The System of Automatic Searching for Vulnerabilities or how to use Taint Analysis to find vulnerabilities

Alex Bazhanyuk (@ABazhanyuk)
Nikita Tarakanov (@NTarakanov)
Who is Alex Bazhanyuk

- Security Researcher
- Organizer of Defcon Ukraine Group
- Working in UC Berkley in BitBlaze project
- Solves problems of automation of RE
Who is Nikita Tarakanov

• Independent Security Researcher
• Author of some articles in ]akep magazine
• Likes to reverse engineer r0 parts
• Discovered a lot of LPE vulns
• Solves problems of automation of RE
Agenda

• Intro
• Taint analysis theory
• BitBlaze theory
• SASV implementation
• Lulz Time
• Pitfalls
• Conclusion
SASV main parts

- IDA Pro plugins
- BitBlaze: Vine+utils, TEMU + plugins
Theory

1. Evolution should not be taught in our schools.
   Why not?

2. Because it's just a theory!

3. But isn't all of science "just a theory"?

4. Well... that will leave a lot more time for gym.
Tainting

• Taint sources:
  Network, Keyboard, Memory, Disk, Function outputs

• Taint propagation: a data flow technique
  Shadow memory

Whole-system
  Across register/memory/disk/swapping
Fundamentals of taint analysis
Taint propagation

• If an operation uses the value of some tainted object, say X, to derive a value for another, say Y, then object Y becomes tainted. Object X taints the object Y.

• Taint operator $t$

• $X \rightarrow t(Y)$

• Taint operator is transitive

$X \rightarrow t(Y)$ and $Y \rightarrow t(Z)$, then $X \rightarrow t(Z)$
Static Taint Analysis

Analysis performed over *multiple paths* of a program

* Typically performed on a control flow graph (CFG):

  statements are nodes, and there is an edge between nodes if there is a possible transfer of control.
BitBlaze: Binary Analysis Infrastructure

- Automatically extracting security-related properties from binary code
- Build a unified binary analysis platform for security
  - Static analysis + Dynamic analysis + Symbolic Analysis
  - Leverages recent advances in program analysis, formal methods, binary instrumentation...

Solve security problems via binary analysis
- More than a dozen different security applications
- Over 25 research publications
BitBlaze

- http://bitblaze.cs.berkeley.edu/
- TEMU, VINE
- Rudder, Panorama, Renovo
Confines TEMU

- Only gcc-3.4
- Qemu 0.9.1 - TEMU
- Qemu 0.10 - TCG(Tiny Code Generator)-TODO
- Qemu 0.10 ↔ Qemu 1.01
Fig. 2. Vine Overview
The Vine Intermediate Language

\[\text{program} ::= \text{decl}* \text{ instr}^* \]
\[\text{instr} ::= \text{var} = \text{exp} | \text{jmp} \text{ exp} | \text{cjmp}\ \text{exp, exp, exp} | \text{halt}\ \text{exp} | \text{assert}\ \text{exp} \]
\[\quad \quad \quad | \text{label}\ \text{integer} | \text{special}\ \text{id}_s\]
\[\text{exp} ::= \text{load}(\text{exp}, \text{exp}, \tau_{\text{reg}}) | \text{store}(\text{exp}, \text{exp}, \text{exp}, \tau_{\text{reg}}) | \text{exp} \triangleleft_b \text{ exp} | \triangleleft_u \text{ exp} \]
\[\quad \quad \quad | \text{const} | \text{var} | \text{let}\ \text{var} = \text{exp}\ \text{in}\ \text{exp} | \text{cast}(\text{cast kind}, \tau_{\text{reg}}, \text{exp})\]
\[\text{cast kind} ::= \text{unsigned} | \text{signed} | \text{high} | \text{low}\]
\[\text{decl} ::= \text{var}\ \text{var}\]
\[\text{var} ::= (\text{string, id}_v, \tau)\]
\[\triangleleft_b ::= +, -, *, /, /_s, \text{mod,}\ \text{mod}_s, \ll, \gg, \gg_a, \&, |, \oplus, ==, \neq, <, \leq, <_s, \leq_s\]
\[\triangleleft_u ::= -\ (\text{unary minus}), !\ (\text{bit-wise not})\]
\[\text{value} ::= \text{const} | \{ n_{a1} \rightarrow n_{v1}, n_{a2} \rightarrow n_{v2}, \ldots \}: \tau_{\text{mem}} \mid \bot\]
\[\text{const} ::= n : \tau_{\text{reg}}\]
\[\tau ::= \tau_{\text{reg}} | \tau_{\text{mem}} | \text{Bot} | \text{Unit}\]
\[\tau_{\text{reg}} ::= \text{reg1}_t | \text{reg8}_t | \text{reg16}_t | \text{reg32}_t | \text{reg64}_t\]
\[\tau_{\text{mem}} ::= \text{mem}_t(\tau_{\text{endian}}, \ \tau_{\text{reg}})\]
\[\tau_{\text{endian}} ::= \text{little} | \text{big} | \text{norm}\]
Example of disasm:

```
fc32dce:  rep stos %eax,%es:(%edi)  R@eax[0x00000000][4](R) T0
          R@ecx[0x00000002][4](RCW) T0  M@0xfb7bff8[0x00000000][4](CW) T1 {15
          (1231, 69624) (1231, 69625) (1231, 69626) (1231, 69627) }

fc32dce:  rep stos %eax,%es:(%edi)  R@eax[0x00000000][4](R) T0
          R@ecx[0x00000001][4](RCW) T0  M@0xfb7bfffc[0x00000000][4](CW) T1 {15
          (1231, 69628) (1231, 69629) (1231, 69630) (1231, 69631) }

fc32dce:  mov %edx,%ecx    R@edx[0x0000015c][4](R) T0
          R@ecx[0x00000000][4](W) T0

fc32dcf:  and $0×3,%ecx    l@0×00000000[0x00000003][1](R) T0
          R@ecx[0x0000015c][4](RW) T0

fc32dcf:  andl $0×0,-0×4(%ebp) l@0×00000000[0x00000000][1](R) T0
          M@0xfb5ae738[0x00000002][4](RW) T0

fc32dcf:  jmp 0x00000000fc32c726  J@0×00000000[0xffffea2d][4](R) T0

fc32c72:  cmpl $0×0,-0×58(%ebp) l@0×00000000[0x00000000][1](R) T0
          M@0xfb5ae6e4[0x00000000][4](R) T0
```
Taint info

• T0 - means that the statement did not tainted.
• T1 - means that the instruction tainted in curly brackets can be seen that there tainted and what it depends.
• Here's an example of:
• fc32dce: rep stos% eax,% es: (% edi) R @ eax [0x00000000] [4] (R) T0 R @ ecx [0x00000001] [4] (RCW) T0 M @ 0xfb7bfffec [0x00000000] [4] (CW) T1 {15 (1231, 628) (1231, 629) (1231, 630) (1231, 631)}
• 4 bits of information tainted and they depend on the offset: 628, 629, 630, 631. 1231 - this number is origin(kind of ID that TEMU plugin sets).
appreplay

• ./vine-1.0/trace_utils/appreplay -trace font.trace -ir-out font.trace.il -assertion-on-var false-use-post-var false

where:
• appreplay - ocaml script that we run;
• -trace - the way to the trace;
• -ir-out - the path to which we write IL code.
• -assertion-on-var false-use-post-var false - flags that show the format of IL code for this to false makes it more readable text.
Example of IL code:

• Begins with the declaration of variables:
• INPUT - it's free memory cells, those that are tested in the very beginning (back in temu), input into the program from an external source.

```plaintext
var cond_000017_0x4010ce_00_162:reg1_t;

var cond_000013_0x4010c3_00_161:reg1_t;
var cond_000012_0x4010c0_00_160:reg1_t;
var cond_000007_0x4010b6_00_159:reg1_t;
var INPUT_10000_0000_62:reg8_t;
var INPUT_10000_0001_63:reg8_t;
var INPUT_10000_0002_64:reg8_t;
var INPUT_10000_0003_65:reg8_t;

var mem_arr_57:reg8_t[4294967296]; — memory as an array
var mem_35:mem32l_t;
```
R_EAX_5:reg32_t = 0x73657930:reg32_t;
{
  var idx_144:reg32_t;
  var val_143:reg8_t;
  idx_144:reg32_t = 0x12fef0:reg32_t;
  val_143:reg8_t = INPUT_10000_0000_62:reg8_t;
  mem_arr_57[idx_144:reg32_t + 0:reg32_t]:reg8_t =
      cast((val_143:reg8_t & 0xff:reg8_t) >> 0:reg8_t)L:reg8_t;
}

T_32t2_60:reg32_t = R_ESP_1:reg32_t;
T_32t1_59:reg32_t = T_32t2_60:reg32_t + 0x1c8:reg32_t;
T_32t3_61:reg32_t = (
    cast(mem_arr_57[T_32t1_59:reg32_t + 0:reg32_t]:reg8_t)U:reg32_t
    << 0:reg32_t
    |
    cast(mem_arr_57[T_32t1_59:reg32_t + 1:reg32_t]:reg8_t)U:reg32_t
    << 8:reg32_t
    |
    cast(mem_arr_57[T_32t1_59:reg32_t + 2:reg32_t]:reg8_t)U:reg32_t
    << 0×10:reg32_t
    |
    cast(mem_arr_57[T_32t1_59:reg32_t + 3:reg32_t]:reg8_t)U:reg32_t
    << 0×18:reg32_t
    );
R_EAX_5:reg32_t = T_32t3_61:reg32_t;
What is STP and what it does?

- STP - constraint solver for bit-vector expressions.
- separate project independent of the BitBlaze
- To produce STP code from IL code:
  - ./vine-1.0/utils/wputil trace.il -stpout stp.code
- where the input is IL code, and the output is STP code
mem_arr_57_8 : ARRAY BITVECTOR(64) OF BITVECTOR(8);
INPUT_10000_0000_62_4 : BITVECTOR(8);

ASSERT( 0bin1 =
(LET R_EAX_5_232 =
0hex73657930
IN
(LET idx_144_233 =
0hex0012fef0
IN
(LET val_143_234 =
INPUT_10000_0000_62_4
IN
(LET mem_arr_57_393 =
(mem_arr_57_8 WITH [(0bin00000000000000000000000000000000 @ BVPLUS(32,
idx_144_233,0hex00000000))) := (val_143_234;0hexff)[7:0])
....... IN (cond_000017_0x4010ce_00_162_392;0bin1)))));

Is this expression false?

QUERY (FALSE);
And give a counter example:

COUNTEREXAMPLE;
STP output example

• How to ask for a decision to STP:

• ./stp stp.code

• Example of STP output:

  ASSERT( INPUT_10000_0001_63_5 = 0x00 );
  ASSERT( INPUT_10000_0002_64_6 = 0x00 );
  ASSERT( INPUT_10000_0000_62_4 = 0x61 );
  ASSERT( INPUT_10000_0003_65_7 = 0x00 );

Invalid.
**SASV Components:**

- **Temu** (tracecap: start/stop tracing. Various additions to tracecap(hooks etc.))
- **Vine** (appreplay, wputil)
- **STP**
- **IDA plugins:**
  - *DangerousFunctions* – finds calls to malloc,strcpy,memcpy etc.
  - *IndirectCalls* – indirect jumps, indirect calls.
  - *ida2sql* (zynamics) – idb in the mysql db. (http://blog.zynamics.com/2010/06/29/ida2sql-exporting-ida-databases-to-mysql/)
- **Iterators** – wrapper for temu, vine, stp.
- **Various publishers** – for DeviceIoControl etc.
How does SASV work?
SASV

• Scheme:

• Min Goal: max coverage of the dangerous code
• Max Goal: max coverage of the all code
SASV basic algorithm

1. Work of IDA plugins -> dangerous places
2. Publisher(s) -> invoke targeted code
3. TEMU -> trace
4. Trace -> appreplay -> IL
5. IL -> change path algo -> IL’
6. IL’ -> wputil -> STP_prorgam’
7. STP_prorgam’ -> STP -> data for n+1 iteration
8. Goto #2
Diagram for new path in graph

input data → software

TEMU

Trace, alloc-file, state

Vine

appreplay

Changer, symbolic execution

IL code

Next Iteration

New input data

stp

Stp’ code

IL’ code
Combo system: Dumb+Smart

1. Input data
2. SASV
   - Set of new input data
3. Coverage
   - Set of new input data
4. Blackbox fuzzer
Disadvantages

• Definition of the vulnerability is difficult task.
• Performance – speed of tracing in TEMU is **AWFUL**
Overhead

• Ideally – 1/1000.

• In Reality - 1/(X * 10000)

• Where X is dynamic and could be 1 to 10^N

• Depends on your target (r3, r0)

• Hooks quantity, etc
Implementation(Vine<->TEMU) issues

- VEX != XED
- VEX is part of valgrind - used for R3.
- Formula only for single thread.
Get rid of that damned QEMU!

• Move taint propagation to Hypervisor!

• Damn good idea!

• But a lot of code to port/rewrite
Automation of Exploit Generation

• Build Primitives (correct exploitation state)!

• A lot of exploit mitigations

• EIP tainted != pwnage (nowadays)
S2E + SASV

S2E=Qemu+Klee
Klee=LLVM+Stp

• Input data => taint analysis (new concept)
• Support ARM
• Support Qemu 0.12
Vulnerabilities in drivers

• Overflows: stack, pool, integer
• Pointer overwrite
• Null pointer dereference (Plague)
• Race condition (Plague)
• Various logical vulnerabilities (how to automate?)
Example of issue

• Total = var1 * var2 (var – could be const)

• Mem = malloc(Total)

• For(i=0;i<var;i++)memcpy(Mem, Mem2, CONST)

• Free(Mem)
Define Vulnerability (Memory corruption)

- $\text{var} = \text{var1 operation var2}$
- $\text{Mem} = \text{alloc(heap, stack)}(\text{var})$
- $\text{Mem}[\text{var3}] = \text{var4}$
- Could $\text{var3} > \text{var}$ (write out-of-bounds)?
Define vulnerability

1. tainted eip. (very rare in real life, look at KingSoft AV)

2. pointers and operations on them.

3. buffer overflow (hook *alloc function and change size of alloc).

4. integer operations and results(VSA).

5. Threads race condition – is there using of synchronising functions?
Attack vectors(r3->r0)

- IOCTL

- SSDT hooks(Native & Shadow)

- various notification routines
DeviceIoControl

• Parameters:
  – hDevice
  – dwIoControlCode
  – lpInBuffer
  – nInBufferSize
  – lpOutBuffer
  – nOutBufferSize
  – lpBytesReturned
  – lpOverlapped
Concept

IOCTL:

Data to taint:

- `dwIoControlCode` - to get list of supported ioctl codes
- `lpInBuffer` - pointer(METHOD_NEITHER) and data (METHOD_BUFFERED)
- `nInBufferSize` - size ranges
- `lpOutBuffer` - pointer(METHOD_NEITHER) and data (METHOD_BUFFERED)
- `nOutBufferSize` - size ranges

*Tracing only driver code*
Shaming examples

• *Lulz Time!*
GData Lulz #0: Minilcpt.sys

- ioctl code 0x83170180 (METHOD_BUFFERED)
- Untrusted data goes to FtlReleaseContext
- Leads to decrement arbitrary memory
- Leads to control of EIP
- TotalCare 2011->2012 (20 months old 0day)
- Wooot TotalCare 2013 fixed feature 😞
GData Lulz #1: GDNDisIc.sys

- What about control over Ndis Filter?
- 0x830020E0 – NPD + switching on/off
- 0x83002108 – switching on/off AutoPilot
- First trigger as non-interesting vuln(NPD)
- But log from DbgPrint shows Lulz
Agnitum (?) VBEngNT.sys FAIL

• VBEngNT.sys – NOT Agnitum code
• VBEngNT.sys – from VirusBuster!
• Plays dll role in kernel land
• 50(!!!) vulnerable functions – one stupid bug
• Full trust on pointers
• Using by several(over 8) products
• Test before you buy some r0(!!!) code!!!
Microsoft Features

• METHOD_BUFFERED “signal”

• METHOD_IN/OUT_DIRECT

• ProbeForRead/ProbeForWrite – known for ages,

  but MS itself FAILS sometime
GData Lulz #2: TS4nt.sys

- New!!!!
- Total Care 2013(future!!!)
- Processes several ioctls
- METHOD_BUFFERED “signal” (NPD)
- Uses pointer than check – smart!
METHOD_BUFFERED “signal”

- CA Internet Security KmxFw(0x85000800)
- CA Internet Security KmxAmrt(0x8E000800)
- CA Internet Security KmxCfg (0x8700004A)
- CA Internet Security KmxCfg (0x87000800)
- Total 4 stupid shutdown features of HIPS! :D
Vipre ISS 2012 SBREDrv.sys

- Rebooting ioctlS: 0x22C418, 0x22C1C, 0x22C0CC
- Kernel Pool Corruptions: 0x22C104, 0x22C108, 0x22C10C, 0x22C110, 0x22C124, 0x22C180
- Total 3(features ) + 6 vulns
- + also presented in Unthreat,LavaSoft products
TrendMicro tmtdi.sys #1

- ioctl code 0x220044 (METHOD_BUFFERED)
- No range check for size
- Just check for correct address – NPD check (MmIsAddressValid)
- Pool corruption in cycle
- No control of overflowing data 😞
TrendMicro tmtdi.sys #1

- `.text:0001D881` mov edi, [ebx+0Ch]
- `.text:0001D884` push edi ; our buffer
- `.text:0001D885` call esi ; `MmIsAddressValid`
- `.text:0001D887` test al, al
- `.text:0001D889` jz loc_1DDAB
- `.text:0001D88F` push [ebp+output_buff_size]
- `.text:0001D892` push edi
- `.text:0001D893` push offset rules_list
- `.text:0001D898` call ioctl_0x220044_vuln
- [..]
TrendMicro tmtdi.sys #1

- .text:000156EA    mov    ebx, [ebp+our_buffer_size_controlled]
- .text:000156ED    mov    [ebp+NewIrql], al
- .text:000156F0    mov    eax, dword_22CA0
- .text:000156F5    mov    edx, offset dword_22CA0
- .text:000156FA    cmp    eax, edx
- .text:000156FC    jz     short loc_15748
- [...]
- .text:00015700    mov    ecx, [eax+0Ch]
- .text:00015703    mov    [ebx], ecx
- .text:00015705    mov    ecx, [eax+10h]
- .text:00015708    mov    [ebx+4], ecx
- .text:0001570B    mov    ecx, [eax+14h]
- .text:0001570E    mov    [ebx+8], ecx ← write outside of the pool chunk
- .text:00015711    mov    ecx, [eax+18h]
- .text:00015714    mov    [ebx+0Ch], ecx
TrendMicro tmtdi.sys #2

- ioctl code 0x220030
- Range check for inbuff_size >= 0x2AA
- Range check for outbuff_size >= 0x4D0
- Allocs pool memory for const size 0x4D0
- And...
- Zeroing it with outbuff_size length! LOL
TrendMicro tmtdi.sys #2

- .text:0001D704 cmp [ebp+inbuff_size], 2AAh
- .text:0001D70B jb loc_1DDAB
- .text:0001D711 mov esi, 4D0h
- .text:0001D716 cmp [ebp+output_buff_size], esi
- .text:0001D719 jb loc_1DDAB
- .text:0001D71F push 746D74h ; Tag
- .text:0001D724 push esi ; NumberOfBytes
- .text:0001D725 push 0 ; PoolType
- .text:0001D727 call ds:ExAllocatePoolWithTag
- [...]
TrendMicro tmtdi.sys #2

- `.text:0001D74B` push edi ; pool_mem_const_size
- `.text:0001D74C` lea eax, [ebp+output_buff_size]
- `.text:0001D74F` push eax ; output_buff_size
- `.text:0001D750` push [ebp+NewIrql] ; inbuff
- `.text:0001D753` push 220030h ; ioctl_code
- `.text:0001D758` call ioctl_several_ioctl_codes
- `[..]`
- `.text:00014918` mov esi, [ebp+output_buff_size]
- `[..]`
- `.text:00014977` push dword ptr [esi]
- `.text:00014979` push 0
- `.text:0001497B` push [ebp+pool_mem_const_size]
- `.text:0001497E` call memset
TrendMicro tmnciesc.sys

• ioctl code 0x222404

• Kernel Pool Corruption

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)

• Your homework ;)
Pitfalls of taint analysis

• Indirect propagation

• Flat model problem (data is tainted, pointer is not) – strlen problem

• Const values tainting (switch problem)

• More taint info (levels) – more overhead
Pitfalls of tainting r0

- Taint info lost
- Check of system variables
- System defense mechanism(s) (win32k.sys WATCHDOG BugCheck)
Pitfalls of tainting r0(IOCTL)

• KeGetPreviousMode

• IoGetCurrentProcess

• Even hooking NtDeviceIoControlFile!
Conclusions

• Quality -> security level
• Taint analysis is not key to every vuln
• SASV just another approach to automate RE
• Sucks for userland software analysis
• Nice approach for kernel land
• But fails sometimes ;)
• MS should fuzz/test/analyze what it signs!
Thanks, 😊

•Questions?

http://twitter.com/#!/ABazhanyuk
http://twitter.com/#!/NTarakanov