Windows Kernel Internals
Common Coding Errors

David B. Probert, Ph.D.
Windows Kernel Development
Microsoft Corporation

© Microsoft Corporation

Improving Software Resilience

- Resilient to:
  - Attacks from malicious users
  - Resource limitations like low memory
  - Concurrency from multiple threads/processes and lock problems
  - Attempts to consume large amounts of scarce resources (memory, disk space, etc.)
  - Being called from rarely seen but legitimate environments

© Microsoft Corporation

Parameter Validation

- Most important at interfaces that cross security boundaries
- The following sub-types are common
  - Buffer length or numerical range check
  - Stack buffer overflows
  - Integer overflow and underflow
  - Probe and capture problems
  - Signed/Unsigned issues

© Microsoft Corporation

Range and Bounds Checking

```
GETUSHORT(&pReqU->cNames,
                pReq->cNames);
for (I = 0; I < pReqU->cNames; i++) {
      GETUSHORT(&pReqU->Names[i].wType,
                      pReq->Names[i].wType);
      memcpy(pReqU->Names[i].NBName,
                pReq->Names[i].NBName,
                NAME_LEN);
}
```

© Microsoft Corporation

Integer Underflow

At this point FrameLength is >= 1

```
if (*FramePointer == 0xC1) || *FramePointer == 0xCF)) {
      PPPProtocolID = (*FramePointer << 8) |
                              *(FramePointer + 1);
      FramePointer += 2;
      FrameLength -= 2;       ← FrameLength may wrap
} else {
     PPPProtocolID = *FramePointer;
     FramePointer++;
     FrameLength--;
}
```

© Microsoft Corporation

Integer Overflow

```
FullName.Length = Dcb->FullFileName.Length +
                          FileName->Length + 1;
FullName.Buffer = ALLOCATE( FullName.Length );

RtlCopyMemory( FullName.Buffer,
                   Dcb->FullFileName.Buffer,
                   Dcb->FullFileName.Length );

FullName.Buffer[ Dcb->FullFileName.Length ] = '¥¥';

RtlCopyMemory( FullName.Buffer +
                   Dcb->FullFileName.Length + 1,
                   FileName->Buffer, FileName->Length );
```

© Microsoft Corporation

Detecting Integer Overflow

- A + B overflows if:

  A + B < A

- A + B + C overflows if :

  A + B < A or

  A + B + C < C

© Microsoft Corporation

Code Reordering For Overflows

```
typdef struct _PATH_BUFFER {
    ULONG PathLength;
    WCHAR Path[1];
} PATH_BUFFER, *P PATH_BUFFER;

RETVAL SetPath (VOID *Buffer, ULONG InputLength)
{
    PATH_BUFFER PathBuffer = Buffer;

    if (FIELD_OFFSET(PATH_BUFFER, Path[0]) +
        PathBuffer->PathLength > InputLength) {
       return ERROR;
    }
```

© Microsoft Corporation

Code Reordering For Overflows

```
RETVAL SetPath (VOID *Buffer, ULONG InputLength)
{
    PATH_BUFFER PathBuffer = Buffer;

    if (InputLength < FIELD_OFFSET(PATH_BUFFER, Path[0])) {
        return ERROR;
    }
    if (InputLength - FIELD_OFFSET(PATH_BUFFER, Path[0]) <
         PathBuffer->PathLength ) {
        return ERROR;
    }
```

© Microsoft Corporation

Integer Overflow with Scaling

```
ProbeForRead(pcptl, ccptl * sizeof(ULONG),
                    sizeof(BYTE));
i = ccptl;
pul = pcptl;
do {
    c += *pul++;
} while (--i != 0);
```

© Microsoft Corporation

Detecting Scaling Overflows

```
if (ccptl > MAX_ULONG / sizeof(ULONG)) {
    return ERROR;
}
ProbeForRead(pcptl, ccptl * sizeof(ULONG),
                    sizeof(BYTE));
i = ccptl;
pul = pcptl;
do {
    c += *pul++;
} while (--i != 0);
```

© Microsoft Corporation

Stack Buffer Overflows

```
  BOOL
  EnumPrinters(
    DWORD   Flags,
    LPWSTR   Name,
…
    LPDWORD pcReturned)
  {
      WCHAR   FullName[MAX_PATH];
 ...
 ...
      if (Name && *Name && (Level == 1)) {
          wcscpy(FullName, Name);
```

© Microsoft Corporation

Signed/Unsigned Issues

```
    typedef struct _EXCEPTION_RECORD {
        ULONG NumberParameters;
        ULONG
ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
    } EXCEPTION_RECORD;

    LONG Length;

    Length = ExceptionRecord->NumberParameters;
    if (Length > EXCEPTION_MAXIMUM_PARAMETERS) {
          return STATUS_INVALID_PARAMETER;
    }

    Length = (sizeof(EXCEPTION_RECORD) +
            ((Length - EXCEPTION_MAXIMUM_PARAMETERS) *
              sizeof(ULONG)));
```

© Microsoft Corporation

Probe and Capture

- User-mode memory properties:
  - Memory addresses must be probed before use
  - Every single reference may raise an exception
  - Contents may change asynchronously (two reads may return two different values)
  - Don’t use as temporary storage (writes may be lost)
  - May not be aligned correctly
  - May contain hostile values designed to break your code

© Microsoft Corporation

Missing Probes

```
NTSTATUS
NtAddAtom (IN PWSTR AtomName,
           IN ULONG Length,
           OUT PRTL_ATOM Atom)
...
    try {
        *Atom = ReturnAtom;
```

© Microsoft Corporation

Missing Try/Except Blocks

```
try {
    ProbeForRead (pCount,
                  sizeof (ULONG),
                  sizeof (ULONG));
    i = *pCount;
} except (EXCEPTION_EXECUTE_HANDLER) {
    return GetExceptionCode ();
}
j = *pCount;
```

© Microsoft Corporation

Memory May Change

```
try {
      ProbeForRead (pCount,
                        sizeof (ULONG),
                        sizeof (ULONG));

      if (*pCount > MAX_VALUE) {
            return ERROR;
      }

      for (i = 0; i < *pCount; i++) {
            …
      }
} except (EXCEPTION_EXECUTE_HANDLER) {
      return GetExceptionCode ();
}
```

© Microsoft Corporation

Double Fetch in a Server

```
Status = LsaTable->CopyFromClientBuffer( NULL,
                                         sizeof( DWORD ),
                                         &Cred,
                                         pRemoteCred );
if (Cred.u.Capi.dwType == SCHANNEL_SECRET_TYPE_CAPI) {
    Size = sizeof( SCH_CRED_SECRET_CAPI );
    …
Status = LsaTable->CopyFromClientBuffer(NULL,
                                        Size,
                                        &Cred,
                                        pRemoteCred );

if(Cred.u.Capi.dwType == SCHANNEL_SECRET_TYPE_CAPI)  {
    pCapiCred = SPExternalAlloc( Size );

    pCapiCred->dwType = Cred.u.Capi.dwType;
    pCapiCred->hProv  = Cred.u.Capi.hProv;
```

© Microsoft Corporation

Error Paths

```
pTdiClientRecvInfo = ExAllocatePoolWithTag(NonPagedPool,
                                    sizeof(*pTdiClientRecvInfo),
                                    PPTP_RECV_CTRLDESC_TAG);

if (pTdiClientRecvInfo == NULL)  return STATUS_SUCCESS;

pIrp = TdiBuildAsynchronousInternalDeviceControlIrp  (
                    TDI_RECEIVE,…          );
if (pIrp == NULL) {
        return STATUS_SUCCESS;      ← Leak here
}
```

© Microsoft Corporation

Error Paths Exploit

```
while (1) {
    ipaddr = *(DWORD *) h->h_addr;
    s = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
    …
    on = 1;
    status = ioctlsocket (s, FIONBIO, &on);
    …
    memset (&sockaddr, 0, sizeof (sockaddr));
    sockaddr.sin_family = AF_INET;
    sockaddr.sin_port = htons (1723);
    sockaddr.sin_addr.s_addr = ipaddr;
    status = connect (s, (struct sockaddr *) &sockaddr,
                         sizeof (sockaddr));
    …
    closesocket (s);
}
```

© Microsoft Corporation

Types of locking problems

- Missing locks
- Dropping locks and assuming preserved state
- Misuse of interlocked operations
- Disjoint sets of locks
- Deadlocks
- Missing APC disable for homegrown locks or resources

© Microsoft Corporation

Race Conditions

```
 BOOL IsCallerSystem( VOID )
{
    …
    static PSID SystemSid = NULL;

    if( SystemSid == NULL ) {
        Status = RtlAllocateAndInitializeSid(
                    &NtSidAuthority,
                    1,
                    SECURITY_LOCAL_SYSTEM_RID,
                    0, 0, 0, 0, 0, 0, 0,
                    &SystemSid);
```

© Microsoft Corporation

Race Conditions with Interlocks

```
context = ExAllocatePoolWithTag( PagedPool,
                  sizeof( IO_COMPLETION_CONTEXT ),
                  'cCoI' );
if (context) {
    if (!InterlockedCompareExchangePointer(
              &fileObject->CompletionContext,
              context, NULL )) {

        context->Port = portObject;
        context->Key = completion->Key;
```

© Microsoft Corporation

Dropping Locks Without State Change

- ExAcquireFastMutex(&LpcpLock);

- if (WakeupThread->LpcReplyMessageId ==
  CapturedRequestMessage.MessageId) {

```
…
  ExReleaseFastMutex( &LpcpLock );
...
  WakeupThread->LpcReplyMessageId = 0;
  WakeupThread->LpcReplyMessage = Msg;
```

© Microsoft Corporation

Object Referencing Problems

- Not having an associated reference for reading or modifying the object
- Relying on a user mode handle for a reference
- Bifurcating the execution stream

© Microsoft Corporation

Missing Reference to Cover Operation

```
ACQUIRE_GLOBAL_LOCK();

for (pAdapt = AtmSmGlobal.pAdapterList;
      pAdapt;
      pAdapt = pAdapt->pAdapterNext){

      if (CompareLength == RtlCompareMemory(pOpenInfo->ucLocalATMAddr,
                                            pAdapt->ConfiguredAddress.Address,
                                            CompareLength))
            break;
}

RELEASE_GLOBAL_LOCK();

if(NULL != pAdapt){
      if(!AtmSmReferenceAdapter(pAdapt)){
```

© Microsoft Corporation

Relying on User Mode Handle

```
st = ObReferenceObjectByHandle(ProcessHandle,
          PROCESS_QUERY_INFORMATION,
          PsProcessType,
          PreviousMode,
          (PVOID *)&Process,
          NULL );
if ( !NT_SUCCESS(st) ) {
     return st;
}

ObDereferenceObject(Process);

 try {
     *ProcessInformation = (PVOID)Process->Wow64;
```

© Microsoft Corporation

Summary

- Most errors could be limited by knowing your application environment
- Code for hostile and failure prone environments
- Test failure paths with fault injection

© Microsoft Corporation

Discussion

© Microsoft Corporation