Secure and Efficient Application Monitoring and Replication

Stijn Volckaert, Bart Coppens, Alexios Voulimeanias, Andrei Homescu, Per Larsen, Bjorn De Sutter, Michael Franz
Adobe Patches 10 Critical Vulnerabilities in Flash Player, Shockwave Player, and ColdFusion

Posted on April 9th, 2013 by Derek Erwin

Adobe Issues Emergency Updates For Zero-Day Flaw in Flash Player

Memory corruption flaw is being exploited in the wild to distribute ransomware samples like Locky and Cerber.

Firefox 45 browser update patches 22 (Another) Update To Adobe Flash Addresses Latest 0-Day Vulnerability

April 15, 2011
By Charlie Osborne for Zero Day | March 10, 2018 -- 10/06 GMT (02:01 PST) | Topic: Security

Update Flash now! Adobe releases patch, fixing critical security holes

BY GRAHAM CLULEY POSTED 22 SEP 2015 - 09:24AM

Critical Adobe Flash bug under active attack currently has no patch
Exploit works against the most recent version; Adobe plans update later this week.

by Dan Goodin - Jun 14, 2016 12:50pm PDT

Adobe Releases Security Update for 19 ‘Critical’ Vulnerabilities in Flash Player

DAVID BISSON
DEC 26, 2015 | LATEST SECURITY NEWS
Possible Solutions

- Type-Safe Languages (e.g. Rust)

- Mitigations:
  - Integrity-Based (e.g. CFI)
  - Randomization-Based (e.g. ASLR)

- Multi-Variant Execution Environments (MVEEs)
Memory Corruption Attacks

0: `void foo() {
1:   char buf[256];
2:   gets(buf);
3:   printf("%s", buf);
4: }
`

0: `int main(int argc, char** argv) {
1:   foo();
2:   return 0;
3: }
`
Multi-Variant Execution Environments (MVEEs)

In a nutshell:

- Run multiple program variants in parallel
- Variant system calls executed in lock-step
- Suspend them at every system call
- Compare system call numbers/arguments
- Master/Slave replication for I/O
Performance Considerations

Programs can execute at **native speed** (assuming you have enough idle CPU cores and memory bandwidth)

**BUT** system call interception is **SLOW!**
Alternative Design

Efficient Monitoring:

- Load monitor into variants’ address spaces
- Replicate results through a shared buffer
- Let master run ahead of slaves

BUT:

- Malicious syscalls can circumvent monitor
- Shared buffer data can be tampered with
Split-Monitor Design:

- Handle security-sensitive system calls in Cross-Process Monitor (CP-MON)
- Handle non-sensitive system calls in In-Process Monitor (IP-MON)
- Configurable relaxation policies
Relaxation Policies

• 3 different policies:
  • Syscalls unconditionally handled by IP-MON (e.g. sys_getpid)
  • Syscalls conditionally handled by IP-MON (e.g. sys_write for non-socket files)
  • Syscalls probabilistically handled by IP-MON (not implemented)
Initializing the In-Process Monitor

Registering IP-MON:

- Call `sys_prctl` with list of non-security-sensitive system calls as argument
- This `sys_prctl` call will **ALWAYS** be reported to CP-MON
- If the call succeeds, all of the syscalls in the list will be forwarded to IP-MON from that point onward

```
prctl(PR_REGISTER_IPMON, <list of system calls>)
```
ReMon Components

System Call Broker:

- Intercept system calls as they enter kernel
- Forward them to appropriate monitor based on active relaxation policy
- Authenticate system calls when resumed by monitor
In-Process Monitor:

- Authorized to execute forwarded calls w/o intervention by cross-process monitor
- Replicates system call results through shared buffer
ReMon Components

Cross-Process Monitor:
- Standard **ptrace-based** monitor
- Completely isolated from variants
No abuse of monitor privileges:

- Monitor cannot execute system calls that did not pass through the broker first
- Broker generates **authentication key** and loads it into register when forwarding call to monitor
- Key must be intact to finish execution of forwarded call
In-Process Monitor Security

No tampering with monitor data:

- Locations of monitor and shared buffer are only known to broker
- Pointers to monitor and buffer are never visible in user space
In-Process Monitor Security

Leak Prevention:

- Sensitive values (e.g. pointers, authorization key) only stored in registers and never leaked or spilled
- Monitor has no indirect branches => control flow cannot be diverted to malicious code
Performance

• Dual Intel Xeon E5-2660 – 20Mb Cache each
• 64Gb ECC DDR3 RAM
• Linux 3.13.11
• Server Benchmarks:
  • Local loopback
  • Gbit Link – 2ms
  • Gbit Link – 5ms
• 2 variants of the protected program
• 4 worker threads for multi-threaded benchmark suites (PARSEC/SPLASH-2x)
## Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CP-MON only</th>
<th>ReMon</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEC CPU 2006</td>
<td>6.37%</td>
<td>3%</td>
</tr>
<tr>
<td>PARSEC 2.1</td>
<td>22%</td>
<td>11%</td>
</tr>
<tr>
<td>SPLASH-2x</td>
<td>29%</td>
<td>10%</td>
</tr>
<tr>
<td>Server Benchmarks</td>
<td>up to 1249%</td>
<td>&lt;3.5%</td>
</tr>
</tbody>
</table>
Conclusions

• Existing Security-Oriented MVEEs:
  • Secure but SLOW

• Existing Reliability-Oriented MVEEs:
  • Fast but INSECURE

• ReMon:
  • FAST and SECURE

https://github.com/stijn-volckaert/ReMon