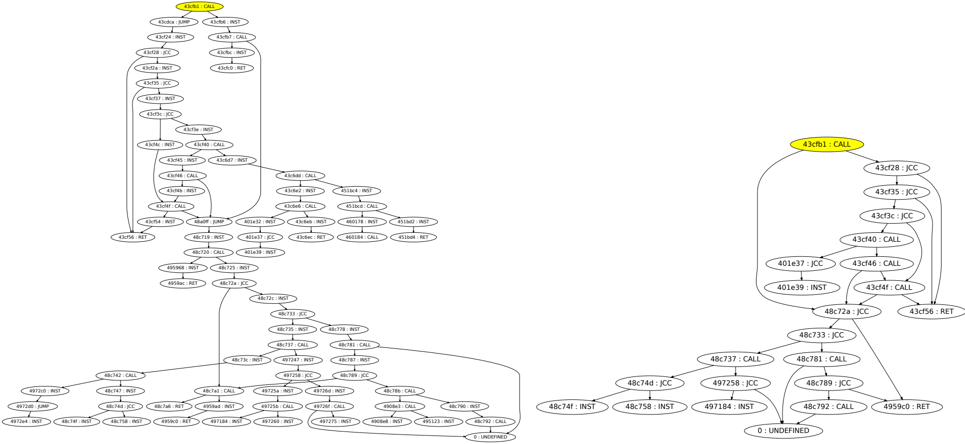
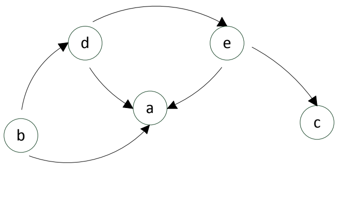


Figure: Part of Waledac without reduction (80 nodes) and with reduction (23 nodes)

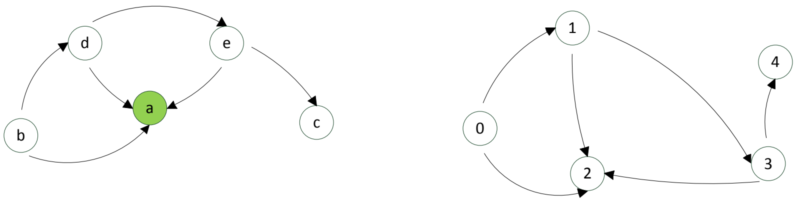
Graph matching

Graph isomorphism detection



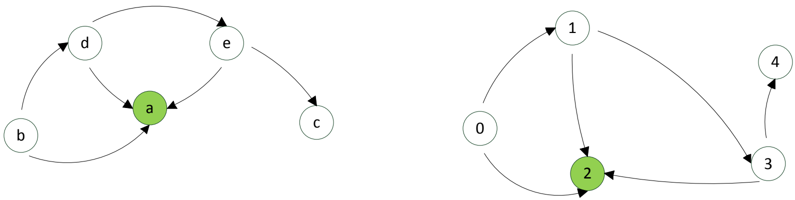
Graph matching

Graph isomorphism detection



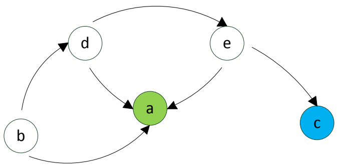
Graph matching

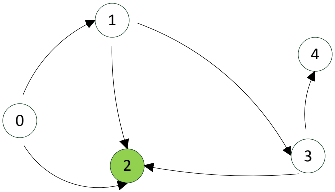
Graph isomorphism detection



Graph matching

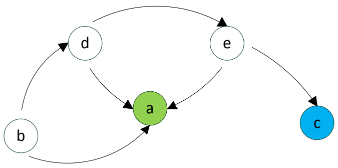
Graph isomorphism detection

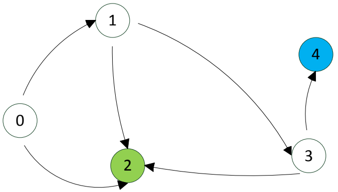




Graph matching

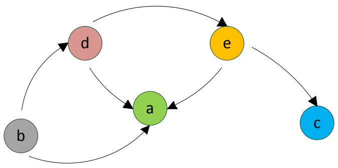
Graph isomorphism detection

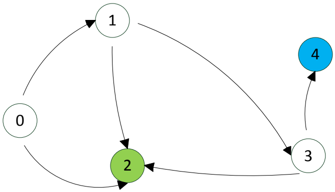




Graph matching

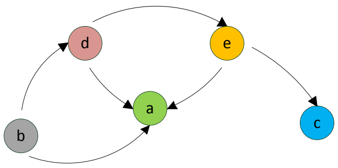
Graph isomorphism detection

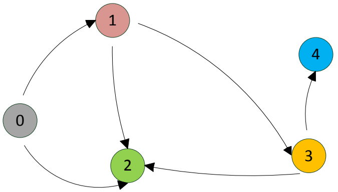




Graph matching

Graph isomorphism detection

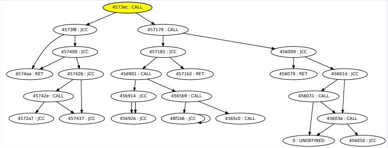




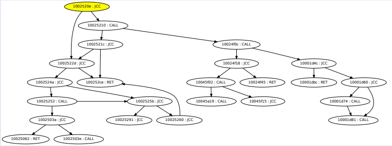
Graph matching : Waledac and OpenSSL

- Entire graphs are not isomorphic

Waledac



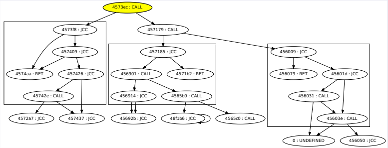
OpenSSL



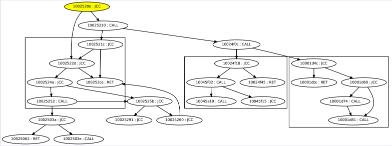
Graph matching : Waledac and OpenSSL

- Entire graphs are not isomorphic
- But some parts (subgraphs) are

Waledac

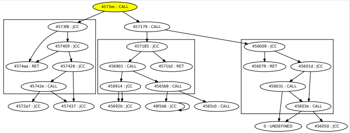


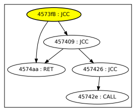
OpenSSL

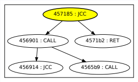


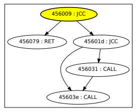
Graph matching : Waledac and OpenSSL

Waledac

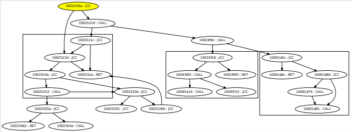


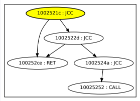


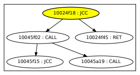


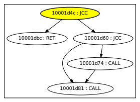


OpenSSL









Subgraphs

- Both graphs are cut into many small subgraphs
- Generated through BFS (Breadth First Search) from each nodes

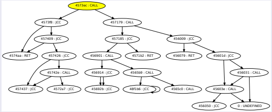
Subgraphs

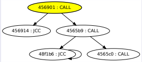
- Both graphs are cut into many small subgraphs
- Generated through BFS (Breadth First Search) from each nodes
- Their size is limited (typically 24 nodes)
- Search graph isomorphisms between subgraphs of both binaries

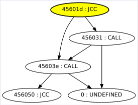
More on subgraphs

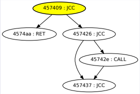
- From one CFG, many subgraphs are generated
- Every reachable node is in many subgraphs

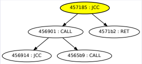
Example on Waledac : 24 nodes to 8 subgraphs of size 5

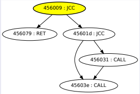


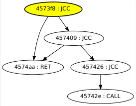


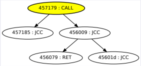


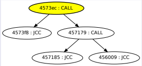












Graph isomorphism problem

- Graph isomorphism has no solution in polynomial time in the general case
- The problem is in NP
- General solutions are slow

Graph isomorphism problem

- Graph isomorphism has no solution in polynomial time in the general case
- The problem is in NP
- General solutions are slow

Property (Simplification)

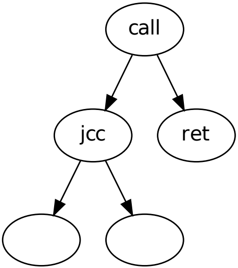
Our subgraphs :

- *Have a root node (from which every other node is reachable)*
- *Each node has at most 2 children (call or jcc)*
- *Children are ordered*

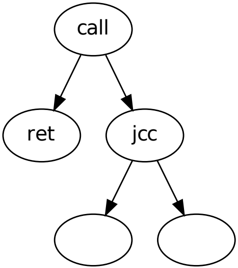
- This problem is in P

Graph isomorphism problem

- Does not exactly resolve the graph isomorphism problem
- But there are fast solutions (polynomial time)



(v) Original graph



(w) Undetected graph

Morphological analysis engine

- Signatures are subgraphs from reducted control flow graphs
- Obtained statically or dynamically
- A database (tree automata) is filled with the signatures

Morphological analysis engine

- Signatures are subgraphs from reducted control flow graphs
- Obtained statically or dynamically
- A database (tree automata) is filled with the signatures
- Learning and scanning is fast (Intel Core i5 CPU M560 2.67GHz)

Operation	Files	Time (s)
Learn	44 binaries ($<$ 2000 nodes)	1.2s
Scan	44 binaries ($<$ 2000 nodes)	1.1s
Learn	OpenSSL (28313 nodes)	12s
Scan	Waledac (14626 nodes)	2.0s

Compare Waledac and OpenSSL

- Waledac uses OpenSSL 0.9.8e (Feb 2007)
- OpenSSL learnt with reduction

Compare Waledac and OpenSSL

- Waledac uses OpenSSL 0.9.8e (Feb 2007)
- OpenSSL learnt with reduction
- One DLL is matched (libeay.dll)

OpenSSL version	Comment	Results (common subgraphs)
0.9.8x	Released in May 2012	53
0.9.8e	Compiled for performance (/0x /02)	53
0.9.8e	Compiled for file size (/01)	1264

Compare Waledac and OpenSSL

- Compile OpenSSL 0.9.8e with option /O1 (size optimization)
- 1264 common subgraphs between one of the DLLs (libeay.dll) and Waledac !!

- We want to know which functions are matched
- We will compare the matched code of OpenSSL and Waledac

Code and nodes

- The larger the matched subgraphs are, the more accurate the matching

Code and nodes

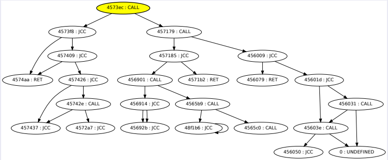
- The larger the matched subgraphs are, the more accurate the matching
- Learns and scans with increasing number of nodes from 24
- Associate nodes that are in the largest subgraphs

Code and nodes

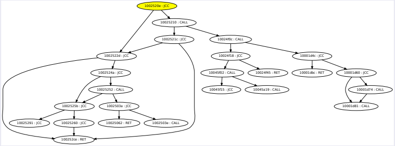
- The larger the matched subgraphs are, the more accurate the matching
- Learns and scans with increasing number of nodes from 24
- Associate nodes that are in the largest subgraphs
- Outputs matched nodes for each size for IDA

Code and nodes

Waledac



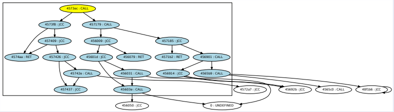
OpenSSL



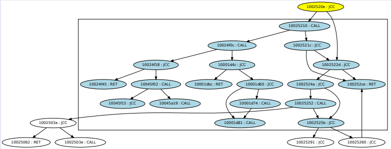
Code and nodes

- Greatest subgraph found has 18 nodes
- Corresponding nodes in matched subgraphs are associated
- Then associate free nodes on matching subgraphs of lesser size

Waledac



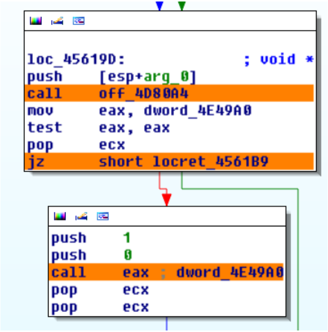
OpenSSL



IDA plugin

With both binaries opened in IDA

- Imports the list of matched nodes
- Marks them in IDA
- Provides browsing through corresponding nodes in both instances



Waledac / OpenSSL : common subroutines

libeay32-098e.dll	subroutine	Waledac48.int	subroutine
10002CDE	CRYPTO_new_ex_data	00455B5C	sub_455B55
10002CE0	CRYPTO_new_ex_data	00455B5E	sub_455B55
10002D0A	CRYPTO_free_ex_data	00455B72	sub_455B6B
10002D0C	CRYPTO_free_ex_data	00455B74	sub_455B6B
10021F6D	AES_set_encrypt_key	00452C1E	sub_452C1B
10021F7C	AES_set_encrypt_key	00452C2D	sub_452C1B
10021F88	AES_set_encrypt_key	00452C39	sub_452C1B
10021F99	AES_set_encrypt_key	00452C4A	sub_452C1B
10021FA0	AES_set_encrypt_key	00452C51	sub_452C1B
10021FA7	AES_set_encrypt_key	00452C58	sub_452C1B
10021FB3	AES_set_encrypt_key	00452C64	sub_452C1B
10021FD9	AES_set_encrypt_key	00452C8A	sub_452C1B
10021FEF	AES_set_encrypt_key	00452CA0	sub_452C1B
100224D9	AES_set_encrypt_key	0045317D	sub_452C1B
100224E0	AES_set_decrypt_key	00453184	sub_45317E
100224F0	AES_set_decrypt_key	00453194	sub_45317E
100224FA	AES_set_decrypt_key	0045319E	sub_45317E

Figure: Matching nodes are in corresponding subroutines

Waledac / OpenSSL : common subroutines

- AES : AES_set_encrypt_key, AES_set_decrypt_key
- X509 : X509_PUBKEY_set, X509_PUBKEY_get
- RSA / DSA : RSA_free, DSA_size, DSA_new_method
- BN (Big Number lib) : BN_is_prime_fasttest_ex, BN_ctx_new, BN_mod_inverse
- CRYPTO : CRYPTO_lock, CRYPTO_malloc
- Misc OpenSSL routines : UI, encoding...
- ...

Comparing matched code : AES_set_encrypt_key

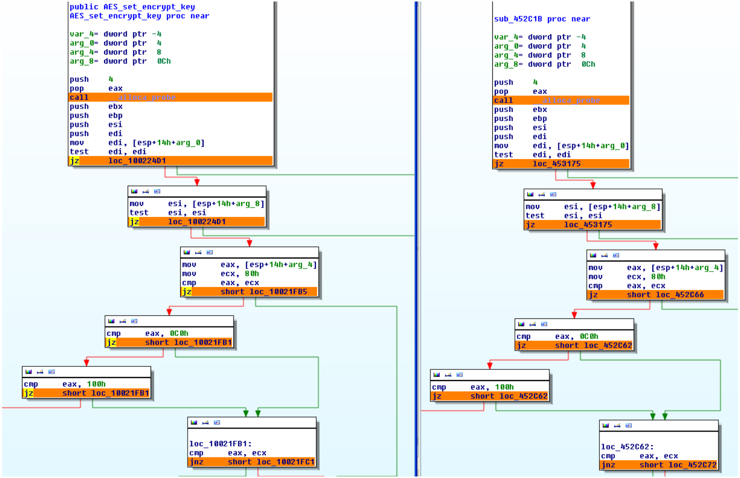


Figure: Matched code between OpenSSL (left) and Waledac (right)

Comparing code : AES_encrypt

- Not detected

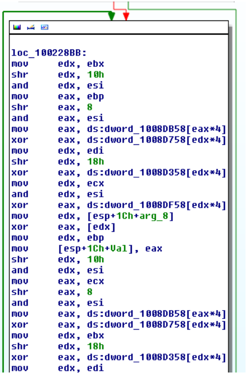


Figure: AES_encrypt subroutine

Comparing code : AES_encrypt

- Not detected
- Control flow graph too small

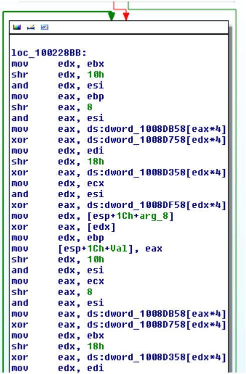


Figure: AES_encrypt subroutine

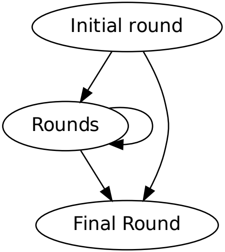


Figure: Simplified AES_encrypt CFG

Waledac / OpenSSL : Findings

- OpenSSL 0.9.8e compiled for being small (option /01)

Waledac / OpenSSL : Findings

- OpenSSL 0.9.8e compiled for being small (option /01)
- Use of AES for symmetric encryption
- X.509 (certificate) handling, use of RSA and/or DSA algorithm
- Calls to primality tests (consistent with asymetric encryption like RSA but not exclusively)

Waledac / OpenSSL : Findings

- OpenSSL 0.9.8e compiled for being small (option /01)

- Use of AES for symmetric encryption
- X.509 (certificate) handling, use of RSA and/or DSA algorithm
- Calls to primality tests (consistent with asymetric encryption like RSA but not exclusively)

- Waledac actually uses X509/RSA and AES encryption
- We were able to find out without actually reversing its code

Duqu and Stuxnet

- Static analysis on their decrypted (and unpacked) main DLLs (maindll.dll for Stuxnet and netp191.pnf for Duqu)

Duqu and Stuxnet

- Static analysis on their decrypted (and unpacked) main DLLs (maindll.dll for Stuxnet and netp191.pnf for Duqu)

First analysis :

- 26.5% of Duqu's subgraphs are common with Stuxnet (846 subgraphs)
- 60.3% of Duqu's nodes are in subgraphs matching with Stuxnet (2215 nodes)
- Duqu and Stuxnet are strongly related

Duqu / Stuxnet : common subroutines

maindll.decrypted.u	subroutine	netp191_Decryptec	subroutine
10042DB0	sub_10042CD2	100136DB	sub_100135FD
10042DC0	sub_10042CD2	100136EB	sub_100135FD
10042DC5	sub_10042CD2	100136F0	sub_100135FD
10042DD3	sub_10042CD2	100136FE	sub_100135FD
10042DE0	sub_10042CD2	1001370B	sub_100135FD
10043116	sub_100430CE	1001353F	sub_100134F7
1004311F	sub_100430CE	10013548	sub_100134F7
10043138	sub_100430CE	10013561	sub_100134F7
1004314D	sub_100430CE	10013576	sub_100134F7
10043155	sub_100430CE	1001357E	sub_100134F7
10043157	sub_100430CE	10013580	sub_100134F7
10043161	sub_100430CE	1001358A	sub_100134F7
1004317B	sub_100430CE	100135A4	sub_100134F7
10043183	sub_1004317C	100144E3	sub_100144DC
1004318D	sub_1004317C	100144ED	sub_100144DC

Duqu / Stuxnet : subroutine identification

- Some of the common subroutines come from standard librairies (libc...)
- They are documented and should not be manually reversed

Duqu / Stuxnet : subroutine identification

- Some of the common subroutines come from standard librairies (libc...)
- They are documented and should not be manually reversed

- msvcr80.dll : Microsoft Visual C++ Run-Time
- How to identify its code within Duqu / Stuxnet in IDA ?

Duqu / Stuxnet : libc identification

- Learn msvcr80.dll ('libc') and scan Duqu, Stuxnet

Duqu / Stuxnet : libc identification

- Learn msvcr80.dll ('libc') and scan Duqu, Stuxnet

IDA plugin will :

- Mark the nodes common with msvcr80.dll
- Rename the matched subroutines

Duqu / Stuxnet : common subroutines

maindll.decrypted.	subroutine	netp191_Decrypt	subroutine
10042DB0	sub_10042CD2	100136DB	sub_100135FD
10042DC0	sub_10042CD2	100136EB	sub_100135FD
10042DC5	sub_10042CD2	100136F0	sub_100135FD
10042DD3	sub_10042CD2	100136FE	sub_100135FD
10042DE0	sub_10042CD2	1001370B	sub_100135FD
10043116	msvcr80$__beginthreadex	1001353F	msvcr80$__beginthreadex
1004311F	msvcr80$__beginthreadex	10013548	msvcr80$__beginthreadex
10043138	msvcr80$__beginthreadex	10013561	msvcr80$__beginthreadex
1004314D	msvcr80$__beginthreadex	10013576	msvcr80$__beginthreadex
10043155	msvcr80$__beginthreadex	1001357E	msvcr80$__beginthreadex
10043157	msvcr80$__beginthreadex	10013580	msvcr80$__beginthreadex
10043161	msvcr80$__beginthreadex	1001358A	msvcr80$__beginthreadex
1004317B	msvcr80$__beginthreadex	100135A4	msvcr80$__beginthreadex
10043183	msvcr80$_free	100144E3	msvcr80$_free
1004318D	msvcr80$_free	100144ED	msvcr80$_free

Figure: Renamed subroutines matching between Duqu and Stuxnet

Highlighting msvcr80.dll in Stuxnet

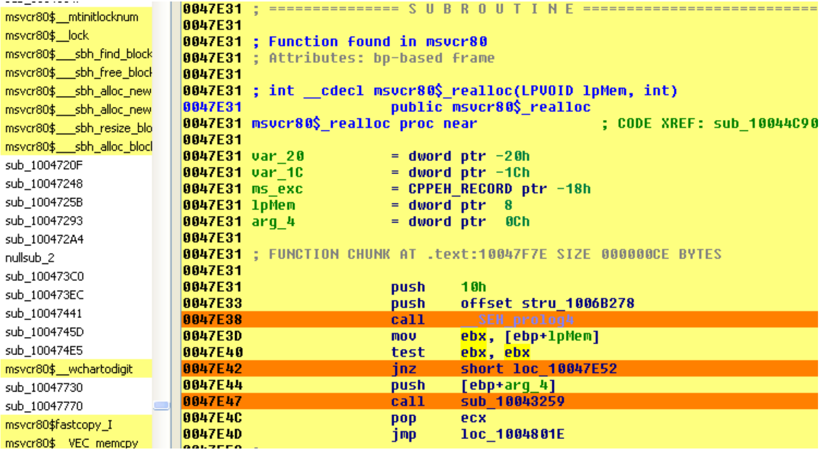


Figure: Colored (yellow) code of msvcr80.dll in Stuxnet, subroutines are renamed

Duqu / Stuxnet : summary

- From the decrypted and unpacked DLLs from Stuxnet, we are able to automatically find code shared with Duqu
- Before reversing, we identify standard (msvcr80.dll) subroutines
- With IDA, we can identify and browse matching subroutines

Conclusion

- Identify used librairies
- Show code similarities
- IDA UI for browsing matched code

Conclusion

- Identify used librairies
- Show code similarities
- IDA UI for browsing matched code

- Thank you
- Any question ? (aurelien.thierry@inria.fr)