BlackBox: A Runtime Approach to Security Auditing

Byron Hawkins and Brian Demsky, University of California, Irvine, USA · Michael B. Taylor, University of California, San Diego, USA

Overview

Goals of BlackBox

1. Log the pivotal control flow events of a successful exploit.

Log sample from an ROP[1] exploit on Adrenalin Media Player:

```
Suspicious entry into dynamically generated code
    adrenalinx.dll(0x16f313) -> DGC(0x3bfffb) raised suspicion
Non-conventional return adrenalinx.dll(0x16f313) -> DGC(0x3bfffb)
Untrusted module calc.exe-1db1446a00060001
Suspicious indirect shlwapi!QISearch -> comctl32!<callback>
Suspicious indirect ntdll!LdrEnumerateLoadedModules(0x86) -> kernel32!<callback>
Untrusted module gdiplus.dll-1db146c800060001
Suspicious indirect kernel32!BaseThreadInitThunk(0xc) -> calc!<callback>
Dynamically generated function owned by adrenalinx.dll-300010001 (4 nodes)
```

2. Block future attempts at exploiting a known program vulnerability.

Blacklist non-conventional returns from an exploitable basic block:

```
edge adrenalinx.dll 0x16f313 <export> 0x3bfffb658587ad92
```

Blacklist non-conventional returns anywhere in a module:

```
node adrenalinx.dll <abnormal-return>
```

Basic Approach

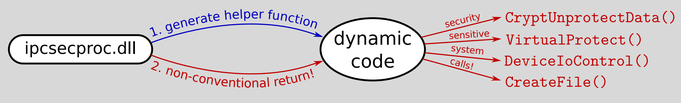
1. Learn normal program behavior during offline profiling.
2. Monitor programs online using binary translation (DynamoRIO).
3. Filter out normal program behavior and highlight unusual behavior.

Why Security Auditing?

Automated security continues to face challenges, e.g. ROP:

1. Stack guards and stack shields can be bypassed.
2. COTS[2] rewriting approaches can be defeated by crafted gadgets.
3. Compilers cannot easily compute all return targets:

A Microsoft media licensing module uses non-conventional returns to enter dynamically generated code during startup of Word.



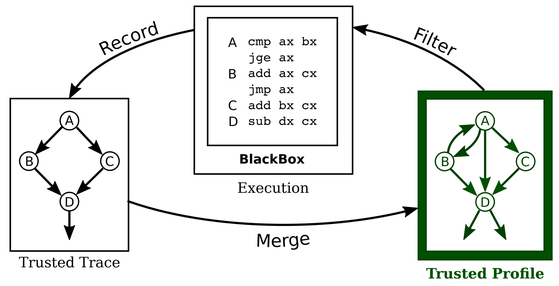
4. Profiling cannot easily identify all non-conventional returns:

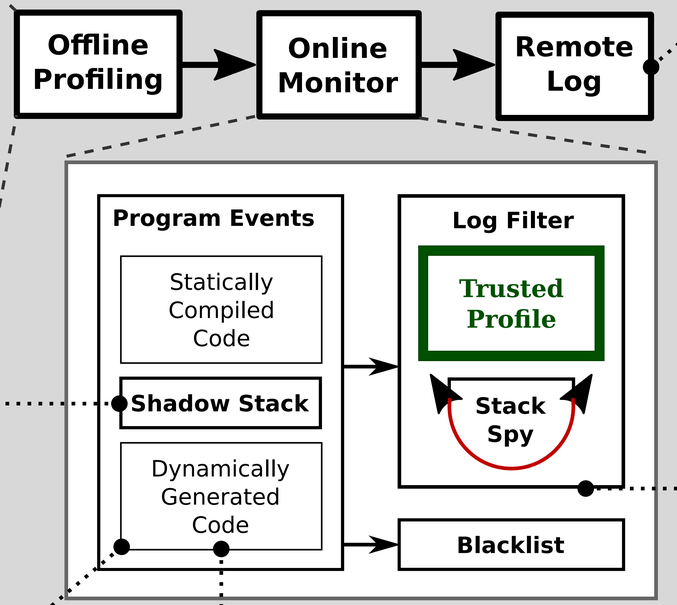
Trusted Profile training continues to encounter new non-conventional returns after opening and scrolling through hundreds of documents.

Application	Total Executions	Executions having non-conventional returns	Last execution having new non-conventional returns
Microsoft Word	1194	57	1156
Microsoft PowerPoint	1914	50	1827
Adobe PDF Reader	6594	52	6587

The Details

Iteratively records normal program events into a Trusted Profile, which becomes a log filter that reduces log content to unusual program behavior.





First-order exploits only affect system calls occurring above those stack frames that contain adversarial control flow edges.

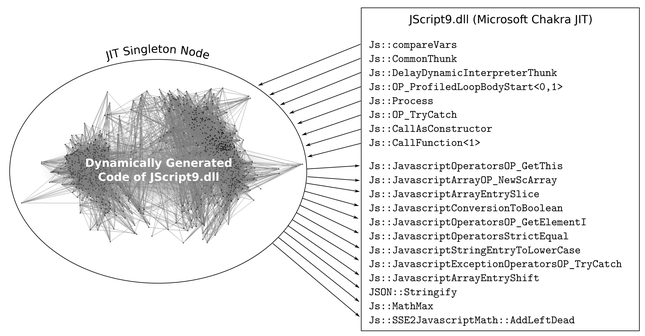
Stack Spy logs system calls that occur on a suspicious call stack, even if the syscall site is trusted.

Higher-order exploits are much more difficult to construct.

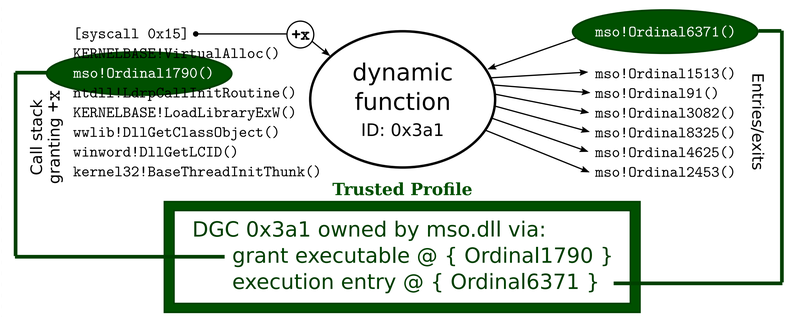
Shadow stack identifies non-conventional returns.

Accommodates windows-specific stack conventions:
- Nested shadow stacks for Windows callbacks.
- Shadow stack suspension for soft context switches.

JIT code varies across executions due to various random factors. BlackBox abstracts JIT monitoring to the API level.



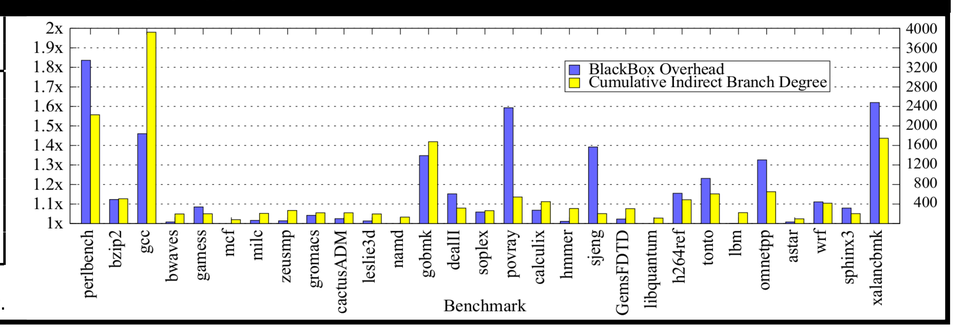
BlackBox monitors code generator behavior to designate a statically compiled module as owner of every dynamic code region; if a region's ownership is untrusted, none of its entry/exit edges will be trusted.



Performance

Program	All Branches (Naive Approach)	Unique Branches (+ Binary Translation)	Forward Indirects (+ Shadow Stack)	Untrusted Indirects (+ Trusted Profile)
Chrome	485,251,278,660	42,957,575	6,137,106	7
Adobe PDF	34,075,711,128	15,579,901	2,292,342	4
Word	603,491,452,236	14,589,337	580,655	24
PowerPoint	251,845,377,624	27,839,593	1,335,817	50
Excel	198,427,776,372	14,810,205	561,401	28
Outlook	547,678,615,056	24,121,810	615,708	4
Adrenalin	48,881,533,212	3,024,797	791,847	603

Total log entries after one hour of normal program use under progressive BlackBox filtering techniques. The BlackBox log focuses concisely on unusual program events.



[1] **R**eturn **O**riented **P**rogramming [2] **C**ommercial **O**ff **T**he **S**helf

BlackBox is supported by the NSF under award 1228992, grants CCF-0846195, CCF-1217854, CNS-1228995, and CCF-1319786; and by the Center for Future Architectures Research (C-FAR).