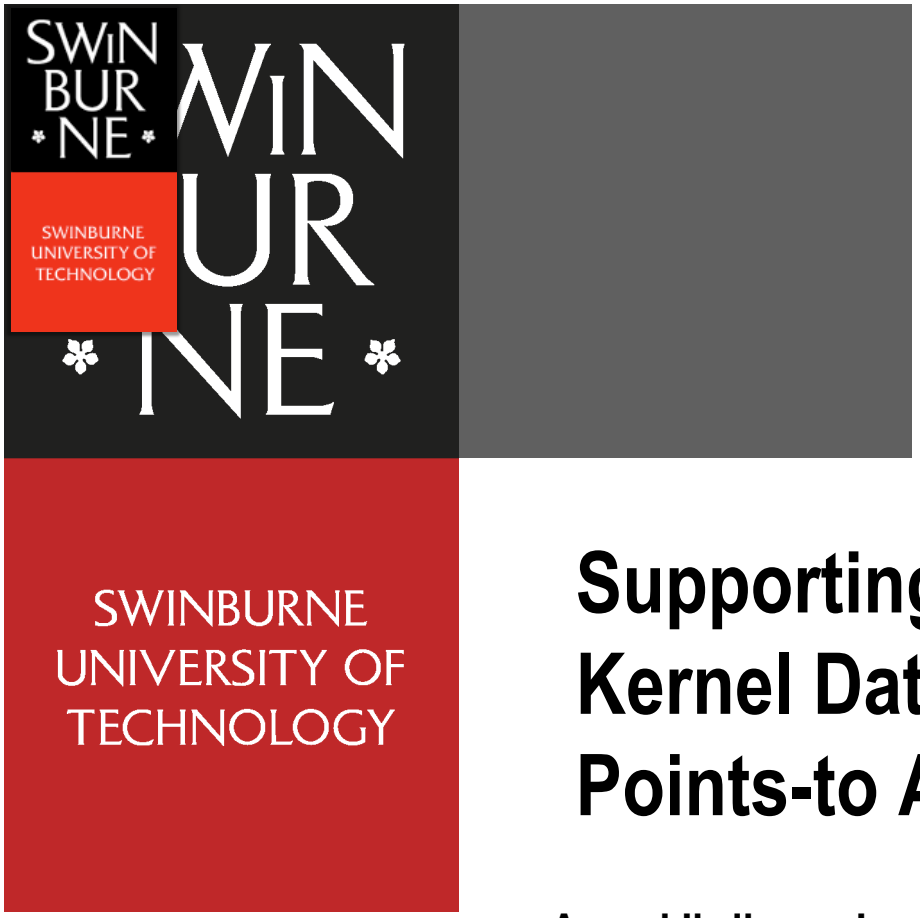


SWINBURNE
UNIVERSITY OF
TECHNOLOGY

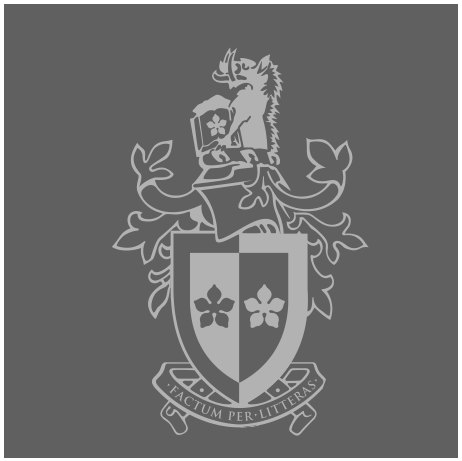
Supporting Operating System Kernel Data Disambiguation using Points-to Analysis

Amani Ibriham, James Hamlyn-Harris, John Grundy & Mohamed Almorsy

Center for Computing and Engineering Software Systems

Swinburne University of Technology

Melbourne, Australia



Outline

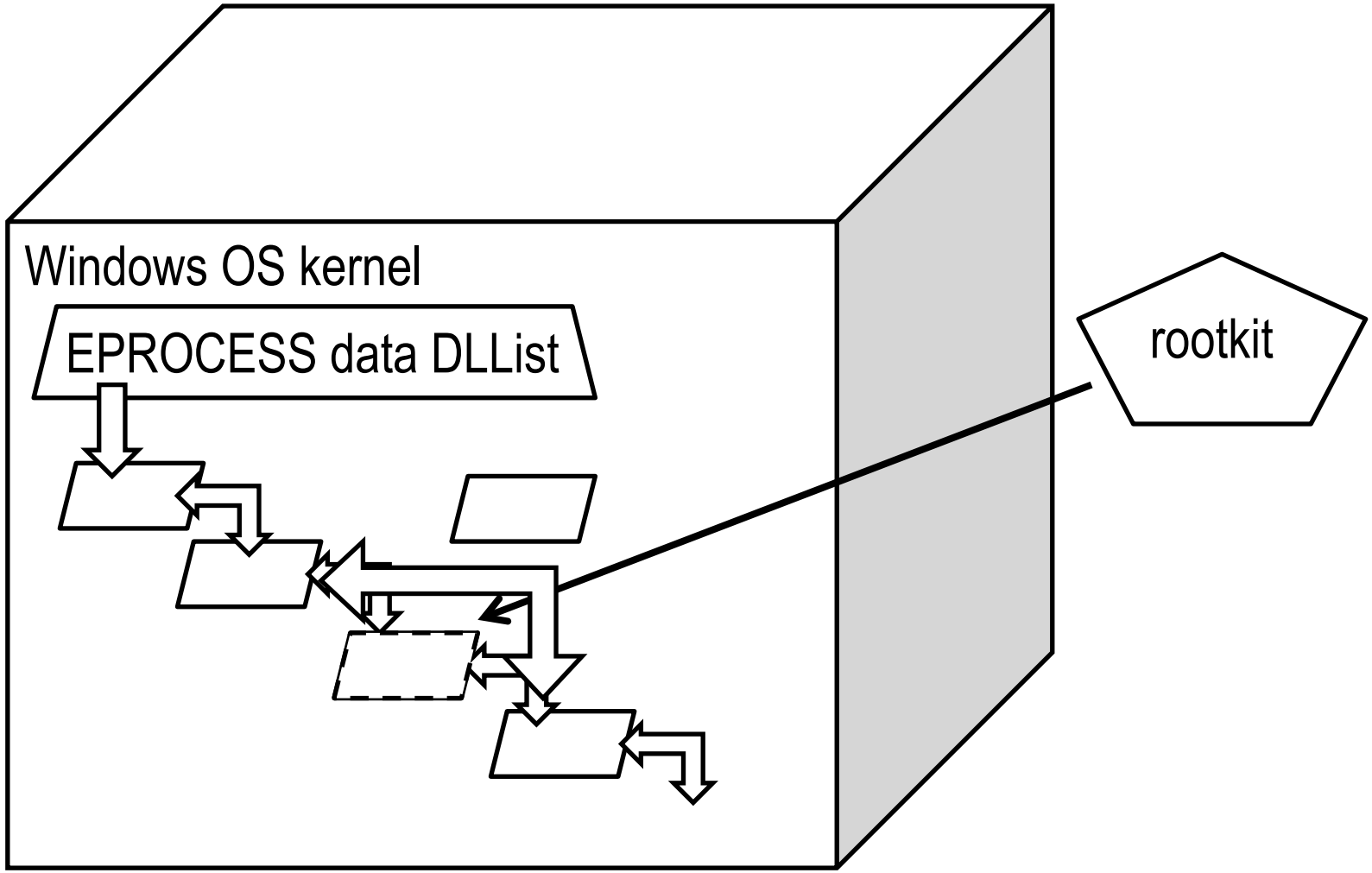
- Problem – C pointer ambiguity
- Operating System Kernel Data
- Our Approach
- Experiments
- Conclusions & Future Research

Problem

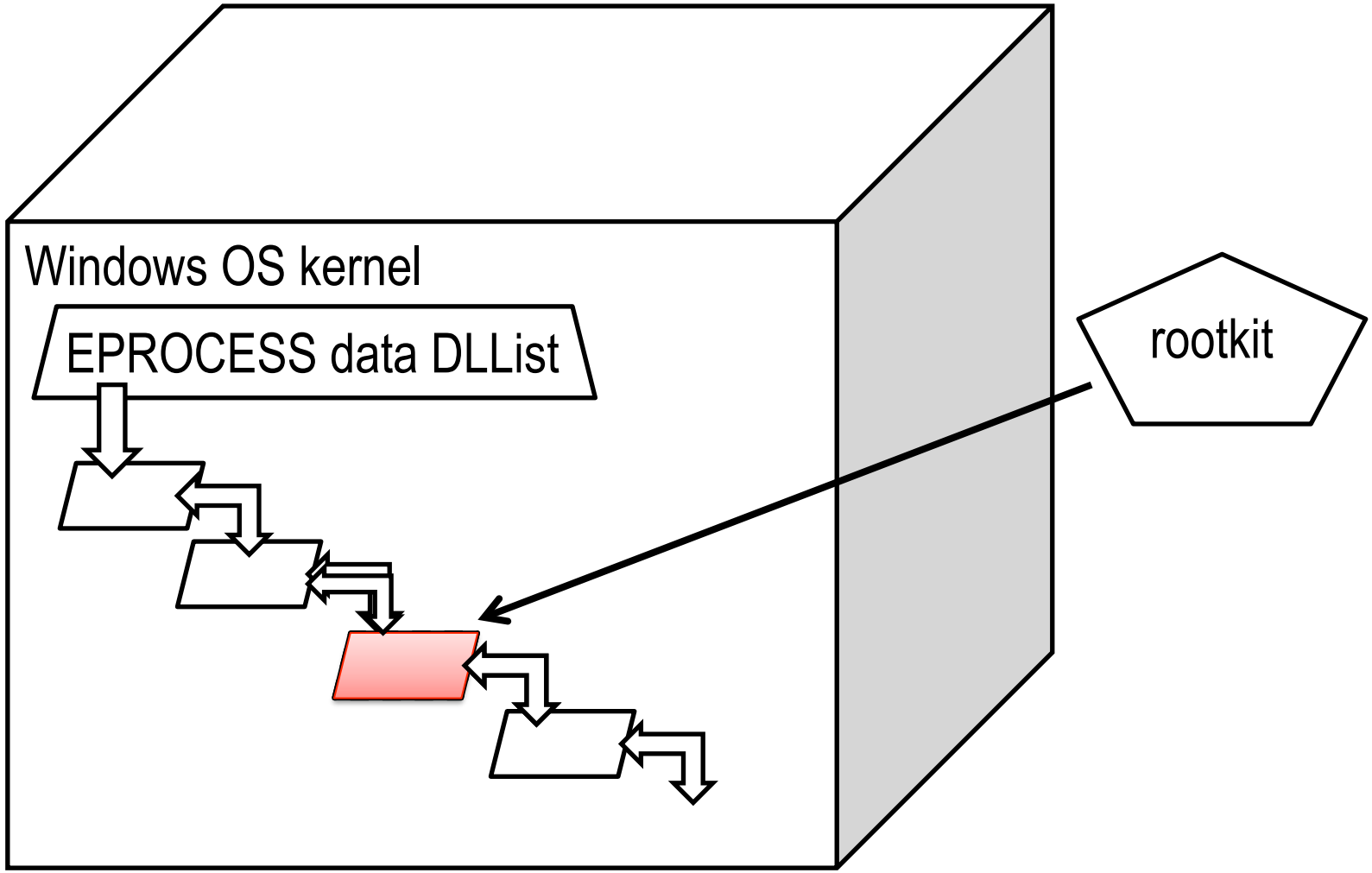
- OS kernel rootkits modify data structures to subvert e.g. retarget processing, access data, hide bad processes etc
- Most OSes are written in C - heavily use C void pointers, null pointers, casting etc to “mimic” objects
- No data structure integrity checking is done by kernel (as its an overhead and not expecting such attacks)
- Running security software in virtualised OS e.g. for Cloud computing is problematic (can be compromised)

=> Serious security holes that need to be addressed

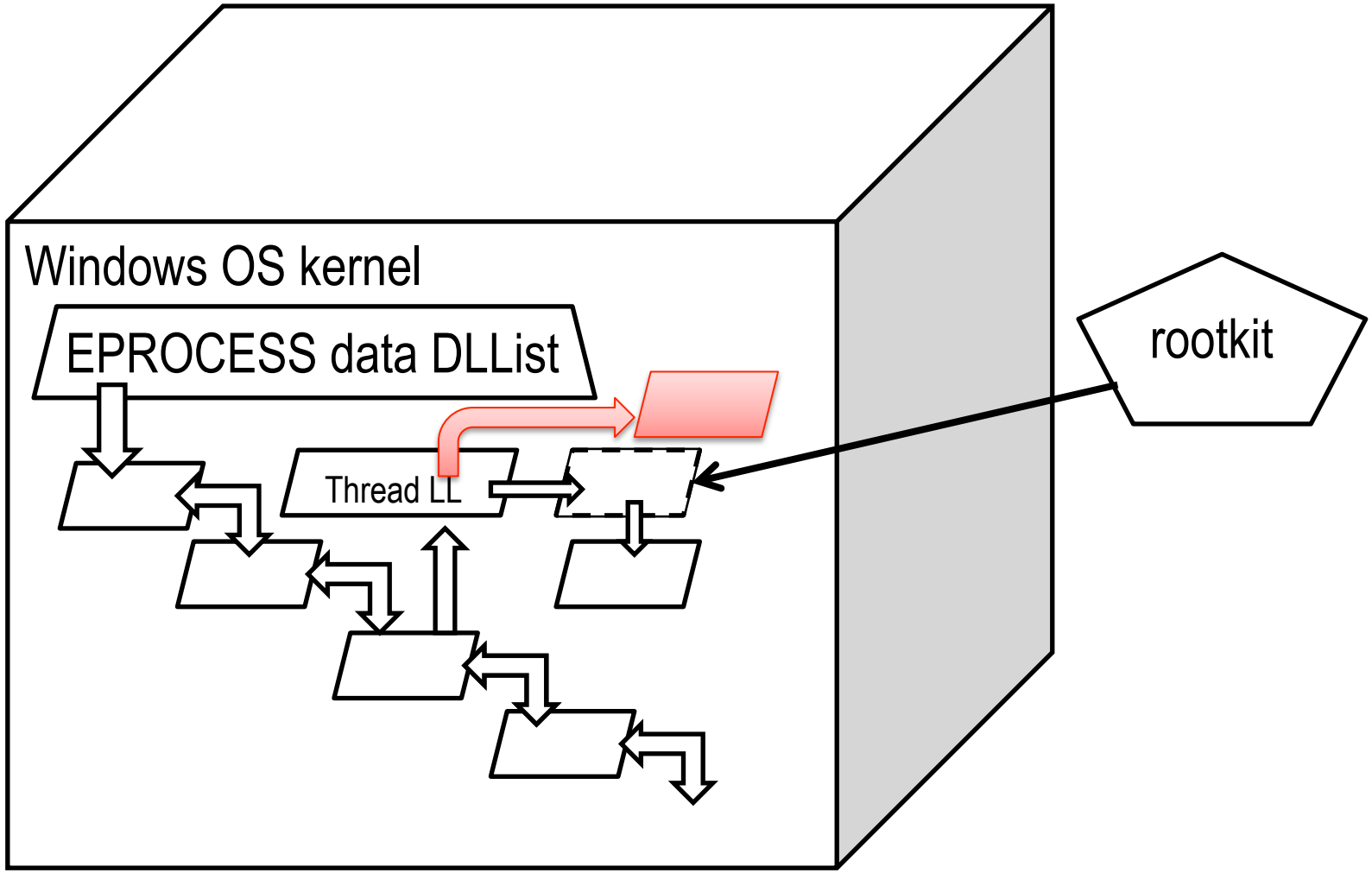
Example 1



Example 2



Example 3



Operating System Kernel Data

- It is very challenging to verify the integrity of OS kernel data
- Kernel Data Complexity:
  - ~ 40% pointer-based relations
  - ~35% of these pointer-based relations are **generic pointers – null, void**
  - OS kernel code also extensively uses C casts
  - OS kernel code is huge:

	TD	GV	Void *	Null *	DL	Uint
Linux	11249	4857	1424	5157	8327	4571
WRK	4747	1858	1691	2345	1316	2587

Example of C code found in OSes

```
typedef struct _LIST_ENTRY {
    struct _LIST_ENTRY *Flink;
    struct _LIST_ENTRY *Blink;
} LIST_ENTRY, *PLIST_ENTRY;
typedef struct _KProcess {
    LIST_ENTRY ThreadHeadList;
} KProcess, *PKProcess;
typedef struct _EPROCESS {
    void* UniqueProcessId; int DebugPort;
    LIST_ENTRY ActiveProcessLinks; KProcess kpc;
} EPROCESS, *PEPROCESS;
typedef struct _EThread {
    void* UniqueThreadId; LIST_ENTRY ActiveThreadLinks;
} EThread, *PEThread;
typedef struct _ExHandle {
    int* handle;
} ExHandle;
LIST_ENTRY PsActiveProcessHead;
PEPROCESS ActiveProcess;
PEPROCESS AllocatePrMemory() {
    return (PEPROCESS) malloc(sizeof(EPROCESS)); }
PEThread AllocateThMemory() {
    return (PEThread) malloc(sizeof(PEThread)); }
void CreateProcess(PEPROCESS p_ptr) {
    p_ptr = AllocatePrMemory();
    ActiveProcess = p_ptr;
    p_ptr->UniqueProcessId = ExHandler();
    p_ptr.DebugPort = (int) ExDebugHandler();
    updatelinks(&ptr->ActiveProcessLinks, &PsActiveProcessHead);
    CreateThread(p_ptr);
    …
}
EThread CreateThread(PEPROCESS p) {
    PEThread th = AllocateThMemory();
    th->UniqueThreadId = ExHandler();
   updatelinks(&th->ActiveThreadLinks,
    p->kpc->ThreadHeadList);
   …
}
void* ExHandler() {
    ExHandle tempHandle;
    tempHandle.handle = CreateHandler();
    …
    return tempHandle.handle;
}
void updatelinks(PLIST_ENTRY src, PLIST_ENTRY tgt) {
     …
    src->Flink = tgt->Flink;
    tgt->Blink = src->Blink;
}
```

UniqueProcessId – void pointer

-analysis of code shows actually points to _ExHandle

_LSIT_ENTRY - null pointer

-calling context gives the type – known only @ run-time

DebugPort – declared Int

…but cast to pointer to function @ run-time

Research Objectives

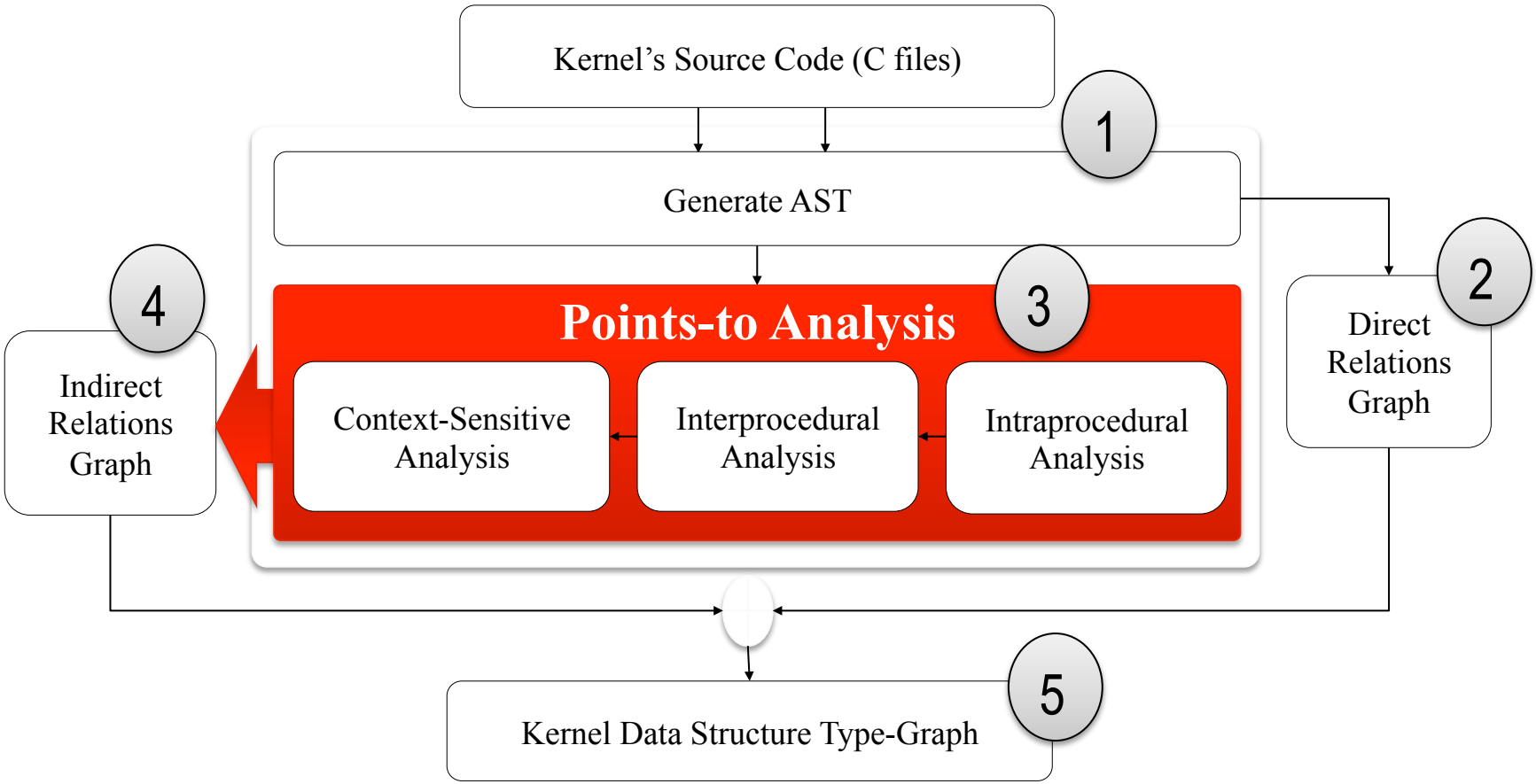
1. **Compute a precise kernel data definition that:**
   - **Precisely models data structures**
   - **Reflects accurate both direct and indirect pointer relations**
2. Discover kernel objects & types at run-time
3. Check kernel data consistency to detect violations
4. Take action e.g. auto-fix, shut down process or VM

- To achieve (1) – we developed a new tool to disambiguate point-based relations in very large C programs (e.g. Windows and LINUX OSes)
- Uses “points-to analysis” – generate type-graph that describes type(s) each pointer can have via deep program analysis

Kernal Data Disambiguator (KDD)

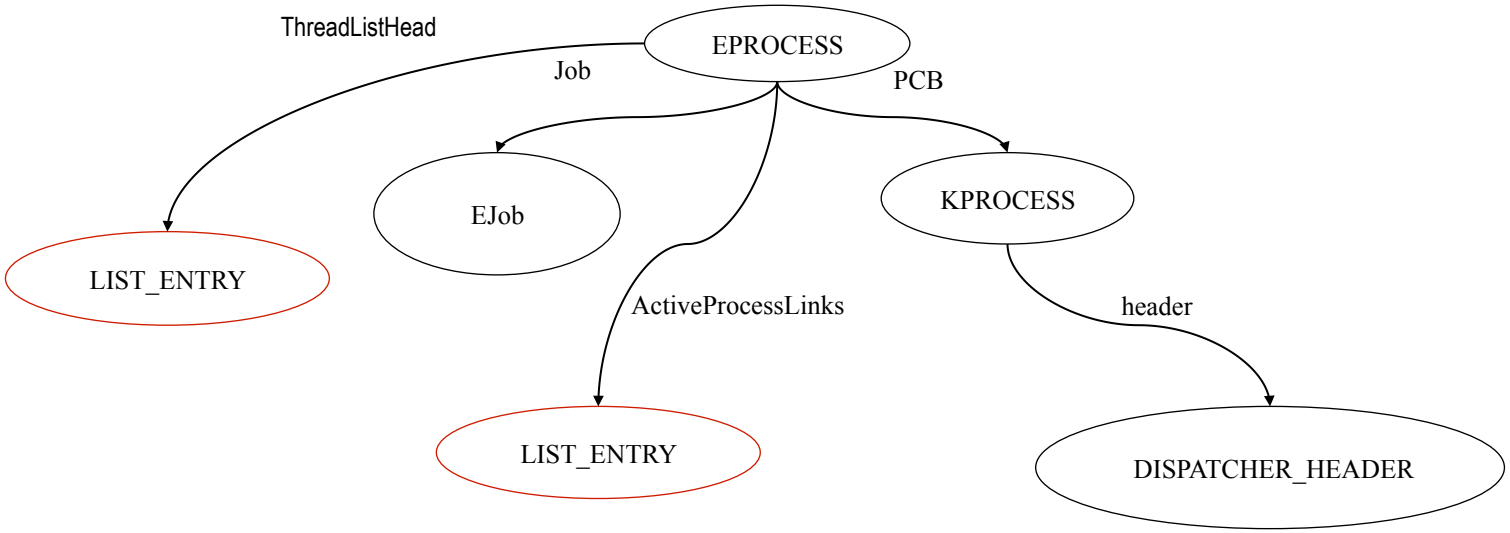
- A new static analysis tool that can generate an accurate type graph for any C program
- Is able to generate a sound data definition for large C-based OS - without any prior knowledge of kernel data layout
- Disambiguates pointer relations including generic pointers to infer their candidate types & values - by performing static points-to analysis on source code
- We designed and implemented a new points-to analysis algorithm that has the ability to provide interprocedural, context-sensitive and field-sensitive points-to analysis
- Accuracy is more important than speed for our application domain
- Scales to extremely large C programs that contain millions of lines of code
- Performs its analysis “off-line” – thus generated type graph can be used by security solutions in on-line security mode

Tool structure



Type graph ouput

- Outputs a initial type-graph that reflects the direct inclusion-based relations between kernel data structures that have clear type definitions
  - Nodes are data structures and edges are data members (inclusion relations) of the structures e.g. for EPROCESS:



Three steps (see paper for details)

- Phase 1: Intraprocedural Analysis
  - Compute a local points-to graph for each procedure without information about caller or callee
  - Create nodes
  - Compute the transfer function
  - Connect edges
  - => this gives us typing of variables within procedures
- Phase 2: Interprocedural Analysis
  - Iincorporate interprocedural information from the callees of each procedure - without any context information yet
  - => extends intermediate type graph with callee information
- Phase 3: Context-Sensitive Points-to Analysis
  - Step 1: Points-to Sets Accumulation
    - Compute the points-to sets
  - Step 2: Graph Unification – unify collections of points-to sets
  - Step 3: Context-Sensitive
    - Context Sensitivity – output of procedure bind to calling site
    - Indirect Points-to Relations found
  - `=> gives complete type graph`

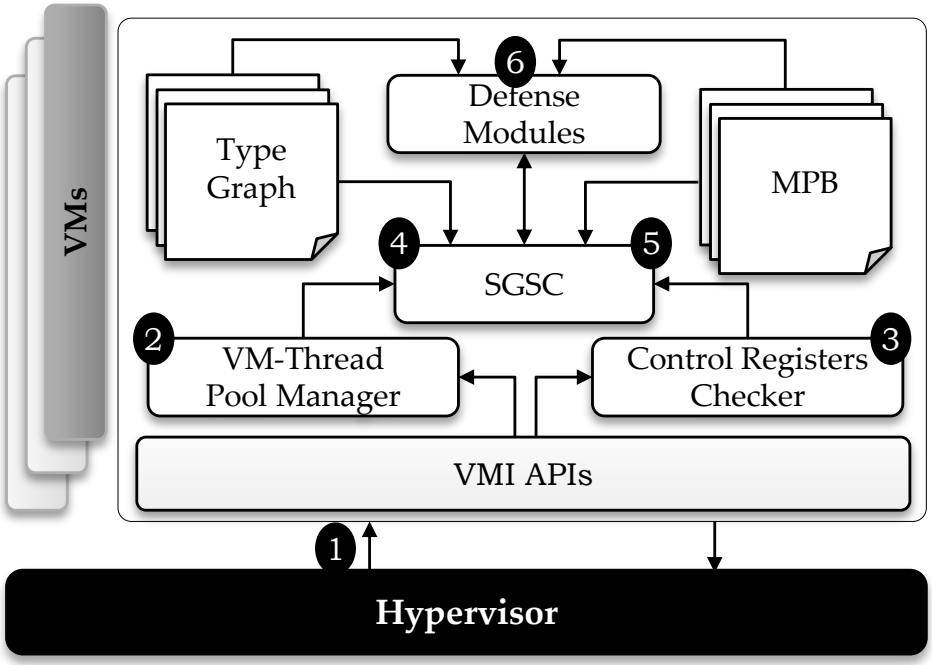
Evaluation 1 - benchmarks

- Soundness and Precision
  - The points-to analysis algorithm is sound if the points-to set for each variable contains all its actual runtime targets, and is imprecise if the inferred set is larger than necessary
    - SPEC2000 and SPEC2006 benchmark suites and other open source C programs
- Kernel Analysis
  - WRK (~ 3.5 million LOC) and Linux kernel v3.0.22 (~ 6 million LOC)
    - 28 hours to analyse the WRK and around 47 hours to analysis the Linux kernel.

Benchmark`	LOC	Pointer Inst	Proc	Struct	AST T (sec)	AST M (MB)	AST C (%)	TG T (sec)	TG M (MB)	TG C (%)	P (%)	S (%)
art	1272	286	43	19	22.7	21.5	19.9	73.3	12.3	17.6	100	100
equake	1515	485	40	15	27.5	25.4	20.4	87.5	14.1	21.1	98.6	100
mcf	2414	453	42	22	43.2	41	28.5	14	23	27	97.2	100
gzip	8618	991	90	340	154.2	144.6	70.5	503.3	81.4	68.3	95.1	100
parser	11394	3872	356	145	305.2	191.2	76.7	661.4	107.8	74.3	94.5	100
vpr	17731	4592	228	398	316.1	298.7	80.2	1031.5	163.2	79	NA	100
gcc	222185	98384	1829	2806	3960.5	3756.5	93.5	12962	2200	94	NA	100
sendmail	113264	9424	1005	901	2017.2	1915.1	91.6	6609	1075.0	91.5	NA	100
bzip2	4650	759	90	14	82.3	78.1	45.5	271.6	44.2	42.9	95.9	100

Evaluation 2 – use for security

- Integrated into VM Monitoring Tool – CloudSec
- Mapping kernel address space based on KDD-generated type graph
- Windows XP 64 bit VM
- Performance Overhead
  - 8.5 minutes for a memory image of 4 GB on a 2.8 GHz CPU with 6 GB RAM.
- Low rate of false positives



Conclusion

- Built new C program analysis tool to recover detailed, accurate type graphs – disambiguating void, null pointers and casting
- Can apply to very large scale C programs – millions of lines of code
- Motivated for use on C-based OS kernal data integrity checking
- Future work:
  - Improve performance of KDD via multi-processes, more efficient intermediate structures
  - Use for external Virtual Machine OS kernel Integrity checking
  - Use for Function Pointer Checking
  - Use for Type-Inference
- Detection of “Zero-Day Threats” i.e. never before seen attacks on OS kernel
  - By checking kernel data integrity at run-time

References

Ibrahim, A., Hamlyn-Harris, J., Grundy, J.C. and Almorsy, M., DIGGER: Identifying OS Kernel Objects for Run-time Security Analysis, International Journal on Internet and Distributed Computing Systems, vol 3, no. 1, January 2013, pp 184-194.

Ibrahim, A., Hamlyn-Harris, J., Grundy, J.C. and Almorsy, M., Supporting Operating System Kernel Data Disambiguation using Points-to Analysis, 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012), Sept 3-7 2012, Essen, Germany, ACM Press.

Ibrahim, A., Hamlyn-Harris, J., Grundy, J.C., Almorsy, M., Operating System Kernel Data Disambiguation to Support Security Analysis, 2012 International Conference on Network and System Security (NSS 2012), Fujian, China, Nov 21-23 2012, LNCS, Springer.

Ibrahim, A., Hamlyn-Harris, J., Grundy, J.C., Almorsy, M. Supporting Virtualizaion-Aware Security Solutions using a Systematic Approach to Overcome the Semantic Gap, 5th IEEE Conference on Cloud Computing (CLOUD 2012), IEEE CS Press, Waikiki, Hawai, USA, June 24-29 2012.

Imbrahim, A., Hamlyn-Harris J., Grundy, J.C. and Almorsy, M., CloudSec: A Security Monitoring Appliance for Virtual Machines in the IaaS Cloud Model, In Proceedings of the 5th International Conference on Network and System Security (NCC 2011), Milan, Italy, September 5-7 2011, IEEE Press.

Ibrahim, A., Hamlyn-Harris, J. and Grundy, J.C., Emerging Security Challenges of Cloud Virtual Infrastructure, In Proceedings of the 2010 Asia Pacific Cloud Workshop 2010 (co-located with APSEC2010), Sydney, Nov 30 2010.