Pushdown model generation for binary code

Mizuhito Ogawa@JAIST

with Nguyen Minh Hai, Quan Thanh Tho@HCMUT

Main activity of our group

- Well-Structured Pushdown System (WSPDS)
  - ✓Combine WSTS and PDS (*P*-automata technique)
  - ✓Forward: *Acceleration* for VASS extensions.
  - ✓Backward: *Antichain* for various Timed PDA

- Confluence of non-linear and non-terminating TRSs.
  - ✓Ultimate goal: *non-E-overlapping right-linear* ⇒ *CR*

- Pushdown model generation for binary code

- SMT for nonlinear constrains over reals. (QFNRA)
  - ✓ICP based approximation refinement for inequality.

Why binary code analysis?

- System software : legacy code, commercial protection
  - ✓Compiled from high-level programming language
  - ✓Large
  - ✓Possibly multi-thread

- Malware : distributed by binary only, no copyright☺
  - ✓Control obfuscation
  - ✓Often small
  - ✓Mostly single-thread (though recently there are observed likely multi-threaded; but not confirmed)

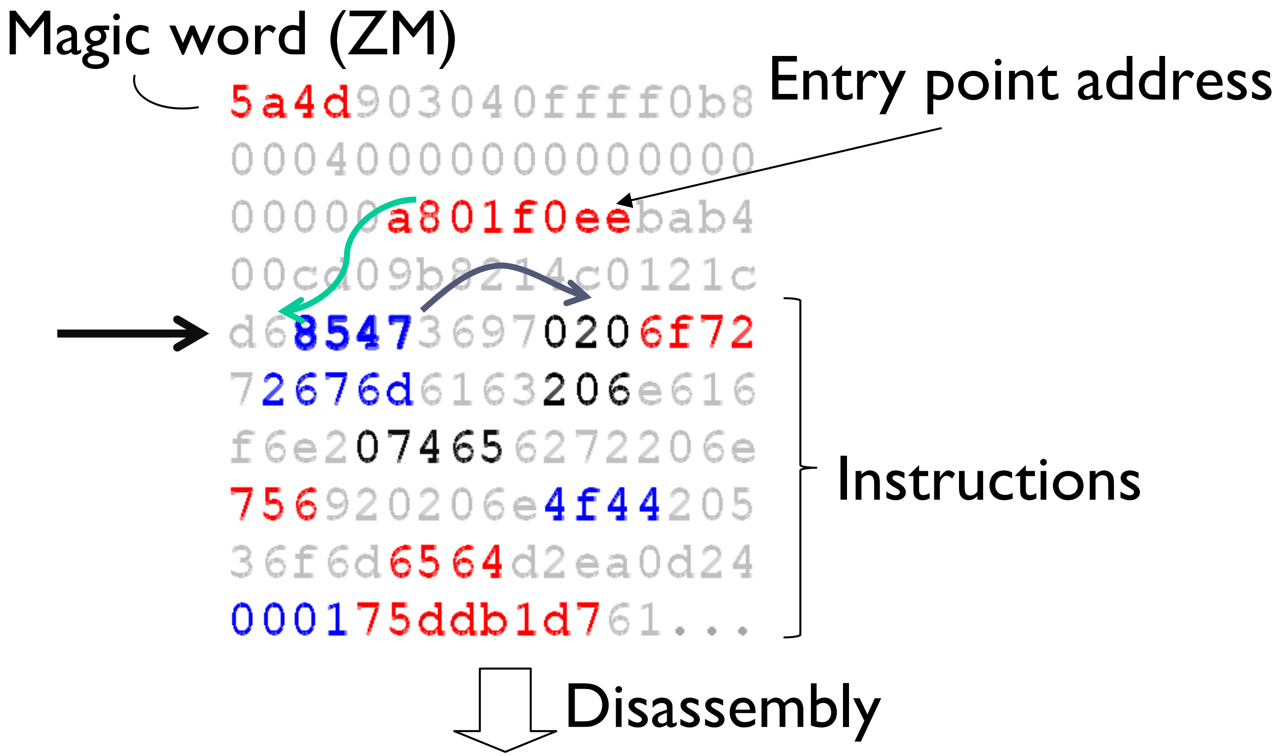
Binary code difficulty

- No clear distinction between *data* and *code*.
  - ✓Code loaded on memory can be modified.
  - ✓Interpretation can be higher-order.

- Dynamic interpretation of CISC (e.g., x86)
  - ✓Instructions have variable length.
  - ✓Memory location can be instruction operands as registers.

Dynamic Interpretation

```
5a4d903040ffff0b8
00040000000000000
00000a801f0eebab4
00cd09b8214c0121c
d6854736970206f72
72676d6163206e616
f6e207465627220 6e
756920206e4f44205
36f6d6564d2ea0d24
000175ddb1d761...
```




```
0x1000: addl $0x2a, %eax
0x1003: cmpl $0x0, %eax
0x1006: jae  0x100f
0x1008: movl $0x5, %ebx
0x100d: jmp  0x1017
0x100f: subl $0x7, %eax
0x1012: movl $0x3, %ebx
0x1017: addl %ebx, %eax
0x1019: ret
```

Today's talk

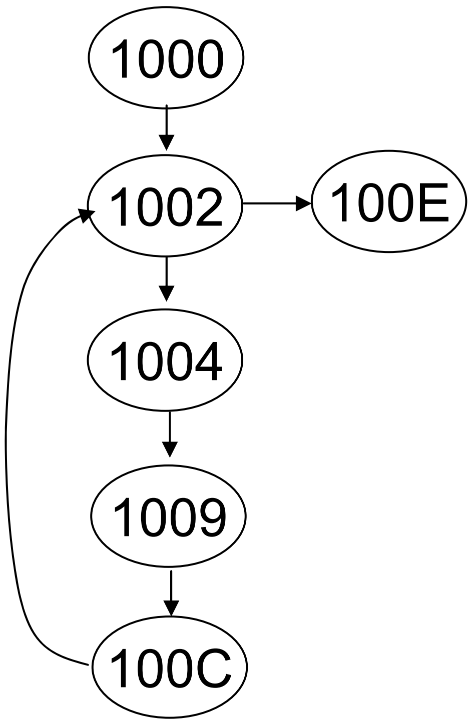
- Binary analysis = model generation + model checking

- Pushdown model generation of binary executable
  - ✓Targeting on obfuscation techniques of malware.
  - ✓Concolic testing (dynamic symbolic execution) to decide control destinations.
  - ✓Will apply *modular weighted pushdown MC*.

Self-modifying binary example

- Next instruction is decided incrementally.
- Instructions can be overwritten.

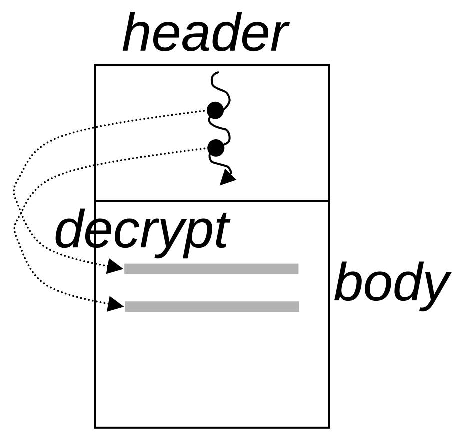
33C0EB0AB803104000C6000A
EBF481FB001000007401C36A
00E8160000000528000000003C
3FFE040E801000000…..



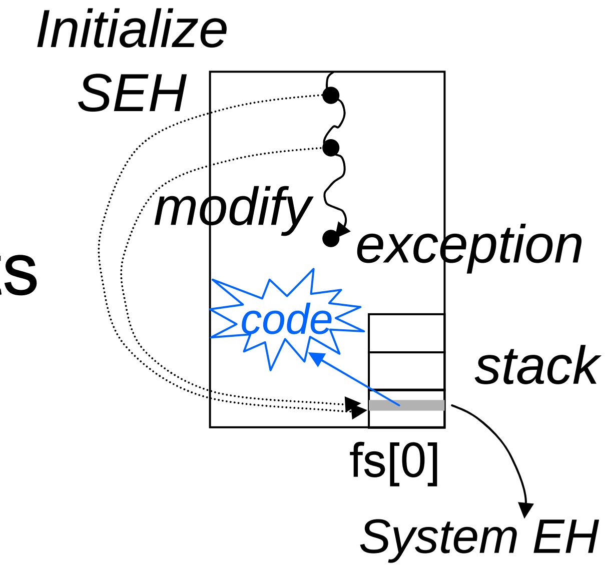
00401000: XOR EAX, EAX
00401002: JMP SHORT 00401004
00401004: MOV EAX, 00401003
00401009: MOV BYTE PTR DS:[EAX], 0A
0040100C JMP SHORT 00401002
00401002: JMP SHORT 0040100E
0040100E: CMP EBX,1000

Control obfuscation techniques of malware

- Indirect jump : *jmp eax*, *RET*
  - ✓Obfuscate destination by arithmetic.
  - ✓Value of *eax* (RET) will be modified.



- Self-modification code (SMC)
  - ✓Modify code loaded on memory
  - ✓*Self-decryption*



- Structural Exception Handler (SEH)
  - ✓Modify fs[0], which originally points to the system exception handler.
  - ✓Intended exception.

Roadmap

- Background : *Obfuscation techniques* and *aim*
- Anti-obfuscation : *Principle ideas*
- BE-PUM (Binary Emulation for Pushdown Model generation) Implementation : *Practical design*
- Experiments : *Statistics, observation*, and *limitation*
- Related and Future work

Fromalize X86 operational semantics

- Memory model
  - ✓Address space M
  - ✓Register, flags

32 bit vector representation

$$\frac{\begin{array}{c} Env_R(eip) \;=\; k, instr(Env_M, k) \;=''\; call\;\; r'', \\ m' = k + |call\ r|, m = Env_R(r), push(S, m') = S' \end{array}}{(Env_F, Env_R, Env_S, Env_M) \rightarrow (Env_F, Env_R[eip \leftarrow m], Env_{S'}, Env_M)} \;[Call]$$

$$\frac{Env_R(eip) = k, instr(Env_M, k) ='' \;ret'', empty(S)}{(Env_F, Env_R, Env_S, Env_M) \rightarrow \perp} \;[Return]$$

$$\frac{Env_R(eip) = k, instr(Env_M, k) ='' \;ret'', \neg empty(S), pop(S) = (S', m)}{(Env_F, Env_R, Env_S, Env_M) \rightarrow (Env_F, Env_R[eip \leftarrow m], Env_{S'}, Env_M)} \;[Return]$$

$$\frac{Env_R(eip) = k, instr(Env_M, k) ='' \;jmp\ r'', Env_R(r) = m}{(Env_F, Env_R, Env_S, Env_M) \rightarrow (Env_F, Env_R[eip \leftarrow m], Env_S, Env_M)} \;[(Indirect)Jump]$$

$$\frac{R(eip) = k, instr(Env_M, k) ='' \;jmp\ m'', M(m) = m'}{(Env_F, Env_R, Env_S, Env_M) \rightarrow (Env_F, Env_R[eip \leftarrow m'], Env_S, Env_M)} \;[Jump]$$

Model generation idea (1) Dynamic interpretation

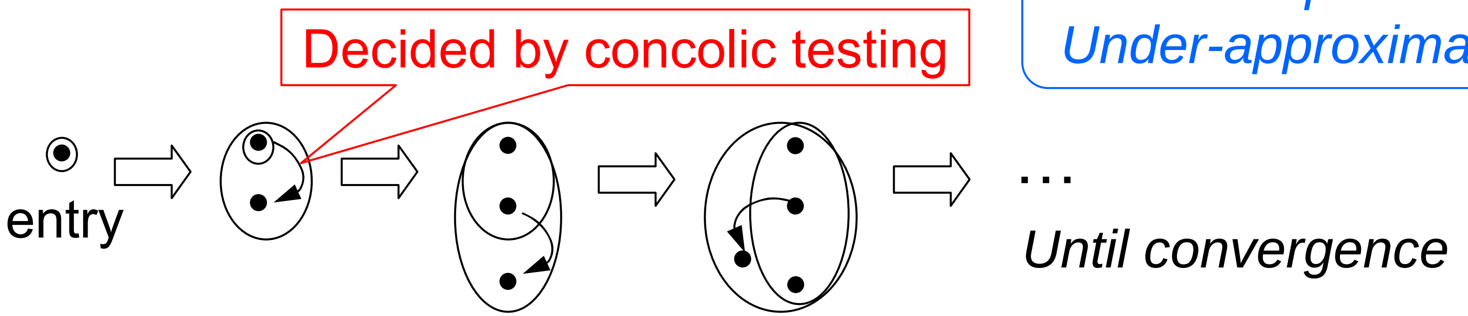
- **Symbolic execution.**

  State = (⟨*binary location, assembly*⟩, *path condition*)

  Transition = (⟨loc, instr⟩, ψ) $\hookrightarrow$ (⟨loc', instr'⟩, ψ') with

  $$\begin{cases} \langle loc', instr' \rangle = next(\langle loc, instr \rangle) \\ \psi' = \psi \vee (SideCond \wedge post(\psi(\langle loc, instr \rangle))) \end{cases}$$

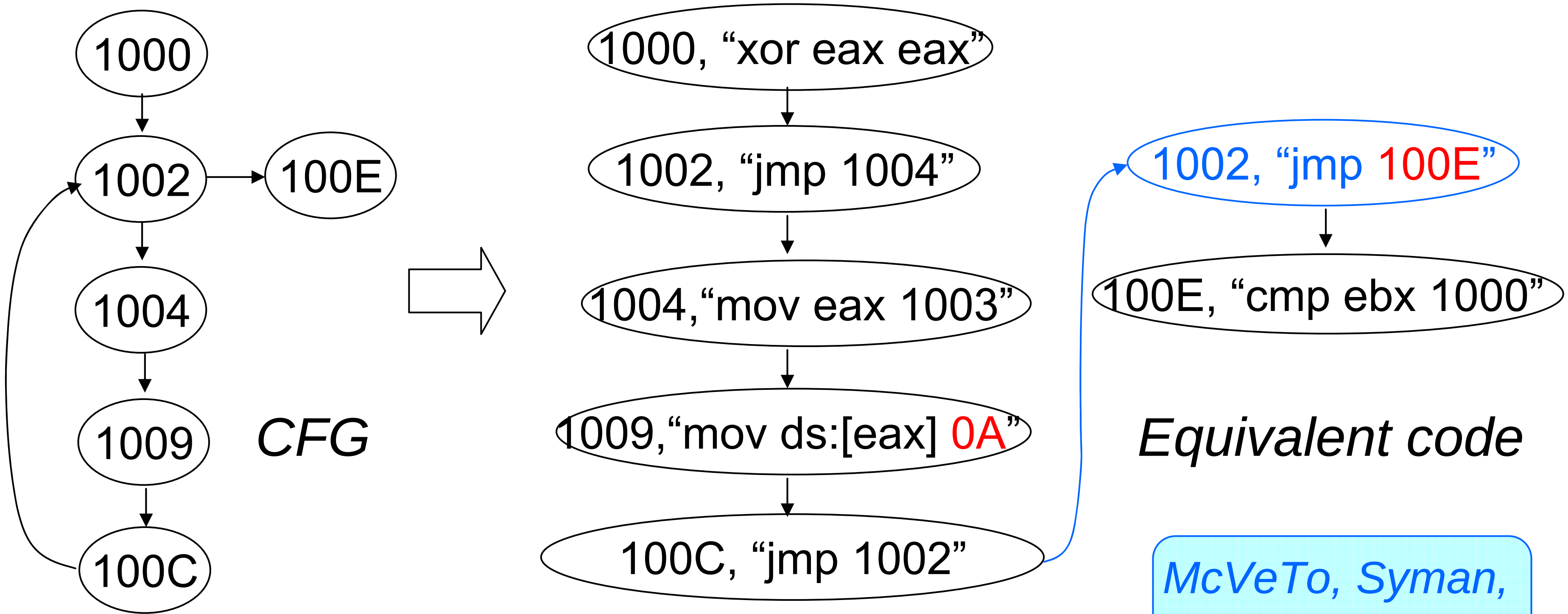
- **On-the-fly**



Without loop invariant, Under-approximation

Model generation ideas (1') SMC

- Generating an equivalent code.
  - ✓*States* = { (*location*, *instruction*, *path condition*) }
  - ✓*Model node* = { (*location*, *instruction*) }



McVeTo, Syman, At LORIA

Model generation idea (2) SEH, RET obfuscation

- Pushdown model
  - ✓Handling exception requires context sensitivity
  - ✓RET address modification is naturally modeled.

$$\frac{\langle p, \gamma w \rangle \hookrightarrow \langle p', \gamma' w \rangle}{(p, \gamma \rightarrow p', \gamma') \in \Delta}\ \mathit{inter} \qquad \frac{\langle p, \gamma w \rangle \hookrightarrow \langle p', \alpha\beta w \rangle}{(p, \gamma \rightarrow p', \alpha\beta) \in \Delta}\ \mathit{push} \qquad \frac{\langle p, \gamma w \rangle \hookrightarrow \langle p', w \rangle}{(p, \gamma \rightarrow p', \epsilon) \in \Delta}\ \mathit{pop}$$

RET address modification

- Assumption
  - ✓Single thread.
  - ✓Stack modification occurs only at the top frame.

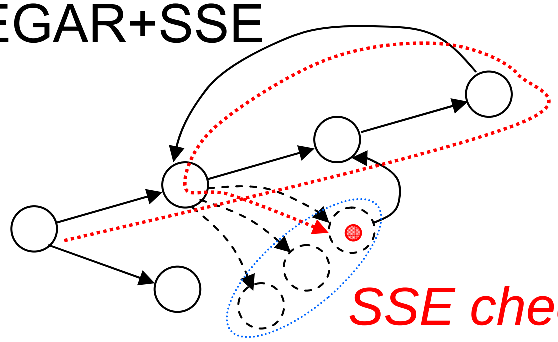
- Pushdown model checkers: Weighted PDS, WPDS+

Model generation ideas (3) Indirect Jumps

- Indirect jump
  - ✓Encapsulate the destination by indirect pointers.
  - ✓Often the destination is overwritten/modified.

- Static vs dynamic (hybrid)
  - ✓Static : CEGAR + Static symbolic execution
  - ✓Dynamic (hybrid) : Dynamic symbolic execution

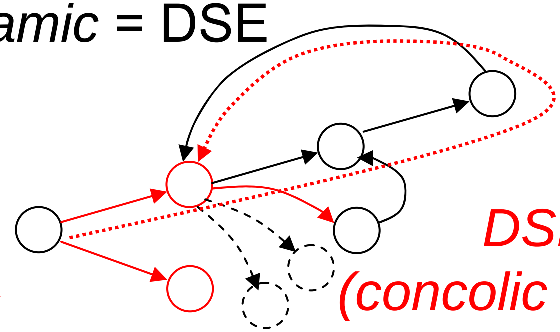
Static = CEGAR+SSE



SSE checks feasibility

Over-approximation by static analysis

Dynamic = DSE



DSE
(concolic testing)

May miss (under-approximation)

Choice of binary emulation

- Full Windows32 emulation (e.g., Syman)
  - ✓ *State = memory snapshot*
  - ✓ *Pros*. Can handle API in the emulation
  - ✓ *Cons*. Models are too detailed (easily explode). Symbolic execution would be not possible

- Single user process emulation
  - ✓ *State = (binary location, corresponding assembly)*
  - ✓ *Pros*. Control structure abstraction nearer to CFG
  - ✓ *Cons*. System call (API) is treated as a stub.

- Dataflow will be re-computed by weighted pushdow model checking.

Roadmap

- Background : *Obfuscation techniques* and *aim*
- Anti-obfuscation : *Principle ideas*
- BE-PUM (Binary Emulation for Pushdown Model generation) Implementation : *Practical design*
- Experiments : *Statistics, observation*, and *limitation*
- Related and Future work

Engineering difficulty

- Huge numbers of x86 instructions & Windows API
  - ✓ >1000 x86 instructions : Complex semantics
  - ✓ >4000 Windows APIs : Not all are specified
    - –Virus probes “sand-box” by unspecified API call.

- Choice of support by statistics (by Jakstab)
  - ✓Most frequent 64 x86 instructions as SE
  - ✓Most frequent 45 APIs as stub

4362 classified malwares from *VX Heaven*

- *VX Heaven*: Malware classification

Kind	Virus	Backdoor	Email	P2P	Constr.	Exploit	IRC	VirTool	Net	Worm	IM	Others
Number	2079	1079	359	105	86	85	73	68	66	64	59	208

- Instruction Occurrences

Instruction	push	mov	jmp	dec	pop	call	add	inc	xor	sub	je	jne	cmp
Occurrences	2974	2756	2590	2547	2469	2282	2155	2089	2037	1771	1707	1618	1607

Instruction	or	jb	jae	lea	and	jbe	ja	ret	imul	shl	xchg	jo	ror
Occurrences	1460	1418	1313	1163	1151	1042	953	894	851	709	660	612	529

- Coverage in *VX Heavens (detected by Jakstab)*

Instructions	200	190	180	170	160	150	140	130	120	110	100	75	50
Covered Malware	4149	4118	4070	4007	3881	3755	3570	3383	3233	3079	2881	2274	1652
Covarage (%)	95.12	94.41	93.31	91.86	88.97	86.08	81.84	77.56	74.12	70.59	66.05	52.13	37.87

Selected 64 x86 instructions & 45 Windows APIs

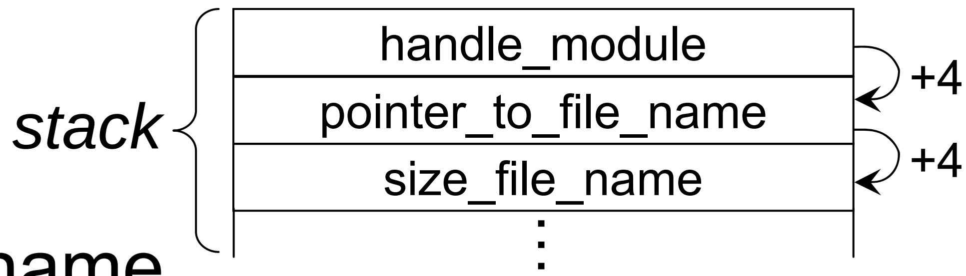
Arithmetic			**Logic**	**Call**	**Conditional Jump**					**Jump**	**Move**	**Return**	**Control**
add	*sub*	adc	*and*	call	*ja*	jae	jna	jnae	loop	*jmp*	*mov*	ret	*cmp*
div	mul	imul	*or*		jb	jbe	jnb	jnbe			int		push
shl	*shr*	sal	xor		jc	*je*	jnc	*jne*			*lea*		pop
inc	*dec*	clc			jg	*jge*	*jng*	jnge			xchg		nop
rol	ror	cld			jl	*jle*	jnl	jnle					test
lods	stos	rep			jp	jo	jnp	jno					cmps
scas					js	jz	jns	jnz					

- **kernel32.dll** ExitProcess, GetProcAddress, LoadLibrary, VirtualAlloc, VirtualFree, CloseHandle, GetModuleHandle, CreateFile, SetFilePointer, GetCommandLine, GetModuleFileName, CopyFile, FindClose, FindFirstFile, GetWindowsDirectory, SetFileAttributes, DeleteFile, FindNextFile, GetLastError, HeapFree, GetCurrentDirectory, GetSystemDirectory, GetSystemTime, GetVersion, lstrcpy, MapViewOfFile, ReadFile, UnmapViewOfFile, WriteFile, CreateFileMapping, CreateProcess, GetFileAttributes, SetEndOfFile, HeapCreate, GetStartupInfo, lstrcat, lstrcmp, lstrlen, MoveFile, HeapDestroy, SetCurrentDirectoryA.
- **user32.dll** MessageBox, SendMessage, FindWindow, PostMessage.

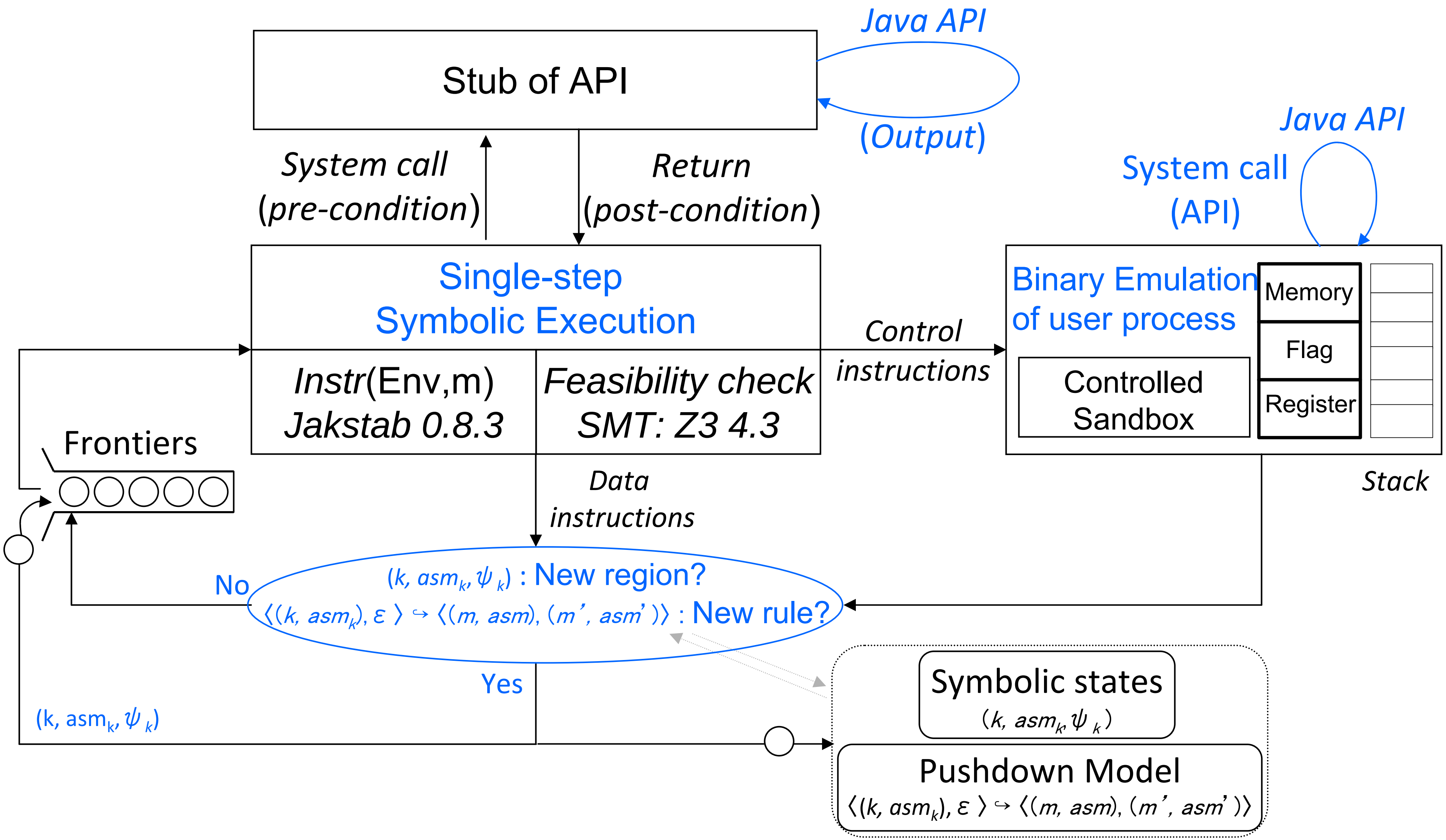
System call (API) as stub

- Symbolic execution requires the conversion from *precondition* to *postcondition* of an API.
  - ✓Obeying to Microsoft Developer Network.
  - ✓Output of API is detected by JavaAPI.

- For instance, *GetModuleFileNameA*
  - ✓*Pre*: *Stack config.*
  - ✓*Post*: EAX= size_file_name



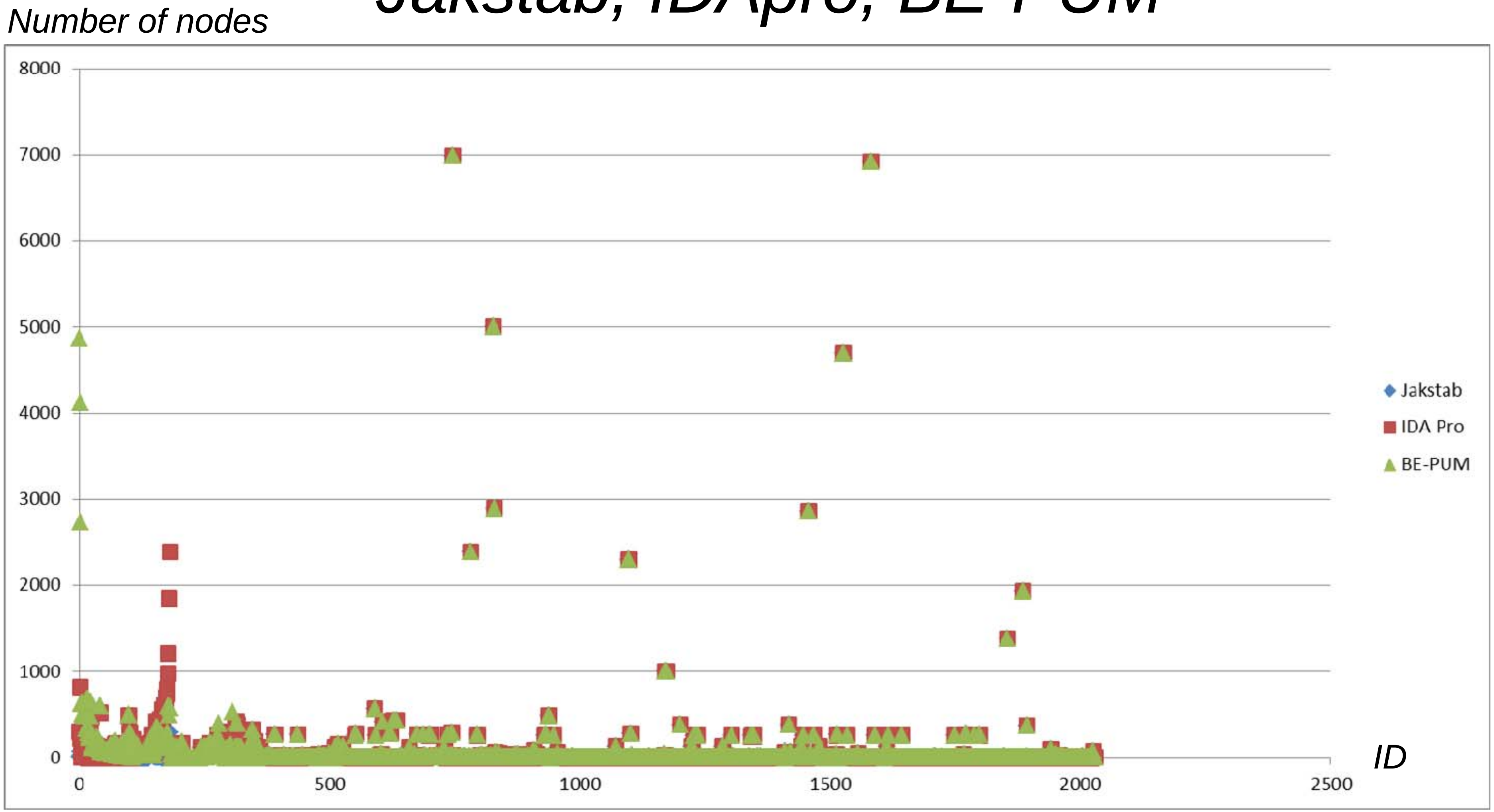
BE-PUM (Binary Emulation for Pushdown Model) *Architecture*



Roadmap

- Background : *Obfuscation techniques* and *aim*
- Anti-obfuscation : *Principle ideas*
- BE-PUM (Binary Emulation for Pushdown Model generation) Implementation : *Practical design*
- Experiments : *Statistics, observation*, and *limitation*
- Related and Future work :

Experiments on 2028 malwares
Jakstab, IDApro, BE-PUM



- Generally, *Jakstab* terminates much earlier, *IDApro* is quite imprecise, compared to *BE-PUM*

Experiment statistics (converged case)

Example	Size	JakStab			IDA Pro			BE-PUM		
	KByte	Nodes	Edges	Time	Nodes	Edges	Time	Nodes	Edges	Time
Email-Worm.Win32.Coronex.a	12	26	27	500ms	148	157	204ms	308	339	1000ms
Trojan-PSW.Win32.QQRob.16.d	25	89	100	766	17	15	382	91	105	953
Virus.Win32.Aidlot	8	81	81	281	64	62	119	105	108	70344
Virus.Win32.Aztec	8	8	102	103	223	215	495	247	259	24384
Virus.Win32.Belial.a	4	41	42	407	118	116	198	128	134	985
Virus.Win32.Belial.b	4	43	44	406	118	116	197	139	146	906
Virus.Win32.Belial.d	4	6	5	328	147	150	158	163	170	1062
Virus.Win32.Benny.3219.a	8	138	153	890	599	603	415	149	164	2438
Virus.Win32.Benny.3219.b	12	42	47	453	745	760	200	149	164	2375
Virus.Win32.Benny.3223	12	42	47	328	770	781	135	149	164	2218
Virus.Win32.Bogus.4096	38	87	98	546	88	86	269	88	98	656
Virus.Win32.Brof.a	8	17	17	343	98	102	167	137	147	1484
Virus.Win32.Cerebrus.1482	8	6	5	156	164	165	70	179	198	735
Virus.Win32.Compan.a	8	25	26	360	83	81	176	91	98	484
Virus.Win32.Compan.b	8	21	22	328	68	71	160	83	86	391
Virus.Win32.Cornad	4	21	20	141	68	72	67	94	100	344
Virus.Win32.Eva.a	8	14	13	329	381	392	145	249	277	13438
Virus.Win32.Eva.b	12	14	13	172	549	553	59	229	252	3515
Virus.Win32.Eva.c	8	14	13	188	448	451	72	292	321	32532
Virus.Win32.Eva.d	8	14	13	156	377	381	59	245	272	11109
Virus.Win32.Eva.e	20	14	13	204	449	456	80	293	321	15375
Virus.Win32.Eva.f	8	14	13	187	350	361	76	204	225	3672
Virus.Win32.Eva.g	8	14	13	188	410	421	74	240	261	3860
Virus.Win32.Htrip.a	8	10	10	359	145	143	172	148	157	2187
Virus.Win32.Htrip.b	8	10	10	343	144	142	164	149	157	2250
Virus.Win32.Htrip.d	8	10	10	265	164	162	124	165	173	2296
Virus.Win32.Seppuku.1606	8	131	136	1968	381	390	965	339	364	8372
Virus.Win32.Wit.a	4	54	60	360	153	151	172	185	203	2641
Virus.Win32.Wit.b	4	7	7	203	168	166	93	197	214	2000
Virus.Win9x.I13.b	12	37	37	313	239	240	145	239	245	890
Virus.Win9x.I13.c	8	37	37	172	117	115	80	117	116	500
Virus.Win9x.I13.f	8	41	41	188	131	137	87	131	141	422
Virus.Win9x.I13.h	14	41	41	203	238	242	95	238	258	4891

Indirect jump (Virus.Win32.Aztec)

SEH (Virus.Win32.Benny.3219.a, Virus.Win32.Benny.3219.b, Virus.Win32.Benny.3223)

SEH (Virus.Win32.Eva.a – Virus.Win32.Eva.g)

SEH & SMC (Virus.Win32.Seppuku.1606)

Observation on experiments of virus

- With source code: *Aztec, Bagle, Benny, Cabanas*
  - ✓Jakstab often fails to find the entry.
  - ✓IDApro may explore more, but in a wrong direction.
  - ✓BE-PUM is under-approximation, even when it converges. Often terminate with *unknown instruction, API, and address (e.g., system EH).*

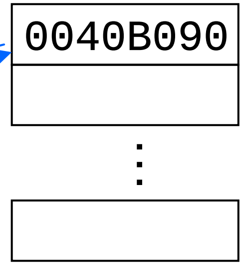
- Without source code: *Seppuku.1606*
  - ✓From differences between results of BE-PUM and IDApro, we found *SEH* and *self-modification*.

Observation: *Indirect jump*

- *Bagle.bf*

```
0040B08A   call 0040B332
0040B08F   E8 EB 13 EB 02
```

ecx 0040B090



stack

```
0040B332  pop     ecx                        popl  %ecx
0040B333  inc     ecx                        incl  %ecx
0040B334  jmp     short loc_40B326           jmp   0x0040b326
0040B326  jmp     short loc_40B32B           jmp   0x0040b32b
0040B32B  push    ecx                        pushl %ecx
0040B32C  retn                               ret

0040B08F  call    near ptr 32BC47Fh
0040B090                                     jmp   0x0040b0a5
```

IDApro *BE-PUM*

- *Aztec (well-investigated)*
  - ✓Similar techniques, and looks for the base address of *kernel32.dll.*

Observation : *SEH* (Structural Error Handler)

- Eva.a : exception occurrence is obfuscated.
  - ✓ As Windows standard, `fs:[0]` initially points to the system exception handler.
  - ✓ New frame pushed at 00401012 and modified at 00401015.
  - ✓ At 00401018, access violation (inc at 00000000).

```
00401010 xor    edx, edx                                  edx = 0
00401012 push   dword ptr fs:[edx]                        esp = 00401007
00401015 mov    fs:[edx], esp ; Overwrite esp on fs:[0]
00401018 inc    dword ptr [edx]                           Violation occurs!
0040101A sub    eax, 10068h
```

(a) IDA Pro			(b) JakStab		(c) BE-PUM	
00401002			call	0x00401010	call	0x00401010
00401010	xor	edx, edx	xorl	%edx, %edx	xorl	%edx, %edx
00401012	push	dword ptr fs:[edx]	pushl	%fs:(%edx)	pushl	%fs:(%edx)
00401015	mov	fs:[edx], esp	movl	%esp, %fs:(%edx)	movl	%esp, %fs:(%edx)
00401018	inc	dword ptr [edx]	incl	(%edx)	incl	(%edx)
0040101A	sub	eax, 10068h	subl	$0x10068, %eax		
00401007					movl	0x8(%esp), %esp

(a) IDA Pro (b) JakStab (c) BE-PUM

Observation : *Self-decryption*

Example	Size	JakStab			IDA Pro			BE-PUM		
	KByte	Nodes	Edges	Time	Nodes	Edges	Time	Nodes	Edges	Time
Virus.Win32.Canabas.2999	8	2	1	656	7	6	85	358	401	8703

- Cabanas.2999: Self-decryption + SEH

```
004047ed   lods   al, ds:[esi]
004047ee   rol    al, cl
004047f0   xor    al, ffffffb5h
004047f2   jns    00404814h
00404814   stos   es:[edi], al
00404815   jne    00404819
00404819   loop   004047ed
```

XORing key

ecx was set to 1a1h

Decryption loop

```
004047de   stosl %eax, %es:(%edi)
004047df   movl  %esp, %fs:(%ebx)
004047e2   pusha
004047e3   xchgl eax, -2(%ebx)
00404841   movl  0x8(%esp), %eax
00404845   leal  -32(%eax), %esp
00404848   popa
```

eax= FFFFFFFE

Access violation

SEH

Investigation of Seppuku.1606

- Manual investigation with help of *Ollydbg* …

Opcode at 00401646: *E8FFFFF9B5* → *E800000000*

```
00401028 xor   eax, eax
0040102A push  dword ptr fs:[eax]
0040102D mov   fs:[eax], esp
00401030 mov   esi, 77E80000h
00401035 lods  ds:[esi]
```

SEH

SEH technique

```
004010E4 PUSH EDI
004010E5 MOV EAX,
      DWORD PTR SS:[EBP+401489]
004010EB STOS DWORD PTR ES:[EDI]
004010EC ADD ESP,4
```

Self-modification

```
00401646  call  sub_401000
00401000  pusha
00401001  call  $+5
00401006
```

(a) IDA Pro

```
call  0x00401000
pusha
call  0x00401006
movl  (%esp), %ebp
```

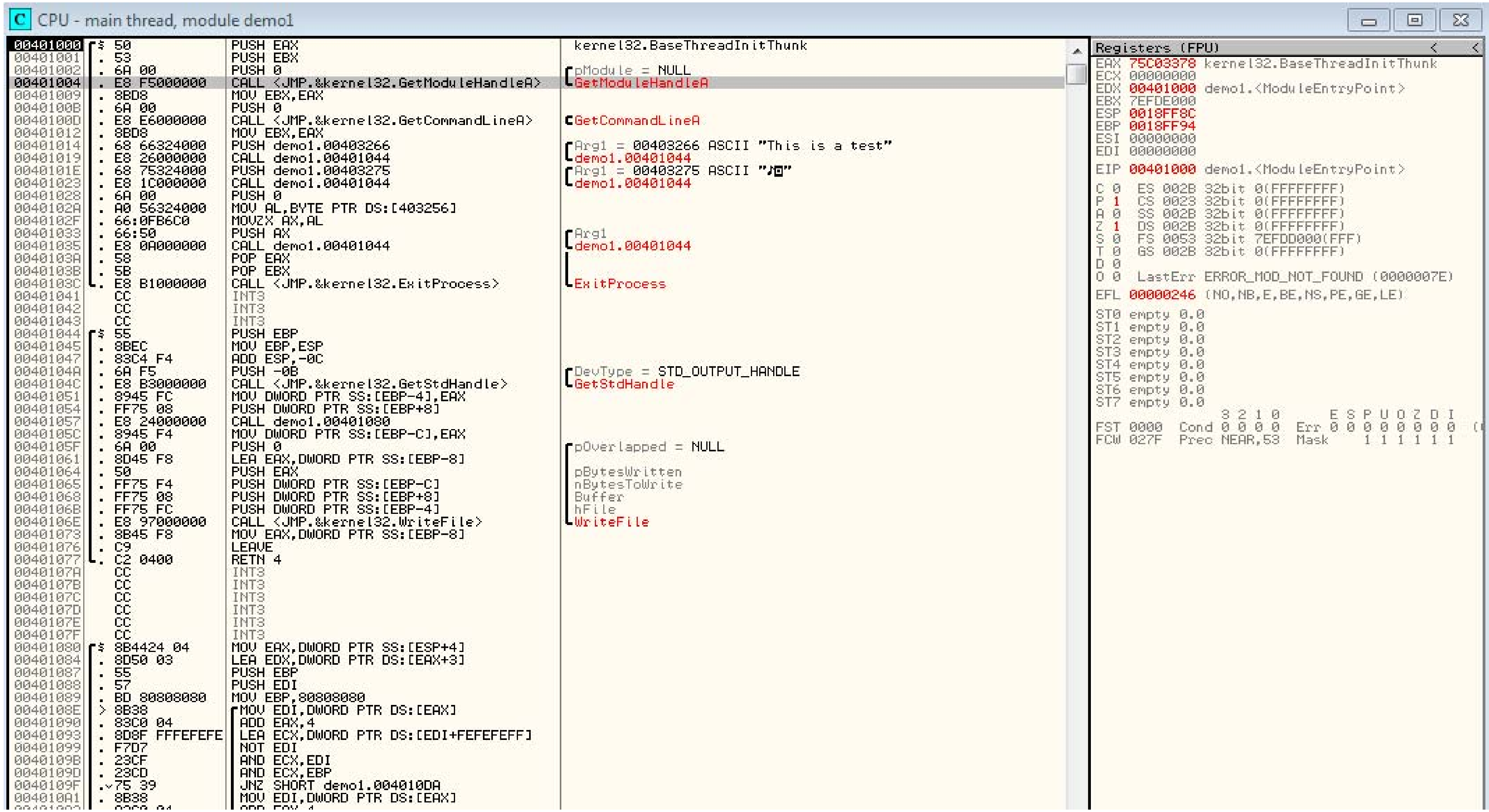
(b) JakStab

```
00401646 call  0x0040164b
0040164b pushl $0x10<UINT8>
0040164d pushl $0x402000<UINT32>
00401652 pushl $0x402027<UINT32>
00401657 pushl $0x0<UINT8>
00401659 call  0x0040166b
0040166b jmp   MessageBoxA@user32.dll
0040165e pushl $0x0<UINT8>
00401660 call  0x00401665
00401665 jmp   ExitProcess@kernel32.dll
```

(c) BE-PUM

OllyDbg (www.ollydbg.de)

- 32bit assembler level analyzing debugger for windows



When *branches are missed*

- Typical number of branch : 20 branches in length 500 (Windows/System32/HOSTNAME.exe, 12k bytes)

- Missing reasons
  - ✓ ***Opaque predicates***. BE-PUM correctly detects in Cabanas.2999.
  - ✓ ***API stub*. ** API output is given by JavaAPI (just one instance in the environment), and assumptions.
  - ✓ ***Loop unfolding***. Bounded unfolding of a loop may miss later exit from the loop.

Roadmap

- Background : *Obfuscation techniques* and *aim*
- Anti-obfuscation : *Principle ideas*
- BE-PUM (Binary Emulation for Pushdown Model generation) Implementation : *Practical design*
- Experiments : *Statistics, observation*, and *limitation*
- Related and Future work

Related work: model generation (binary CFG rebuilt)

- Static analysis
  - ✓CodeSurfer/x86 (CC04/05) : *Memory-as-state, static analysis comes first.*
  - ✓McVeto (CAV10) : On-the-fly pushdown model generator, CEGAR is used for indirect jumps.
  - ✓JakStab (VMCAI09,12): BE-PUM built on JakStab

- Dynamic testing
  - ✓BIRD (CGO06) : Disassembly
  - ✓BINCORE/OSMOSE (CAV11): Memory-as-state, DBA (Dynamic Bit-vector Automaton)
  - ✓Syman (ICSE06) : On-the-fly diassembly, Windows emulator Alligator (not conclic testing)

Related work

- Pushdown model checking
  - ✓SCTPL (TACAS12), SLTPL (TACAS13)
    - Target on binaries without self-modification (IDApro can handle)
    - Malicious behavior = system calls

- Self-decryption, packer
  - ✓PolyPack (ACSAC06) : Testing based
  - ✓Renovo (RM07)
  - ✓At Nancy/LORIA: Trace analysis

Future work

- Conformance testing of generated models.
  - ✓Formalization of semantics of x86/API is difficult.
- Weighted pushdown model checking.
  - ✓*Target*: Obfuscation, infection, malicious behavior
  - ✓Towards automatic obfuscation classification.
- Loop handling
  - ✓More precise under-approximation.