Security threats from hardware

Chao Liao
Outline

- Background
- Analysis of Pros & Cons
- GPU assisted malware
- Other techniques
- Conclusion
Motivation

- Security threats are not only from software level anymore.

**CIH**, the famous malware could damage hardware.

**Stuxnet**: aims for critical industrial infrastructure with a programmable logic controller (PLC) rootkit.

Current most viruses utilize vulnerabilities in software.

? What happen if they focus on the counterpart--hardware?

? What happen if they utilize hardware to implement its malicious purpose?
New trend of today’s malware

With the disappearance of an OS monoculture, attackers would do well to find attacks that are neither OS or application specific.

source: left [http://marketshare.hitslink.com](http://marketshare.hitslink.com)
right Gartner [www.gartner.com/it/page.jsp?id=1421013](http://www.gartner.com/it/page.jsp?id=1421013)
New trend of today’s malware

- One way to do that, is to target attacks at hardware, rather than software.

CPU share by 2010 Q3
- 80.40%
- 19.20%
- 0.40%

Discrete Graphics Market Share
- 49.00%
- 61.00%

Source: http://news.softpedia.com/news
Pros & Cons

✓ High fixing (removing) cost, no easy patch unless replace the hardware.
✓ Hard to detect at the first time.

X Limited infection targets – Heavily hardware specified
Ways of Exploiting

Hardware specified exploiting

Backdoor in hardware
- Keyboard recording
- ...

Hardware vulnerability exploiting
- Implement Rootkit in NIC
- CIH
- ...

Hardware assisted Malware
- Exploit with CPU
- Exploit with GPU
- ...
Backdoor in the hardware

- Samuel T. King and et al implemented two general purpose mechanisms for designing malicious processors, and used them to implement attacks that steal passwords, enable privilege escalation, and allow automatic logins into compromised systems.
Guillaume Delugre demonstrates how to embed a rootkit into NetExtreme firmware which could:

- Take over the network
  - Packet interception/forge by the rmware
  - Embedding an IP/UDP stack and a light DHCP client
  - Stealthy communication (OS never aware)

- Corrupt physical memory
  - Reuse DMA capabilities over PCI to corrupt system RAM

- No trace on the operating system, as it is being hidden inside the NIC.
GPU-Assisted Malware

- Using Compute Unified Device Architecture (CUDA) / FireStream to implement GPU code.
- OpenCL could fill the gap between Nvidia and AMD.
- OpenGL and DirectX is not suitable for general-purpose computation programming.
GPU-Assisted Malware

• Malware can be statically linked with the CUDA library into a single stand-alone executable, without the need to install any additional files on the infected system.

• The execution of GPU code, as well as data transfers between the host and the device do not require any administrator privileges. The malware will run successfully even under user privileges!
Self-unpacking
Figure 2. The intermediate PTX code of a simple XOR-based unpacking function for NVIDIA graphics cards.
Self-unpacking

Modern GPUs are powerful enough to pack the malware with EXTREMELY complex encryption schemes due to the massively parallel architecture of GPUs.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Throughput</th>
<th>CPU Throughput</th>
<th>Relative Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blowfish</td>
<td>3763 MB/s</td>
<td>91 MB/S</td>
<td>41.7x</td>
</tr>
<tr>
<td>AES</td>
<td>4219 MB/s</td>
<td>114 MB/s</td>
<td>37x</td>
</tr>
<tr>
<td>Secure-AES</td>
<td>3171 MB/s</td>
<td>N/A</td>
<td>28x(Vs. Normal AES on CPU)</td>
</tr>
</tbody>
</table>

Table 2: Throughput of the encryption schemes on the GPU Vs. CPU

From Rishabh Mukherjee and et al, Presenting new Speed records and constant time encryption on the GPU
How to detect

In contrast to x86 code, analysis of GPU machine code is at a nascent stage, and it is currently not supported by existing malware analysis systems.

Existing dynamic malware analysis systems built on top of VMs, like Renovo, provide only simulated graphics devices which currently do not support any GPGPU functionality.
How to detect

The only CPU code that is exposed in the original malware image is init section that copy the packed data to the newly allocated buffer and bootstrap the execution of the unpacking routine on the GPU. This minimal x86 code footprint does not leave much to existing static and dynamic malicious code analysis systems to actually analyze.
Polymorphism

Polymorphic code is code that uses a polymorphic engine to mutate while keeping the original algorithm intact. That is, the code is re-encrypted using a different randomly generated key, causing the malware to constantly mutate in unpredictable ways in the host’s memory.

Figure 4. Schematic representation of the execution of GPU-assisted malware that employs runtime polymorphism.
How to detect

- The encrypted code of each function is stored in memory segments that are accessible from both the CPU and the GPU.
- In contrast, the decryption keys are stored in private device memory that is not accessible from the CPU. This effectively hinders existing analysis methods that extract the keys and decrypt all encrypted code blocks using runtime instrumentation.
Other ways?

*Framebuffer*, which contains what is displayed on the monitor.

Malicious code running on the GPU could access the screen buffer periodically and harvest private data displayed on the user screen, and do so in a more stealthy way than existing screen capture methods.
Other ways?

More sophisticated malware could attempt to trick users by displaying false, benign-looking information when visiting rogue web sites (e.g., overwriting suspicious URLs with benign-looking ones in the browser’s address bar)
Other ways?

GPUs offer massive parallelism, which can be used to speed up CPU-intensive operations. For example, a botnet can be set up for large-scale password cracking—a task that GPGPUs excel in. Bots can easily be extended with GPGPU support and then use the GPUs of infected hosts to offload password cracking.
Super powered malware

• Besides 3D games, what about SLI / Crossfire powered malware?
Solutions

• For Backdoor in hardware
  – Build your own foundry!

The DOD also maintained its own chip-making plant at Fort Meade, near Washington, D.C., until the early 1980s, when costs became prohibitive.

• Hardware vulnerability exploiting
  – Update hardware

• Hardware assisted malware
  – Develop anti-virus software.
How to identify particular hardware

- To exploit hardware specificities, we need to identify particular hardware.

- Simple? How does a malware identify current CPU?

- The Intel Assembly Language instruction CPUID can be used, but it has at least two severe drawbacks:
  - It is easy to “find” it whenever scanning the file (malware detection issue);
  - Some other processors cannot recognize and process this instruction.
How to identify particular hardware

Algorithm 1: The $\sqrt{}$ problem
Input: — a real $A$;
Output: — a boolean $B$
Begin:
   $B = \sqrt{A} \cdot \sqrt{A};$
   Return($A == B$);
End.

1. Mathematically: True is returned;
2. Practically: False is returned!

Exploiting IEEE P754 standard
How to identify particular hardware

<table>
<thead>
<tr>
<th>Processor</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.2-0.8 == 0.4</td>
</tr>
<tr>
<td>VAX 750</td>
<td>Yes</td>
</tr>
<tr>
<td>AMD 32</td>
<td>No</td>
</tr>
<tr>
<td>AMD 64</td>
<td>No</td>
</tr>
<tr>
<td>ATOM</td>
<td>No</td>
</tr>
<tr>
<td>INTEL DC</td>
<td>No</td>
</tr>
<tr>
<td>MIPS 12000</td>
<td>No</td>
</tr>
<tr>
<td>dsPIC33FJ21</td>
<td>No</td>
</tr>
<tr>
<td>IPHONE 3G</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 4. A few easy computations

<table>
<thead>
<tr>
<th>Processor</th>
<th>$\sin(10^{10} \pi_1)$</th>
<th>$\sin(10^{17} \pi_1)$</th>
<th>$\sin(10^{37} \pi_1)$</th>
<th>$\sin(10^{17} \pi_1) == \sin(10^{17} \pi_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPHONE 3G</td>
<td>0.375...</td>
<td>0.423...</td>
<td>-0.837...</td>
<td>No</td>
</tr>
<tr>
<td>AMD 32</td>
<td>0.375...</td>
<td>0.424...</td>
<td>-0.837...</td>
<td>No</td>
</tr>
<tr>
<td>AMD 64</td>
<td>0.375...</td>
<td>0.424...</td>
<td>0.837...</td>
<td>No</td>
</tr>
<tr>
<td>ATOM</td>
<td>0.375...</td>
<td>0.423...</td>
<td>-0.832...</td>
<td>No</td>
</tr>
<tr>
<td>INTEL DC</td>
<td>0.375...</td>
<td>0.423...</td>
<td>-0.832...</td>
<td>No</td>
</tr>
<tr>
<td>MIPS 12000</td>
<td>0.375...</td>
<td>0.423...</td>
<td>-0.832...</td>
<td>No</td>
</tr>
<tr>
<td>dsPIC33</td>
<td>0.81...</td>
<td>0.62...</td>
<td>-0.44...</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5. Computation of $\sin(10^{10} \pi)$ for various numerical values of the constant $\pi$
Conclusion

• Why we need to care about threats from hardware.
  – It is hard to detect and currently no effective detection mechanism exists

• Three ways to utilize hardware for malicious purposes:
  – Backdoor in hardware
  – Hardware vulnerability exploiting
  – Hardware assisted Malware

• Finally, we briefly discuss how to identify particular hardware
References


4. Samuel T. King, et al, Designing and implementing malicious hardware, ProceedingLEET'08 Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats
Any Questions?

Thanks!