TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

Protecting XALT from Users

Robert McLay

April 21, 2022

XALT: Outline

- XALT is linking with every program that runs on the system
- Users will occasionally make mistakes
- Need to protect XALT from user mistakes
- Show more ways that XALT protects itself and users.

Three examples of protection

- User's bug hidden by zeroed memory initially
- User's mixing Fortran routine with C library routines badly
- XALT expecting well managed memory heap.

How XALT works

```
#include <stdio.h>
void myinit(int argc, char **argv)
{ printf("This is run before main()\n"); }
void myfini()
{ printf("This is run after main()\n"); }

__attribute__((section(".init_array"))) __typeof__(myinit) *__init = myinit;
__attribute__((section(".fini_array"))) __typeof__(myfini) *__fini = myfini;
```

- my_docs/22/xalt_monthly_mtg_2022_03_17/code/bad_memory/ex1

How XALT works (II)

```
% cat try.c

#include <stdio.h>
int main()
{
  printf("Hello World!\n");
  return 0;
}
```

How XALT works (III)

```
$ ./try

Hello World!

$ LD_PRELOAD=./libxalt.so  ./try
This is run before main()
Hello World!
This is run after main()
```

- my_docs/22/xalt_monthly_mtg_2022_03_17/code/bad_memory/ex1

How XALT works (IV)

- The xalt runtime library attaches to every program run on your system!
- I sometimes feel like I'm a developer on every program team.
- XALT programs are in the same namespace as the user's program (UGH!)

Hiding XALT routine names from users

```
% nm $LD_PRELOAD| grep __XALT_build
0000000000009e80 T __XALT_buildEnvT_xalt_1_5
000000000000a840 T __XALT_buildUserT_xalt_1_5
00000000000163a0 T __XALT_buildXALTRecordT_xalt_1_5
```

- XALT routine names are hidden by macros supplied in xalt_obfuscate.h

User's bug hidden by initially zeroed memory

- Initially all memory is zeroed before program starts
- Note that pointer zero, integer zero and float zero are all zero bits
- Link lists require a NULL pointer at end of list.
- Used memory is *NOT* zeroed for you in C.
- User's program work w/o XALT, Failed with XALT.

Example code clean/used memory

```
% cat try.c

#include <stdio.h>
#include <stdlib.h>
#define SZ 1000
int main()
{
  int *a = (int *) malloc(SZ*sizeof(int));
  printf("Hello World! a:%d\n",a[0]);
  return 0;
}

% cat xalt.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SZ 1000
void myinit(int argc, char **argv)
{
  int i;
  int *a = (int*) malloc(SZ*sizeof(int));
  for (i = 0; i < SZ; ++i) a[i] = 15;
  free(a);
  printf("This is run before main()\n");
}
__attribute__((section(".init_array"))) __typeof__(myinit) *__init = myinit;
```

- my_docs/22/xalt_monthly_mtg_2022_03_17/code/bad_memory/ex2

Example code clean/used memory(II)

```
% ./try

Hello World!  a:0

% LD_PRELOAD=./libxalt.so  ./try  ; echo
This is run before main()
Hello World!  a:15
This is run after main()
```

- my_docs/22/xalt_monthly_mtg_2022_03_17/code/bad_memory/ex2

XALT Fix: zero memory before free()

- To protect XALT from broken user code
- XALT in myinit() zero's memory before free
- Note that non-MPI tracking does little allocation
- MPI tasks $>$ 127 init record $\Rightarrow$ much allocation

XALT Fix: zero memory before free()

```
% cat try.c

#include <stdio.h>
#include <stdlib.h>
#define SZ 1000
int main()
{
  int *a = (int *) malloc(SZ*sizeof(int));
  printf("Hello World! a:%d\n",a[0]);
  return 0;
}

% cat xalt.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SZ 1000
void myinit(int argc, char **argv)
{
  int i;
  int *a = (int*) malloc(SZ*sizeof(int));
  for (i = 0; i < SZ; ++i) a[i] = 15;
  memset((void *) a, 0, SZ*sizeof(int));
  free(a);
  printf("This is run before main()\n");
}
__attribute__((section(".init_array"))) __typeof__(myinit) *__init = myinit;
```

- my_docs/22/xalt_monthly_mtg_2022_03_17/code/bad_memory/ex3

XALT Fix: zero memory before free() (II)

```
% ./try

Hello World!  a:0

% LD_PRELOAD=./libxalt.so  ./try  ; echo
This is run before main()
Hello World!  a:0
This is run after main()
```

- my_docs/22/xalt_monthly_mtg_2022_03_17/code/bad_memory/ex3

Protecting XALT from Fortran mixed with C programs badly

```
% cat msg.f90
subroutine msg
   print *, "Hello World!"
end subroutine msg

% nm try | grep msg
00000000000011c7 T msg_
```

- Normally Fortran routines get a trailing underscore when compiled
- This can be disabled:
- Fortran: -fno-underscoring
- ifort: -assume nounderscore
- Can make mixing C/Fortran easier
- Also make collisions with C library easier

XALT uses libuuid

- libuuid.so is used to get a unique identifier
- It uses libc's random()
- Can't have two routines named random()
- my_docs/22/xalt_monthly_mtg_2022_03_17/code/random/ex3

Collision over random() routine

```
% cat try.f90

program tryMe
   implicit none
   real*8 d, random
   print *, "Hello World!"
   d = random(1.0, 2.0. 3.0)
   print *, "d: ",d
end program tryMe

% cat random.f90

real*8 function random(a, b, c)
   implicit none
   real*8 a, b, c
   print *, "In random(a, b, c)"
   random = a*b + c
end function random

% cat xalt.c
#include <stdio.h>
#include <stdlib.h>
void myinit(int argc, char **argv)
{
  long int a;
  printf("This is run before main()\n");
  a = random();
  printf("called random(): a: %ld\n",a);
}
__attribute__((section(".init_array"))) __typeof__(myinit) *__init = myinit;
```

Collision over random() routine (II)

```
% ./try

 Hello World!
 In random(a, b, c)
 d:    5.0000000000000000

% LD_PRELOAD=./libxalt.so  ./try  ; echo
This is run before main()
 In random(a, b, c)
Segmentation fault
```

- The linker chooses the user's Fortran random() instead of the C lib random()
- The segfault happens because the Fortran random() expects 3 arguments
- the random() call in xalt.c passes none.

How to fix this issue

- Other Fortran program might do the same thing
- Trick: Use dlopen()/dlsym() to dynamically link in libuuid.so
- At this point libuuid.so can't "see" the Fortran random() routine
- This trick solves many problems with libuuid

XALT is still susceptible to similar issues

- XALT is now protected from a user's random() function
- But XALT is vulnerable some Fortran code replacing a c library routine
- We will just have to fix them as they come up

Protecting XALT from badly managed memory heap

```
void my_free(void *ptr,int sz)
{
  if (s_start_record && ptr != NULL)
    {
      memset(ptr, '\0', sz);
      free(ptr);
    }
}
```

- Reporting an end record in myfini() requires memory allocations
- However some user programs can leave the heap broken
- XALT replaces free() with my_free()
- Memory is only freed for a start record.

Containers

- Containers are chrooted environments
- They can have minimum setup and libraries
- Typically only what the user needs
- But not always what XALT needs.
- The xalt library needs many libraries libcrypto, libuuid, libcurl, etc.

Containers (II)

- For XALT to work at all in a container. LD_PRELOAD must be set.
- And add XALT_INSTALL_DIR to container's available paths.
- So XALT during installation copies system libraries to the XALT_INSTALL_DIR
- XALT does a ldd on executables and libraries to find all the system libraries that need to be copied.
- This allows XALT to dlopen()/dlsym() libuuid.so to solve the problems mentioned before.

MPI libraries disallow system() after main() returns

- One of the version of IMPI, stopped supporting system() being called after main() finishes
- This was a disaster for collecting data from MPI programs.
- It even prevented collecting data from one task mpi programs
- This was a problem because XALT used C++ programs to collect data.

Disallowed system() after main()

- XALT needs Hash Table (AKA Dictionaries) to store many key-value pairs.
- C++ has built-in hash tables via the STL (unordered-maps)
- C does not.
- The XALT library (libxalt_initialize.so) is written in C
- It would be a bad idea to have written the library in C++

Solution

- Re-write the XALT submission programs into C
- And include them in libxalt_initialize.so
- Where to find a hash table implementation
- uthash

UTHASH and friends

- UTHASH: github: https://github.com/troydhanson/uthash
- UTHASH: Docs: https://troydhanson.github.io/uthash/userguide.html
- uthash is written in C via uthash.h (no library!)

UTHASH

```
// One C++ stmt:
userDT["start_time"] = start_time;

// Becomes several C stmts:
insert_key_double(&userDT, "start_time",  start_time);
// -->
void insert_key_double(S2D_t** userDT, const char* name, double value)

  S2D_t* entry = (S2D_t *) XMALLOC(sizeof(S2D_t));
  utstring_new(entry->key);
  utstring_bincpy(entry->key, name, strlen(name));

  entry->value = value;

  HASH_ADD_KEYPTR(hh, *userDT, utstring_body(entry->key),
                  utstring_len(entry->key), entry);
```

UTHASH rewrite

- It took a couple weeks to rewrite the submission programs into routines
- The only system calls that still exist are the curl calls for transport.

Support for filtering of packages

- Python packages could be directly filtered through the sitecustomize.py file
- But not R or MATLAB packages
- All packages collection goes through the xalt_record_pkg program

All package filtering

- New pkg_pattern array available in XALT 2.10.37+
- This goes in your site Config.py
- The pattern looks like this:
- Program:kind:pattern

```
pkg_patterns = [
  ["SKIP",  r'^R:name:stats'],          # SKIP the R pkg named stats
  ["SKIP",  r'^R:name:base'],           # SKIP the R pkg named base
  ["SKIP",  r'^R:name:methods'],        # SKIP the R pkg named methods
 ["SKIP",  r'^python:name:_.*'],      # SKIP all python name that start with an '_'
  ["SKIP",  r'^python:path:[^/].*'],    # SKIP all python built-in packages
 ["SKIP",  r'^python:path:\/home'],    # SKIP all python package in user locations
]
```

This package filtering could be added to ingestion

- The same routine could be added to ingestion.
- If wanted it could be added in late May 2022.

Conclusions

- XALT has matured greatly from working with user programs
- Since the XALT library is in the same namespace as the user code
- There is always a risk of routine collision.

Future Topics?

- Recent changes to importing json records
- Others?