Polymorphing Software by Randomizing Data Structure Layout

Zhiqiang Lin

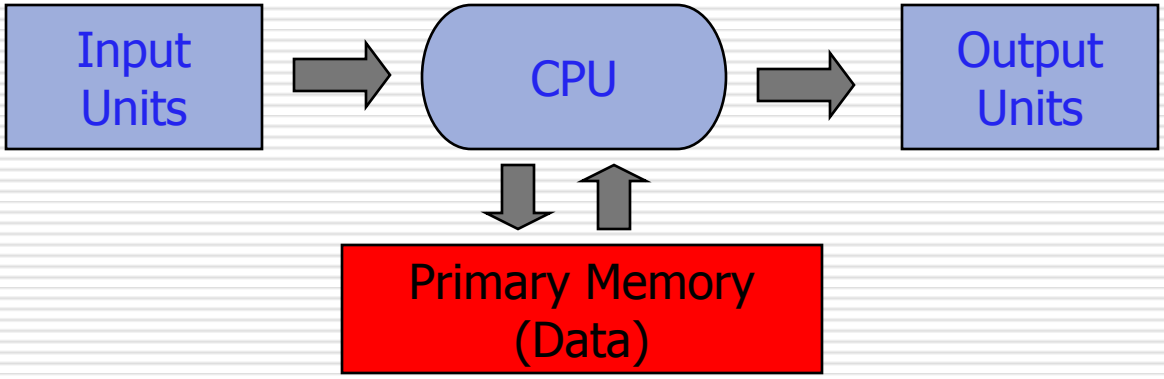
Ryan D. Riley and Dongyan Xu

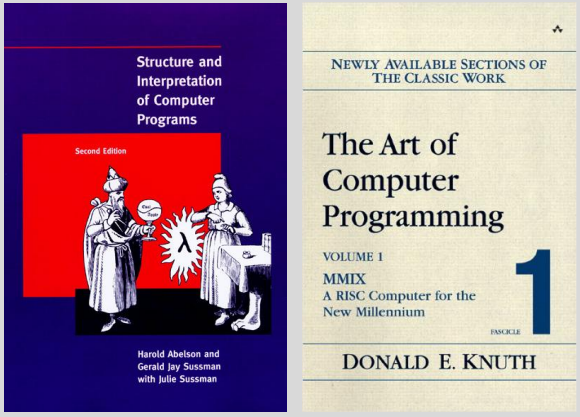
July 9th, 2009

CERIAS
Center for Education and Research in Information Assurance and Security

PURDUE UNIVERSITY

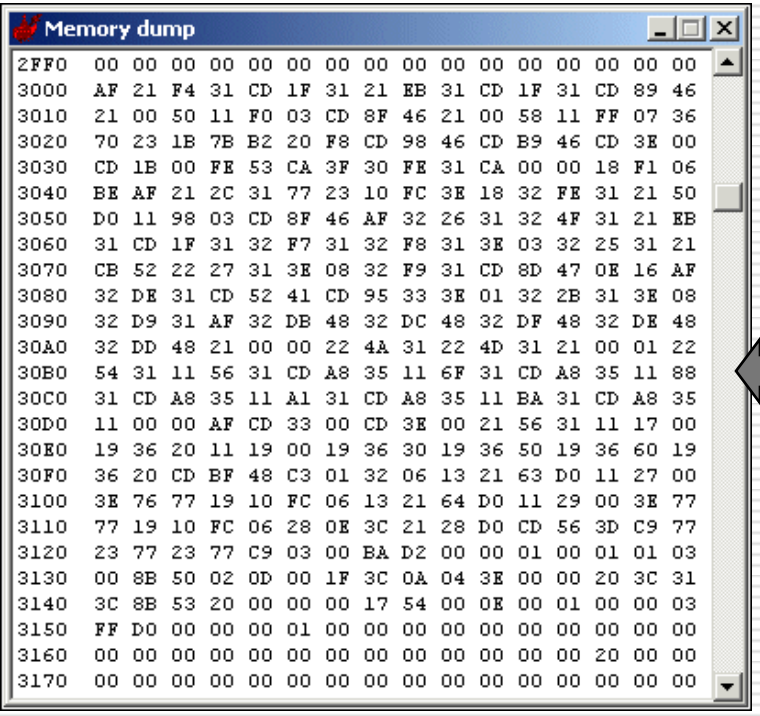
Observations: all programs use data structures

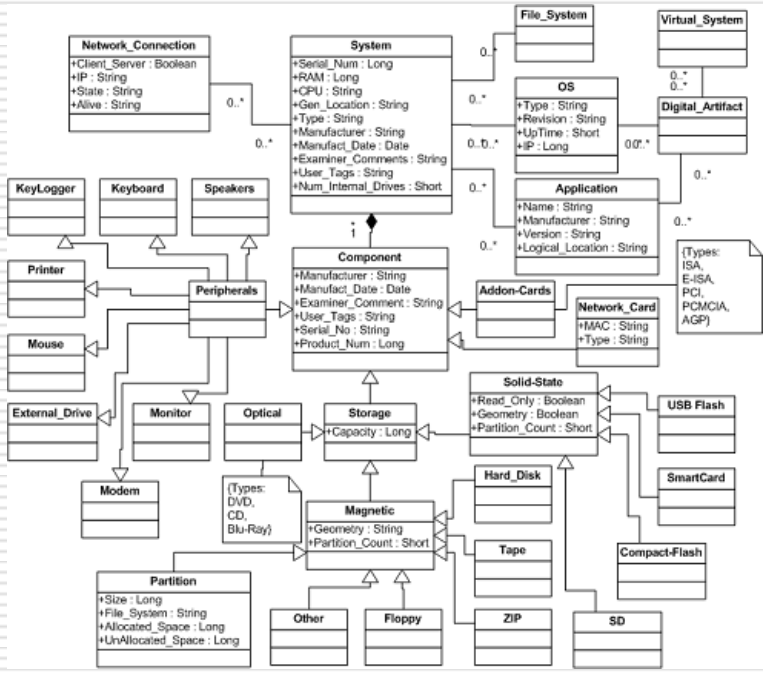




In computer science, a **data structure** is a particular way of storing and organizing data in a computer so that it can be used *efficiently*

Why we need to care about it in security?





Digital Forensics -- Identify digital evidence for an investigation

Why we need to care about it in security?

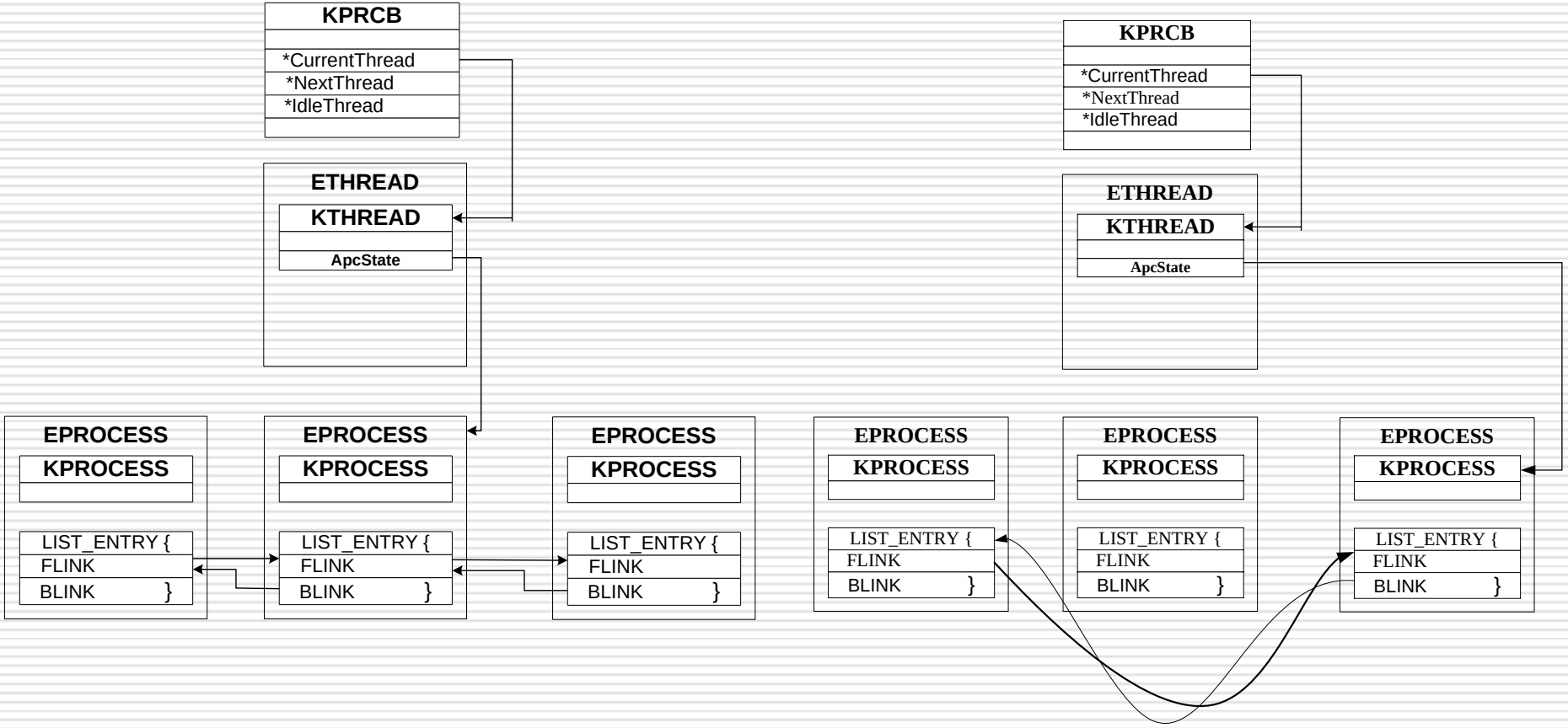
```
struct A
{
        int a;
        char b;
        int c;
        float d;
};
```

=

```
struct A
{
        int a;
        char b;
        int c;
        float d;
};
```

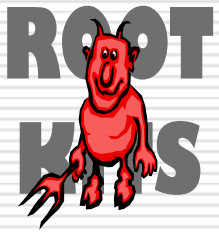
Laika [OSDI'08]: Using Data structure as a classifier → Malware Signature

Why we need to care about it in security?



ROOT KITS

Motivation



DSLR: Polymorphing data structure layout

Data structure signature

Outline

- Motivation
- Technical Challenges
- Design & Implementation
- Evaluation
- Related Work
- Conclusion

Technical Challenges (I)

- Randomizability of Data structure
  - Network communication

```
41 struct udphdr {
42      u_int16_t   uh_sport;       /* source port */
43      u_int16_t   uh_dport;       /* destination port */
44      u_int16_t   uh_ulen;        /* udp length */
45      u_int16_t   uh_sum;         /* udp checksum */
46 };
```

Technical Challenges (I)

- Randomizability of Data structure
  - Network communication
  - Standard definition (e.g., stdio.h)

```
/usr/include/fstab.h
...
57 struct fstab
58   {
59     char *fs_spec;          /* block special device name */
60     char *fs_file;          /* file system path prefix */
61     char *fs_vfstype;       /* File system type, ufs, nfs */
62     char *fs_mntops;        /* Mount options ala -o */
63     const char *fs_type;    /* FSTAB_* from fs_mntops */
64     int fs_freq;            /* dump frequency, in days */
65     int fs_passno;          /* pass number on parallel dump */
66   };
67
```

Technical Challenges (I)

- Randomizability of Data structure
  - Network communication
  - Standard definition (e.g., stdio.h)
  - Zero length array

```
struct stringlib
{
        int length;
        char str[0];
};
```

Technical Challenges (I)

- Randomizability of Data structure
  - Network communication
  - Standard definition (e.g., stdio.h)
  - Zero length array
  - Directly using data offset to access some fields

```
p=&struct_a;
var_a=*(p+4);
```

Technical Challenges (I)

- Randomizability of Data structure
  - Network communication
  - Standard definition (e.g., stdio.h)
  - Zero length array
  - Directly using data offset to access some fields
  - Initialized data structure

```
struct role = { "Hamlet", 7, FALSE, "Prince of Denmark", "Kenneth Branagh"};
```

Technical Challenges (I)

- Randomizability of Data structure
  - Network communication
  - Standard definition (e.g., stdio.h)
  - Zero length array
  - Directly using data offset to access some fields
  - Initialized data structure
  - …

- *How to spot the randomizable data structures*

Programmer has the confidence

Technical Challenges (II)

- How to randomize a data structure
  - (1) Reorder the Field

```
struct A
{
        int a;
        char b;
        int c;
        float d;
};
```

```
struct A
{
        char b;
        int a;
        float d;
        int c;
};
```

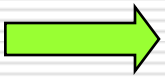
#m fileds → **m!**

#n data structures, #m fields each →**variance** **$(m!)^n$**

Technical Challenges (II)

- How to randomize a data structure
  - (1) Reorder the Field
  - (2) Insert Garbage fields

```
struct A
{
        int a;
        int b;
};
```



```
struct A
{
        int a;
        Garbage;
        int b;
};
```

Technical Challenges (II)

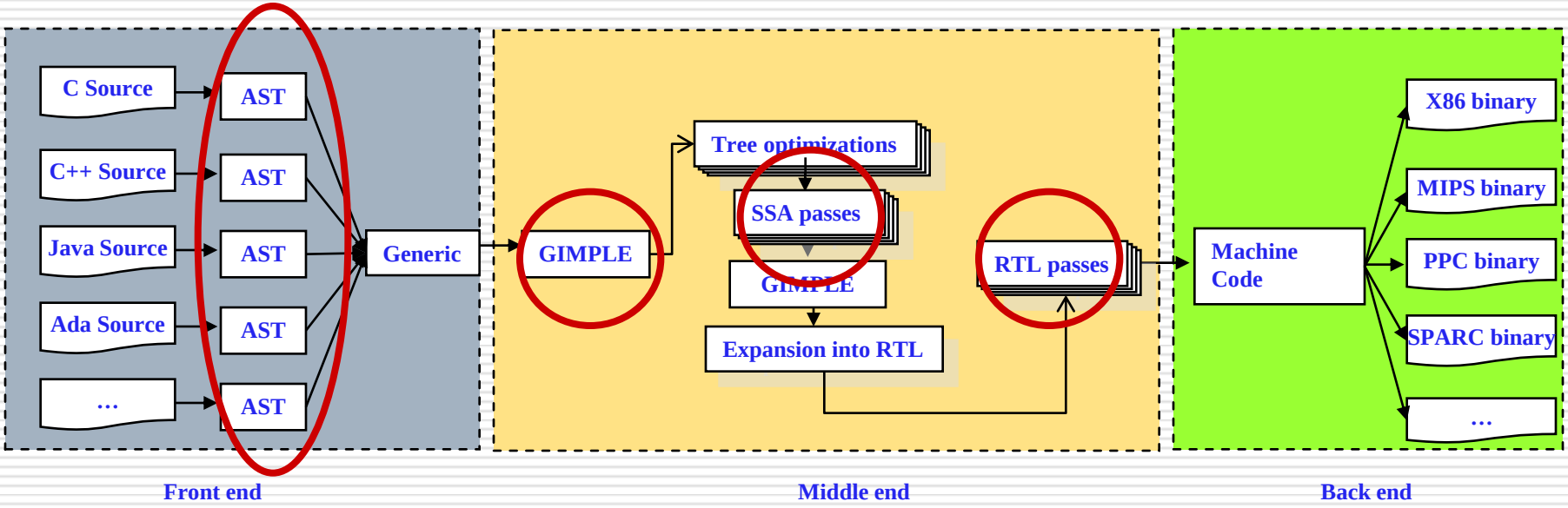
- How to randomize a data structure
  - (1) Reorder the Field
  - (2) Insert Garbage fields
  - (3) Split the struct

```
struct A
{
        int a;
        char b;
        int c;
        float d;
};
```

```
struct A1
{
        int a;
        int c;
};
```

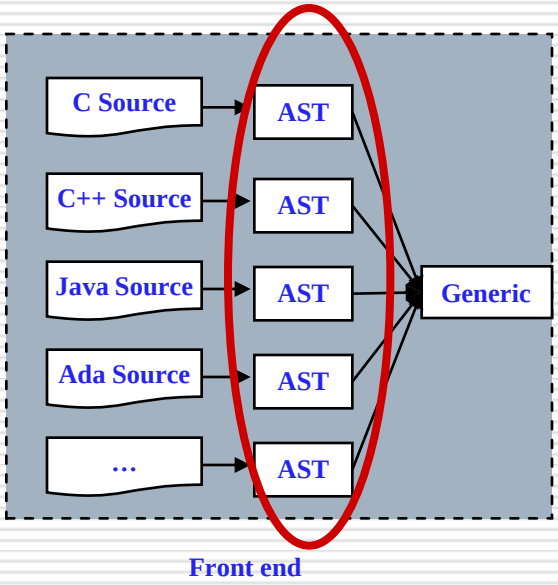
```
struct A2
{
        char b;
        float d;
};
```

Design in GCC



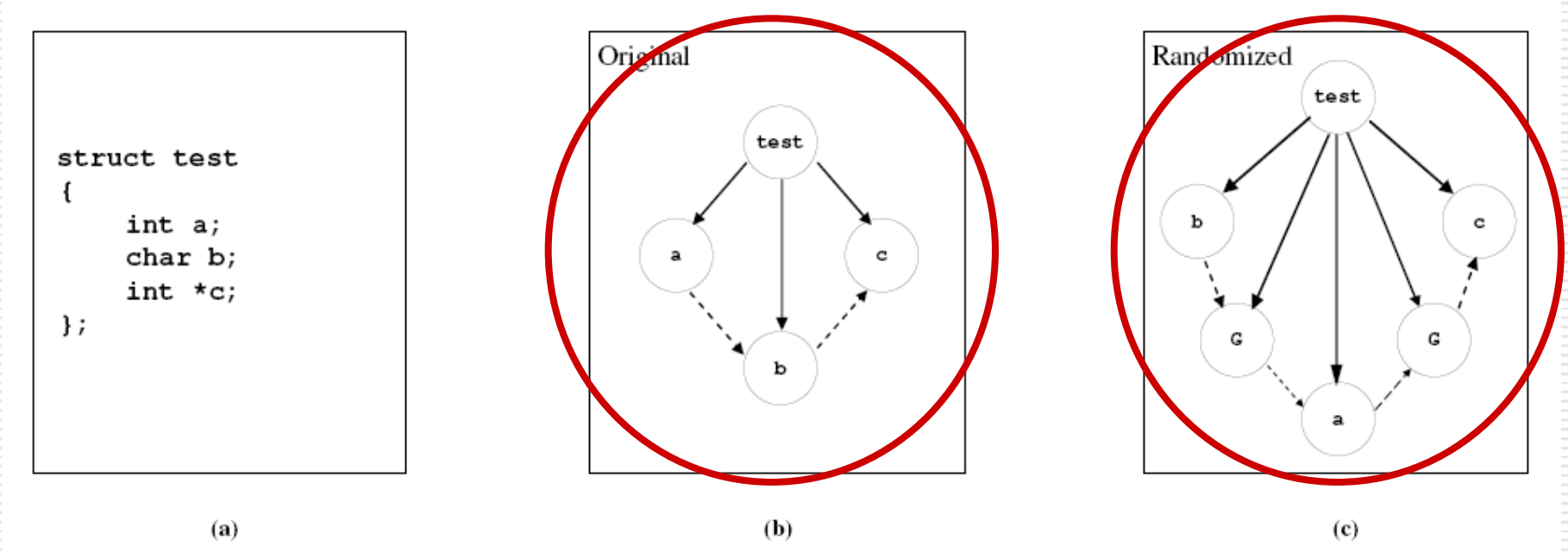
Four Intermediate Representations: AST, GIMPLE, SSA, RTL

Instrument at AST because of



- (1) Much more original information
  - The type and scope information for data structures
- (2) Easier to understand and modify
- (3) No need to touch specific memory addresses, and the instructions.

Examples



Randomized “AST”

How to use our compiler

<obfuscate-specifier> ::= **__obfuscate__((<obfuscate-list>))**
<obfuscate-list> ::= <obfuscate-property>
| <obfuscate-list>, <obfuscate-property>
<obfuscate-property> ::= **ε | __reorder__ | __garbage__**

```
1 class Test
2 {
3     int a;
4     char b;
5     int *c;
6     ...
7 } __obfuscate__ (( __reorder__ ));
```

```
 1 #include <stdio.h>
 2 struct Test
 3 {
 4     int a;
 5     char b;
 6     int *c;
 7 } __obfuscate__ (( __reorder__ , __garbage__ ));
 8 __obfuscate__ (( __reorder__ )) int main(void)
 9 {
10     int loc1 = 1;
11     char loc2 = 'n';
12     char loc3[4];
13     printf(" The address in struct:
14             %x , %x , %x\n", &t.a, &t.b, &t.c);
15     printf(" The address in local:
16             %x, %x, %x\n",&loc1,&loc2,&loc3);
17     return 0;
18 }
```

Evaluation on Estimation of Randomizability

Program	LOC	Struct	class	w
42-Virus	0.88	1/1	-	4E5
Slapper	2.44	26/30	-	5E47
pingrootkit	4.81	26/27	-	5E15
Mood-nt	5.31	36/37	-	8E119
tnet-1.55	11.56	14/17	-	7E82
Suckit	24.71	110/111	-	9E159
agobot3	245.44	23/31	50/50	2E1106
patch2.5.4	11.53	5/7	-	4E3
bc-1.06	14.29	20/21	-	6E56
tidy4	15.95	9/18	-	2E52
ctags-5.7	27.22	51/79	-	3E668
openssh4.3	76.05	63/80	-	4E1271

Most of the programmer defined data structure is randomizable

Evaluations against rootkit attacks

Randomized "task_struct"

Rootkit	Attack Vector	Prevented?
adore-ng 0.56	LKM	√
enyelkm 1.2	LKM	√
override	LKM	√
fuuld	DKOM-/dev/kmem	√
intoxnia ng2	LKM	X
Mood nt	/dev/kmem	X

No data structure involved in these attacks

Evaluation on Signature Evasion

Benchmark	Binary	Code diversity	Mixture Ratio
7zip-4.6.4 (A)	502K	4.26%	0.50942826
7zip-4.6.4 (B)	504K	5.88%	0.51487480
agobot3-0.2.1 (A)	1.18M	6.18%	0.70016150
agobot3-0.2.1 (B)	1.19M	6.34%	0.60932887

Laika[osdi'08] uses a mixture ratio to quantify similarity: the closer the value is to 0.5, the higher the similarity

7-zip only has 25 data structures randomized, and it has more than 80 unandomizable one (library). Agobot has 49 "stucts", and 50 classes in its own

Limitation & Future work

- AST vs. GIMPLE, SSA, RTL
- Automatic Identification of the Randomizability
- Struct/Class split
- Variant Instance is generated by compiling

Related work

- Security through Diversity [Forrest et al. 1997]
  - Address Space Randomization (ASR) (e.g., Kernel Patch [PAX], Loader [SRDS'03], Source code [Sec'05], binary code [Sec'03])
  - Instruction Set Randomization (ISR) [CCS'03]
  - Data Randomization (e.g., PointGuard [Sec'03])
  - Operating System Interfaces Randomization (e.g., RandSys[SRDS'07])
  - Multi-variant System (e.g, N-variant[Sec'05], Die-hard[PLDI'06])
- Layout Manipulations and Obfuscations in Compilers
  - Propolice
  - Cache aware layout reorganization optimization [GCC Developers' Summit '05, '07, PLDI'04]

Conclusion

- DSLR -- A new software randomization technique
- *Thwart data structure need-to-know attack*
  - Rootkit

- *Increase the bar for software defense*
  - Forensics analysis
  - Data structure based signature (false negatives)

Source code (GPL)
http://www.cs.purdue.edu/homes/zlin/dimva09.html

Q&A

zlin@cs.purdue.edu