GPGPU AND THREAT ANALYSIS

Takashi Katsuki

Symantec Security Response

Agenda

1. Introduction
2. Background
3. Actual Tools For Threat Analysis
4. Pros, Cons And Performance Of OpenCL Coding
5. Further Implementation of PyOpenCL
6. Conclusion

About me (1)

- Software engineer at Symantec Security Response
  - Threat analysis & research
  - Developing signaturesß

About me (2)

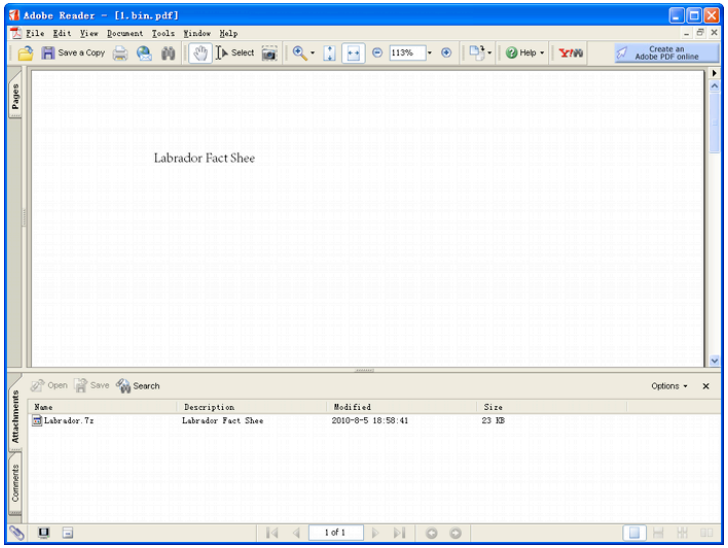
- Trojan.Kardphisher
  - Phishing Trojan horse eg. MS dialog

About me (3)

- Trojan.Pidief family
  - Using ASCII85Decode

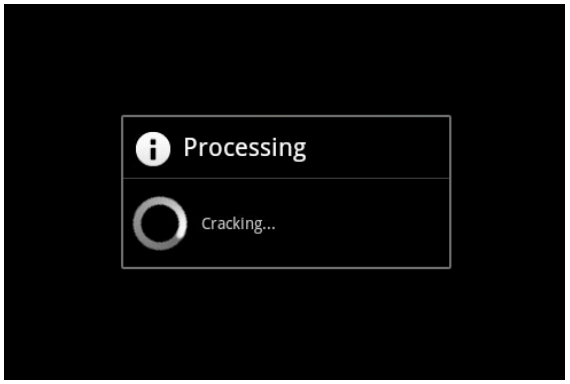
```
<< /Length 2339 /Filter [/ASCII85Decode /FlateDecode]
 >>
stream
Gb"/*=`3(U%),?\QaE\g8?s!=`B<f!F+OJB=b(3i"@^X$iZa;d-2HHp1
endstream
```

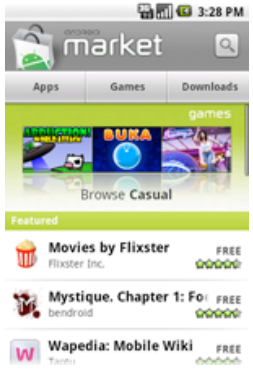
  - PDF as container



About me (4)

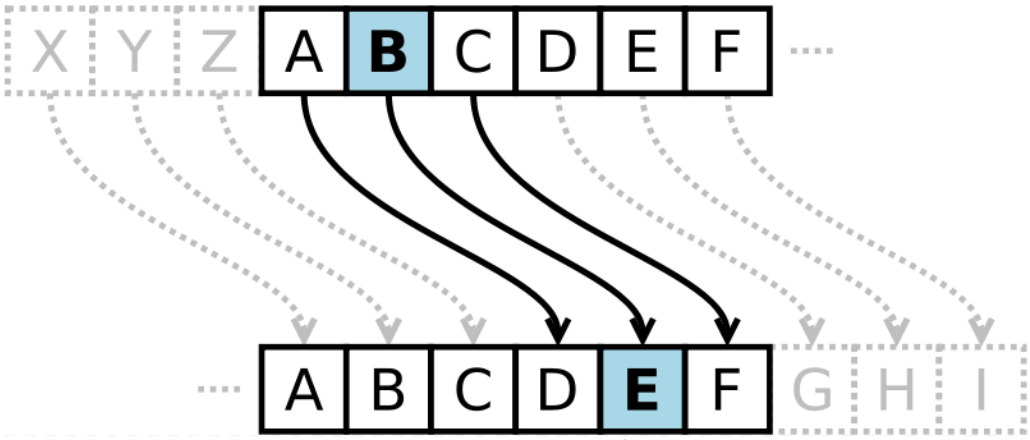
- Some Android threats
  - Android.Walkinwat
  - Android.Uxipp





Inception (1)

- Threats use simple encryption
- If this can be decrypted easier...



Inception (2)

- Brute-forcing is not only for password cracking
- I want to introduce brute-forcing for analysis

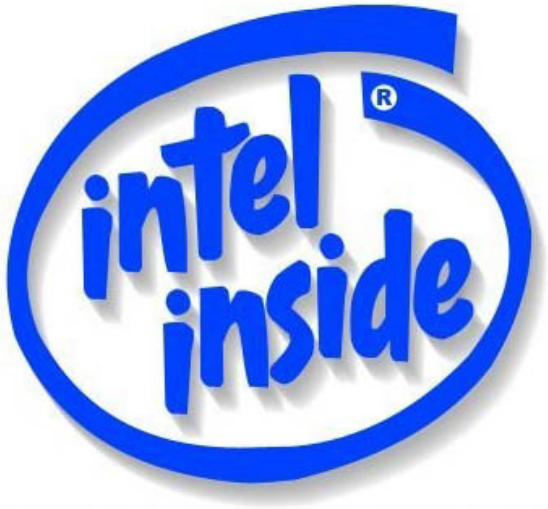
Inception (3)

- Majority of malware employee simple algorithm for encryption such as
  - xor
  - add (sub)
  - rotation

- Encrypted data such as
  - URL
  - Dropped PE file

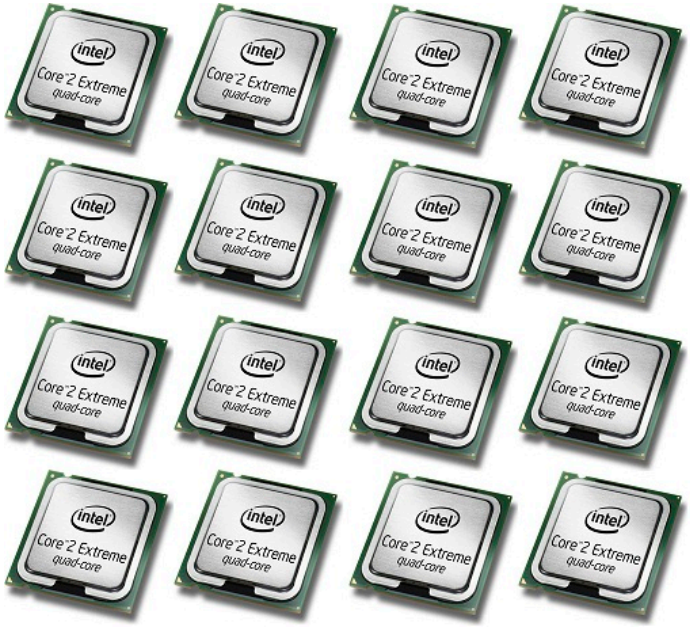
Inception (4)

- How to find it?
- Need more powerful CPU?



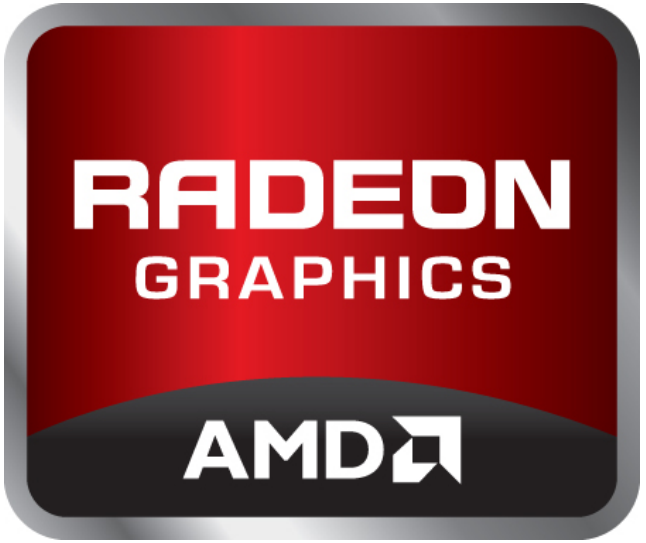
Inception (5)

- Once CPU power depended on frequency
- Now it depends on number of cores
- Need **parallel processing code**



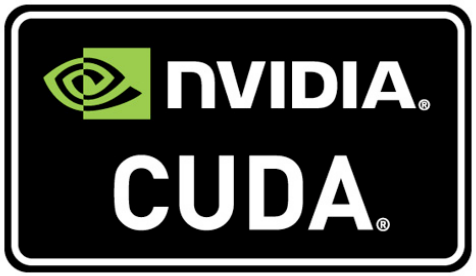
Inception (6)

- Also GPU has many cores to calculate
- GPGPU (General-Purpose Computing on Graphics Processing Units)
- Need **parallel processing code**



Inception (7)

- No one standard on SDKs when it comes to parallel processing
- What's is best solution?



Grand Central Dispatch

OpenMP

Agenda

1. Introduction
2. Background
3. Actual Tools For Threat Analysis
4. Pros, Cons And Performance Of OpenCL Coding
5. Further Implementation of PyOpenCL
6. Conclusion

OpenCL (1)

> OpenCL™ is the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/ embedded devices.

from OpenCL official site

OpenCL (2)

- Cross-platform
- Portable
- For both CPU and GPU
- Khronos Group leaded by Apple

OpenCL (3)

- 2 parts of the structure
  - host code (Standard C programming)
  - kernel code (OpenCL C programming)

PyOpenCL (1)

- Python binding for OpenCL
- host code is now Python code
- Runtime compile for kernel code

PyOpenCL (2)

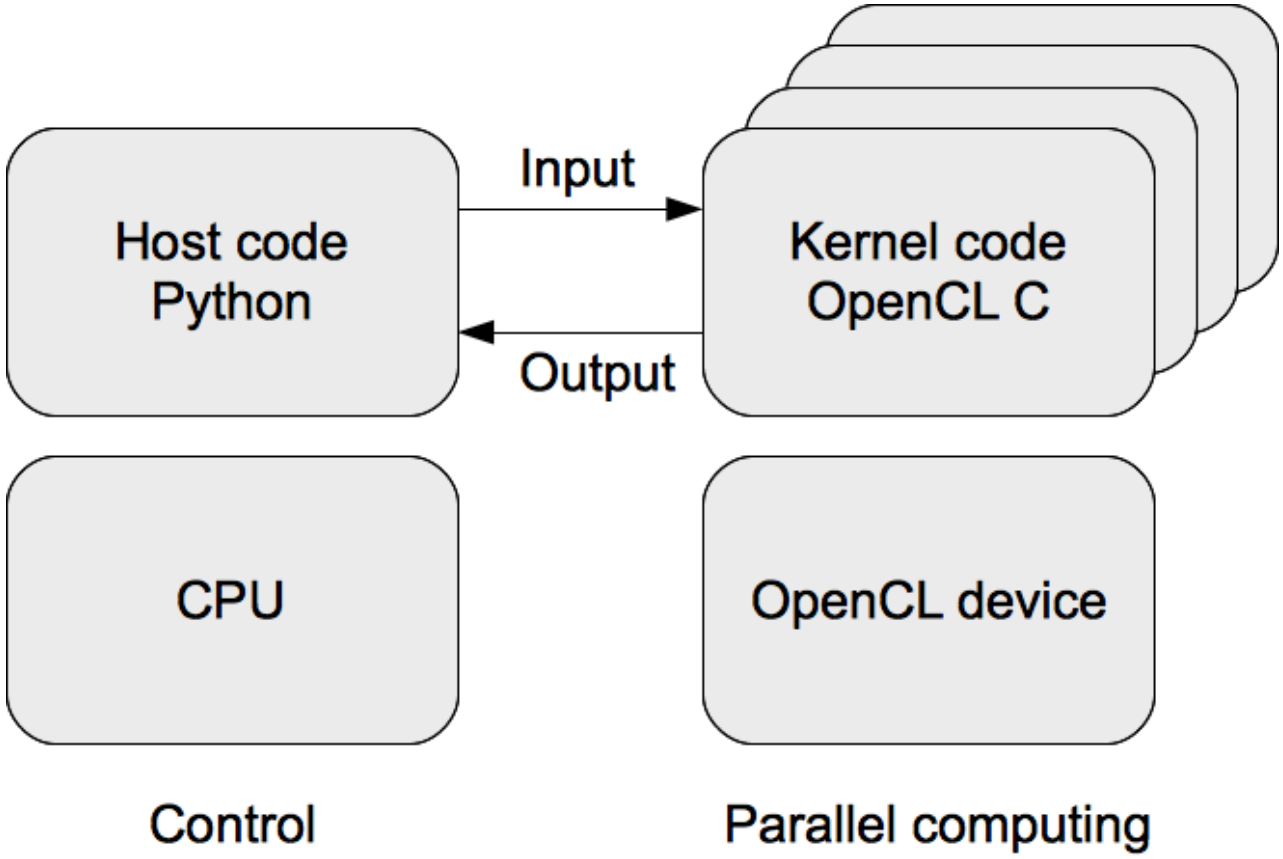
- Easy to set-up for Ubuntu Desktop 11.04
  - Install the NVIDIA official driver from the Additional Drivers menu.
  - Install ‘python-pyopencl’ from the Synaptic Package Manager.

PyOpenCL (3)

- Easy to set-up for Mac OS X Snow Leopard 10.6
  - Install MacPorts
  - Install pyopencl

PyOpenCL (4)

- Basic structure



Case for GPGPU use in security

- "Bitcoin Mining with Trojan.Badminer" by Poul Jensen
  - A Trojan horse uses GPGPU to mine BitCoin

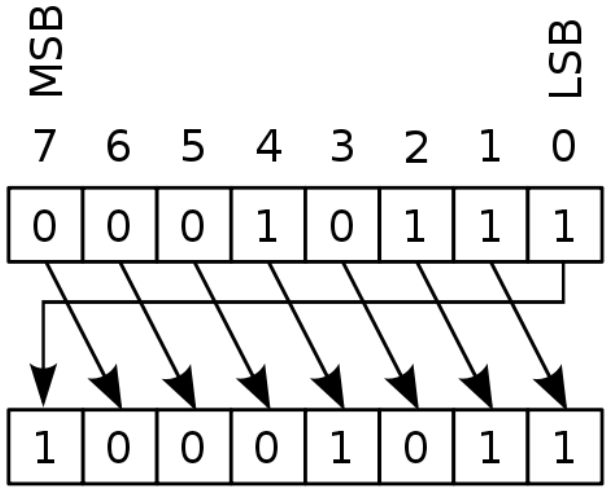
Agenda

1. Introduction
2. Background
3. Actual Tools For Threat Analysis
4. Pros, Cons And Performance Of OpenCL Coding
5. Further Implementation of PyOpenCL
6. Conclusion

Generic keyword search (1)

- Find specific encrypted “keyword” in any binary file
  - Ex. encrypted URL in Downloader

- Simple encryption method
  - xor
  - add (sub)
  - rotation (right and left)



Generic keyword search (2)

- increase or decrease
  - 0 (no increase)
  - +1
  - -1

- This is very common way found in real threats

Generic keyword search (3)

- Encryption block length
  - byte (0x00 - 0xFF)
  - word (0x0000 - 0xFFFF)
  - dword (0x00000000 - 0xFFFFFFFF)

- **Not** performance friendly

Generic keyword search (4)

algorithm	xor	add (sub)	rotation
increase	0	+1	-1
block	byte	word	dword

Generic keyword search (5)

- The main part of the kernel code is on Listing 1 of the Appendix
  - findKeyword()
    - called by host code in parallel
  - searchKeyword()
    - main part of brute-forcing

Generic keyword search (6)

```
for (long k = (inc == 0 ? startValue + 1 : startValue); k <= endValue; ++k) {
    long j;
    for (j = 0; j < keywordSize; ++j) {
        BYTE a = data[j];
        BYTE b = (BYTE)(k + inc * j);
        if (decrypt(function, a, b) != keyword[j]) {
            break;
        }
    } if (j == keywordSize) {
        *resultValue = k; return TRUE;
    }
}
```

Generic keyword search (7)

- The main part of the host code is on Listing 2 of the Appendix
  - host code reads kernel code and compiles it in runtime
  - execute()
    - loads input data
    - initializes memory
      - Buffer
      - LocalMemory
    - enqueues kernel code

Generic keyword search (8)

- kernel code (OpenCL C)
  - 435 instructions
- host code (Python)
  - 171 instructions

```
        if (resultFunction[id] != f_not_found) {
                return;
        }

        uint localPosition = localId * keywordSize;
        for (int i = 0; i < keywordSize; ++i) {
                dataLocal[localPosition + i] = data[id + i];
                keywordLocal[localPosition + i] = keyword[i];
        }

        __local uchar* dataPointer = dataLocal + localPosition;
        __local uchar* keywordPointer = keywordLocal + localPosition;

        if (searchKeyword(dataPointer, keywordPointer, keywordSize, startValue, endVal
        { resultFunction[id] = f_xor | 0x00000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, startValue, endVal
        { resultFunction[id] = f_xor | 0x01000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, startValue, endVal
        { resultFunction[id] = f_xor | 0x02000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, startValue, endVal
        { resultFunction[id] = f_add | 0x00000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, startValue, endVal
        { resultFunction[id] = f_add | 0x01000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, startValue, endVal
        { resultFunction[id] = f_add | 0x02000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, 1, 7, f_ror, 0, re
        { resultFunction[id] = f_ror | 0x00000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, 1, 7, f_ror, 1, re
        { resultFunction[id] = f_ror | 0x01000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, 1, 7, f_ror, -1, r
        { resultFunction[id] = f_ror | 0x02000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, 1, 7, f_rol, 0, re
        { resultFunction[id] = f_rol | 0x00000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, 1, 7, f_rol, 1, re
        { resultFunction[id] = f_rol | 0x01000000; } else
        if (searchKeyword(dataPointer, keywordPointer, keywordSize, 1, 7, f_rol, -1, r
        { resultFunction[id] = f_rol | 0x02000000; } else
        { resultFunction[id] = f_not_found; }

        return;
}

__kernel void
findKeyword_w(
        __global __const uchar* data
,       uint dataSize
,       __local uchar* dataLocal
,       __global __const uchar* keyword
,       uint keywordSize
,       __local uchar* keywordLocal
,       uint startValue
,
```

W32.Qakbot!conf (1)

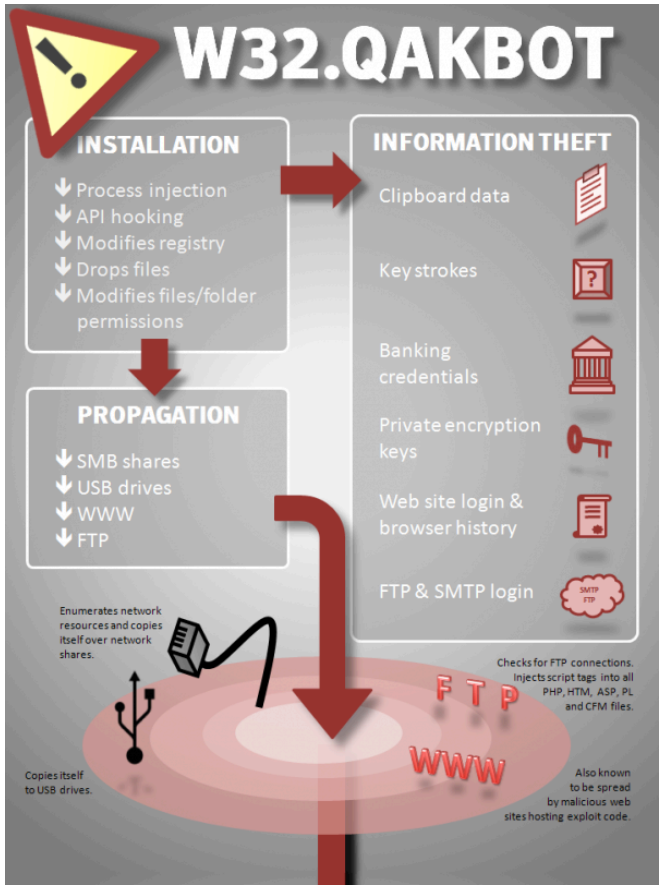
- Configuration file of W32.Qakbot
- A.k.a. “.cb” file
- Simple encrypted

encrypted.bin

00000000	5B29	1D7C	494F	0DEC
00000008	5B39	617C	4A23	74E6
00000010	1439	0914	0729	2D5C
00000018	5423	68DC	54AD	2914
00000020	56E5	3D14	5333	758D
00000028	6513	310C	4EAD	3564
00000030	0729	2D5C	5423	0054
00000038	5629	907C	5623	3DEC
00000040	654F	7D64	5519	1D04
00000048	50AD	2DB4	5F8B	095C
00000050	5F2D	1DD6	143B	595C
00000058	3033	093C	5B17	C58D
00000060	4B35	05D4	1439	0914
00000068	0729	2D5C	5423	6806
00000070	5E29	0927	5B29	1D7C
00000078	494F	C5FC	582F	6904
00000080	4E23	005C	423B	4C14
00000088	5F3B	013C	0E1D	2D06
00000090	5F01	2D27	5B29	1D7C
00000098	494F	755C	5929	054C
000000A0	1419	59D4	0729	2D5C

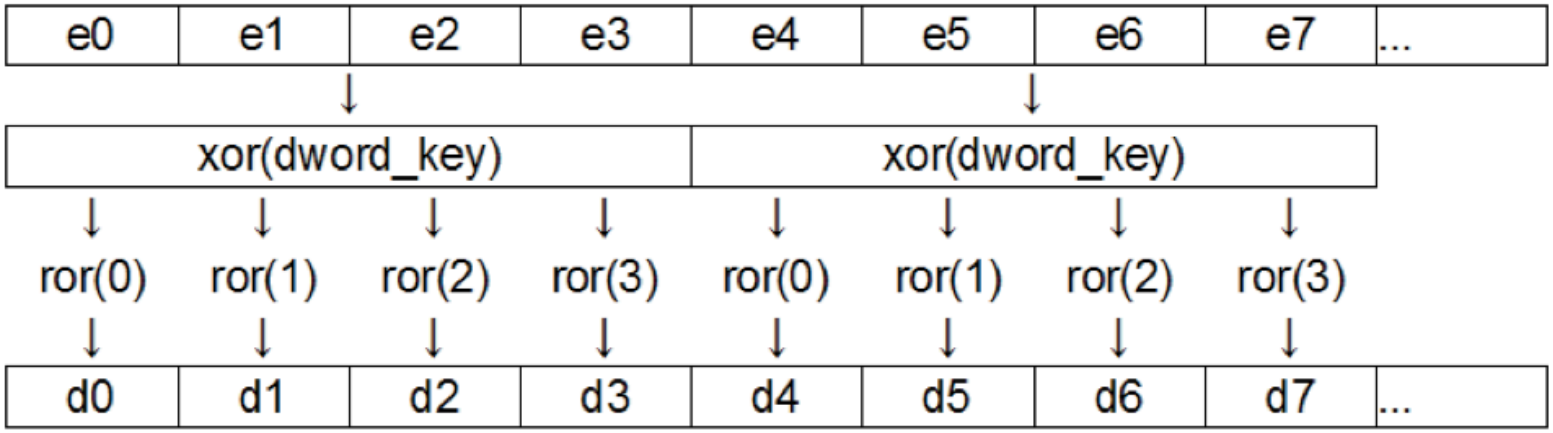
decrypted.bin

00000000	616C	6961	735F	6D73
00000008	6164	7661	7069	3332
00000010	2E64	6C6C	3D6C	6565
00000018	6E69	3475	6E2E	646C
00000020	6C0A	616C	6961	735F
00000028	5F71	626F	742E	6362
00000030	3D6C	6565	6E69	2E64
00000038	6C6C	0A61	6C69	6173
00000040	5F5F	7162	6F74	696E
00000048	6A2E	6578	653D	6C65
00000050	656E	6934	2E65	7865
00000058	0A61	6C69	6173	5F5F
00000060	7162	6F74	2E64	6C6C
00000068	3D6C	6565	6E69	342E
00000070	646C	6C0A	616C	6961
00000078	735F	5F71	626F	746E
00000080	7469	2E65	7865	3D6C
00000088	6565	6E69	3476	652E
00000090	6578	650A	616C	6961
00000098	735F	7365	636C	6F67
000000A0	2E74	7874	3D6C	6565



W32.Qakbot!conf (2)

- The encryption is combination of xor and rotation right
- Block is dword



W32.Qakbot!conf (3)

- Decrypted data

  alias__qbot.cb=eaoiot.dll
  alias__qbotinj.exe=eaoioto.exe
  alias__qbot.dll=eaoioto.dll
  alias__qbotnti.exe=eaoiotouo.exe

- How to decrypt

  1. Try to decrypt some bytes by each dword key.
  2. Search for keywords such as ‘qbot’ to confirm whether the key is correct or not.

W32.Qakbot!conf (4)

- host code is almost same as Generic keyword search
- Kernel code

```
for (int dwordKey = startValue; dwordKey < endValue; ++dwordKey) {
        for (int i = 0; i < searchLength; ++i) {
                int j;
                for (j = 0; j < keywordLength; ++j) {
                        uchar x = (dwordKey >> (8 * (j % 4))) & 0xFF;
                        if (dec(dataLocal[i + j], x, j % 4) != keywordLocal[j]) {
                                break;
                        }
                }
                if (j == keywordLength) {
                        result[0] = dwordKey;
                        return;
                }
        }
}
```

W32.Qakbot!conf (5)

- kernel code (OpenCL C)
  - 52 instructions
- host code (Python)
  - 76 instructions

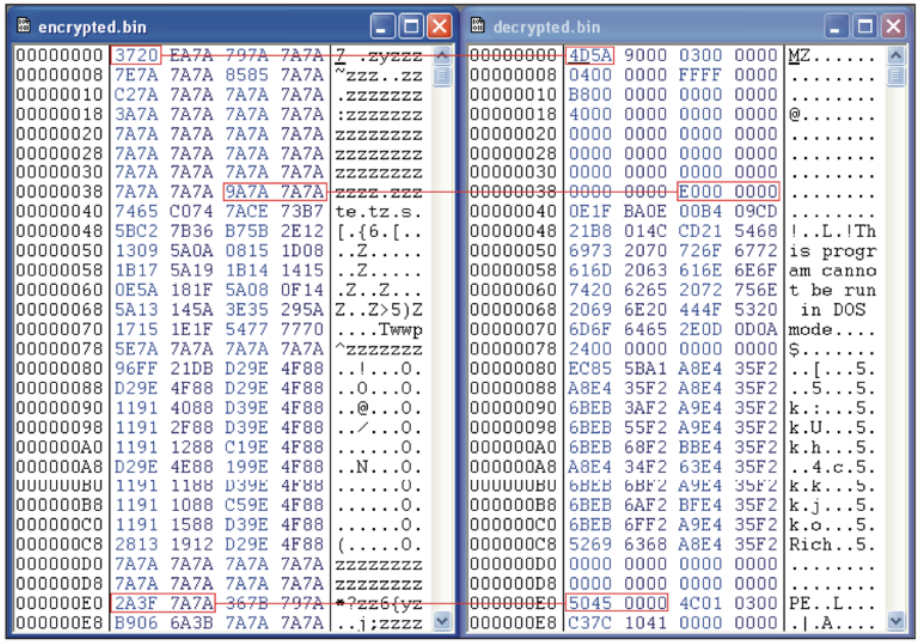
W32.Qakbot infection

Finding Hidden PE Files (1)

- Find hidden MZ header and PE header
- MZ = 4D, 5A

  1.xor(0x37, 0x4D:'M') = 0x7A

  2.xor(0x20, 0x7A) = 0x5A:'Z'



Finding Hidden PE Files (2)

- Listing 4 in the Appendix is the kernel code for finding a hidden PE file

```
WORD key = decrypt_w(function, *((WORD*)(data + offset)), SIG_MZ);
if (offset + 0x3C >= dataSize) {
        return FALSE;
}
WORD peOffset = decrypt_w(function, *((WORD*)(data + offset + 0x3C)), key);
if (offset + peOffset >= dataSize) {
        return FALSE;
}
if (SIG_PE != decrypt_w(function, *((WORD*)(data + offset + peOffset)), key)) {
        return FALSE;
}
if (0x00 != decrypt_w(function, *((WORD*)(data + offset + peOffset + 2)), key)) {
        return FALSE;
}
resultValue[offset] = key;
return TRUE;
```

Finding Hidden PE Files (3)

- kernel code (OpenCL C)
  - 220 instructions
- host code (Python)
  - 97 instructions

Agenda

1. Introduction
2. Background
3. Actual Tools For Threat Analysis
4. Pros, Cons And Performance Of OpenCL Coding
5. Further Implementation of PyOpenCL
6. Conclusion

Pros (1)

- Follows a write once, deploy everywhere philosophy
- Supported hardware is widely available and reasonably priced
- Many products support OpenCL – GPU, CPU and specialized devices such as NVIDIA Tesla

W32.Qakbot infection

Pros (2)

- Helps in vector calculation
- Provides additional power by adding GPU to the machine
- Allows users to use multiple devices at the same time

Cons (1)

- Requires an understanding of how to divide the target data or the whole process to fit parallel computing
- Non standard development tools with different learning curves
- The OpenCL device endian is not always the same

Cons (2)

- In some cases, pointer-cast is not feasible

- With the NVIDIA device, the parallel calculation cannot be ended until five seconds have elapsed

Performance (1)

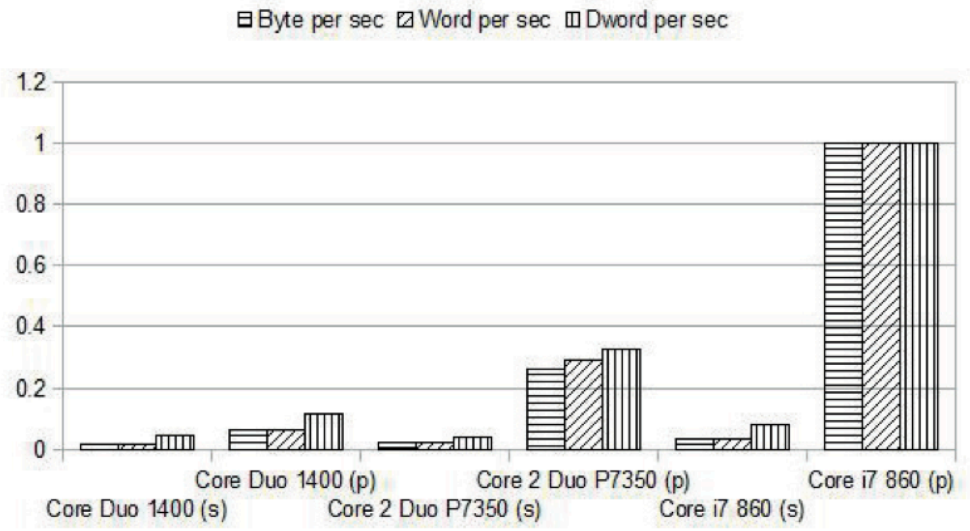
- Details of the devices used to evaluate the performance

Device	Platform	OS	Core	Freq	Release date
Core Duo 1400	iMac (Early 2006)	Mac OS X	2	1.83GHz	2006/01
Core 2 Duo P7350	MacBook (Early 2009)	Mac OS X	2	2.0GHz	2009/01
Core i7 860	iMac (Late 2009)	Mac OS X	8	2.8GHz	2009/10
Geforce 9400M	MacBook (Early 2009)	Mac OS X	16	450MHz	2009/01
Radeon HD 4850	iMac (Late 2009)	Mac OS X	800	625MHz	2009/10
Radeon HD 6950	DELL PRECISION T1500	Linux	1408	800MHz	2010/12

Performance (2)

- The results for traditional serial processing compared with parallel processing

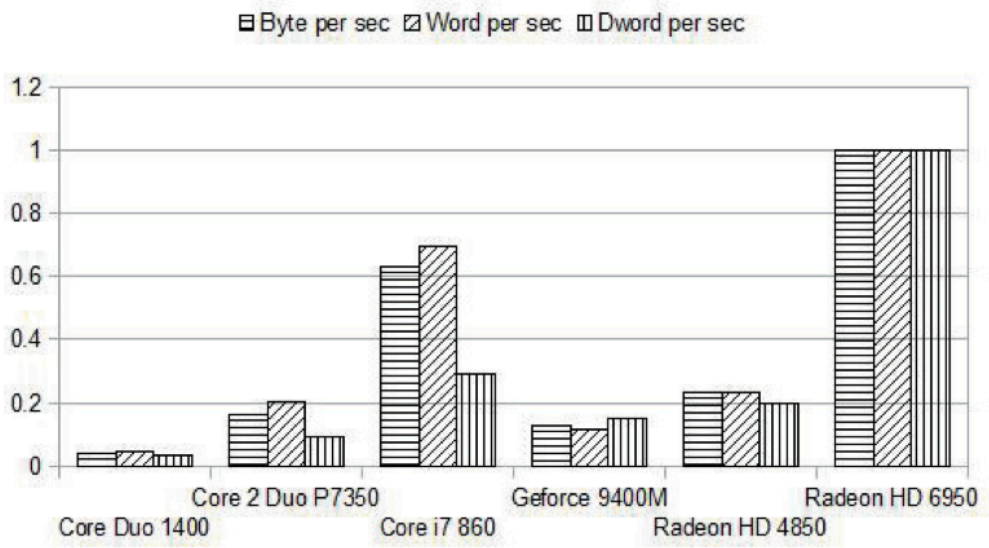
Device	Bytes/sec	Words/sec	Dwords/sec
Core Duo 1400 (s)	22942	93.7	0.0017
Core Duo 1400 (p)	85590	377.9	0.0044
Core 2 Duo P7350 (s)	27829	117.9	0.0016
Core 2 Duo P7350 (p)	344900	1725.5	0.0126
Core i7 860 (s)	48607	201.6	0.0031
Core i7 860 (p)	1321000	5911.2	0.0388



Performance (3)

- The parallel computing performance

Device	Bytes/sec	Words/sec	Dwords/sec
Core Duo 1400	85590	377.9	0.0044
Core 2 Duo P7350	344900	1725.5	0.0126
Core i7 860	1321000	5911.2	0.0388
Geforce 9400M	265500	975.9	0.0201
Radeon HD 4850	487000	1971.2	0.0265
Radeon HD 6950	2095700	8478.2	0.1321



Agenda

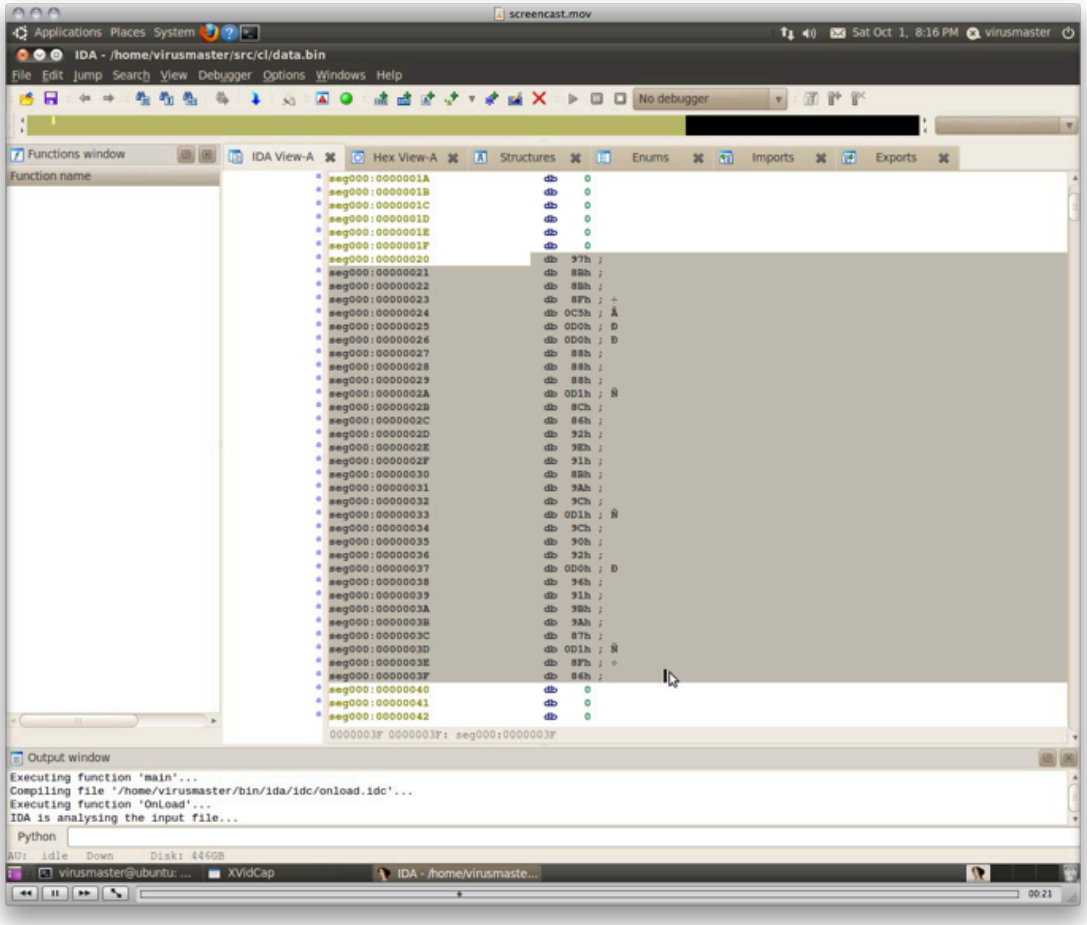
1. Introduction
2. Background
3. Actual Tools For Threat Analysis
4. Pros, Cons And Performance Of OpenCL Coding
5. Further Implementation of PyOpenCL
6. Conclusion

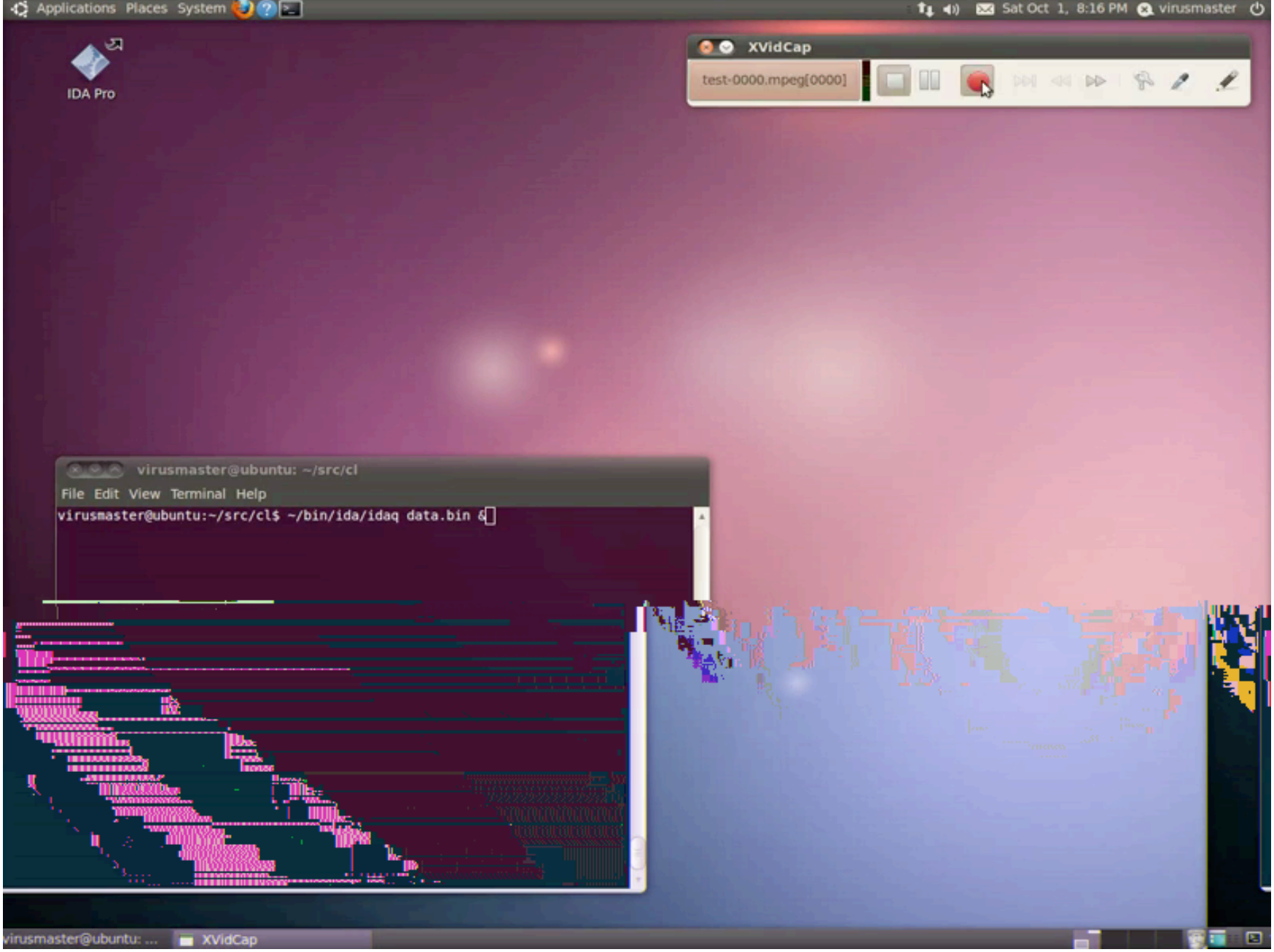
IDA Pro

- IDAPython and PyOpenCL
- Set-up
  - Ubuntu Linux Desktop 10.10 32-bit
  - IDA Pro 6.0 Standard for Linux
  - Python 2.6.6 (installed by default)

IDA Pro (2)

- Generic keyword search in IDAPython




Applications Places System
Sat Oct 1, 8:16 PM virusmaster
IDA Pro
XVidCap
test-0000.mpeg[0000]
virusmaster@ubuntu: ~/src/cl
File Edit View Terminal Help
virusmaster@ubuntu:~/src/cl$ ~/bin/ida/idaq data.bin &
virusmaster@ubuntu: ...
XVidCap

Other Python applications

- Wireshark can extend its function by using Python.
- PyIDS is IDS made by Python so it should be easy to implement using PyOpenCL.
- Immunity Debugger can be extended using Python.

Agenda

1. Introduction
2. Background
3. Actual Tools For Threat Analysis
4. Pros, Cons And Performance Of OpenCL Coding
5. Further Implementation of PyOpenCL
6. Conclusion

Conclusion

- By introducing 3 simple tools and advanced IDA Pro use, I showed brute-forcing is useful for threat analysis.
- Writing a parallel computing program is not always easy
- OpenCL can be the standard and one code can be created for all devices that support OpenCL
- PyOpenCL is the OpenCL interface for Python, and it allows easier creation of OpenCL programs
- The usage of GPGPU and OpenCL will increase and eventually become ubiquitous
- How to use for threat analysis depends on you!

Thank you!

Takashi Katsuki
takashi_katsuki@symantec.com