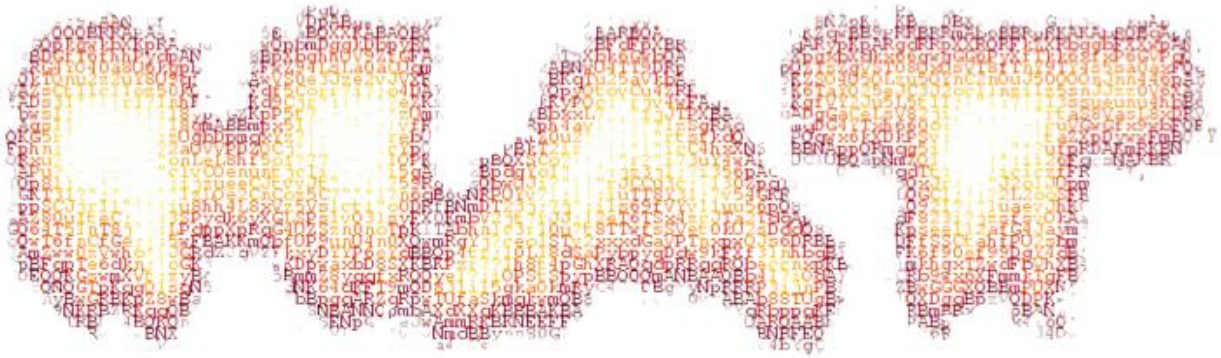


WINDOWZ: Remote Overflow Case Study and Protections

Matteo ‘ryujin’ Memelli

Antonio ‘s4tan’ Parata

Agenda

- Introduzione al SEH
- Exploiting SEH Overwrite
- Demo
- Windows Protections
- /SafeSEH + /GS internals
- Conclusioni
- Riferimenti e Ringraziamenti

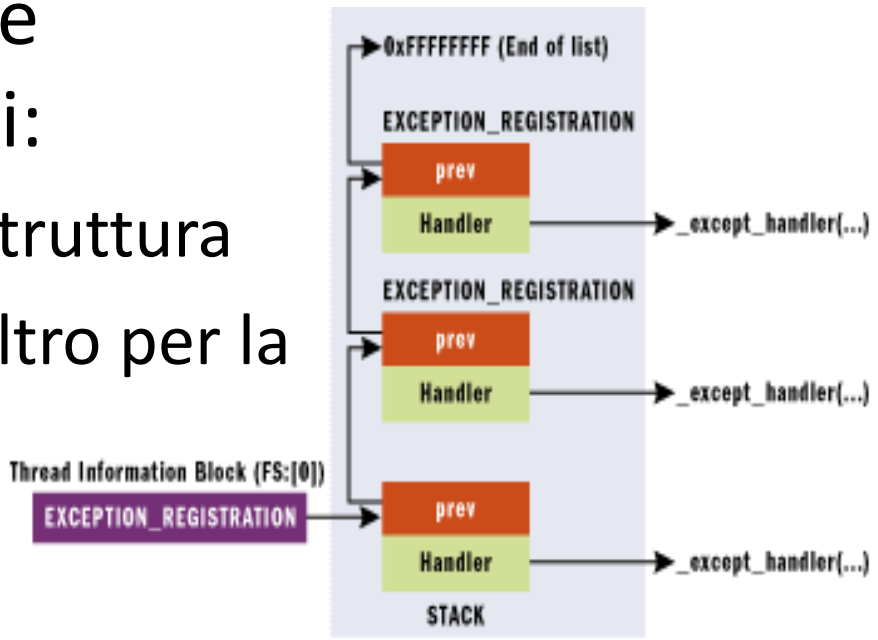
Introduzione al SEH

- Costrutto software utilizzato per catturare e gestire eventuali eccezioni sia a livello hardware che a livello software
- Esempio di utilizzo:

```
__try
{
      __asm
      {
         xor eax, eax
         call eax // Causa l'exception
      }
 }
 __except(MyExceptionHandler()) // mio filtro
 {
      printf("Ops…"); // codice dell'eccezione
 }
```

Introduzione al SEH

- La gestione delle eccezioni avviene utilizzando una lista di strutture (EXCEPTION_REGISTRATION_RECORD) memorizzate sullo stack e contenenti due puntatori:
  - Puntatore alla prossima struttura
  - Puntatore alla funzione filtro per la gestione dell'eccezione



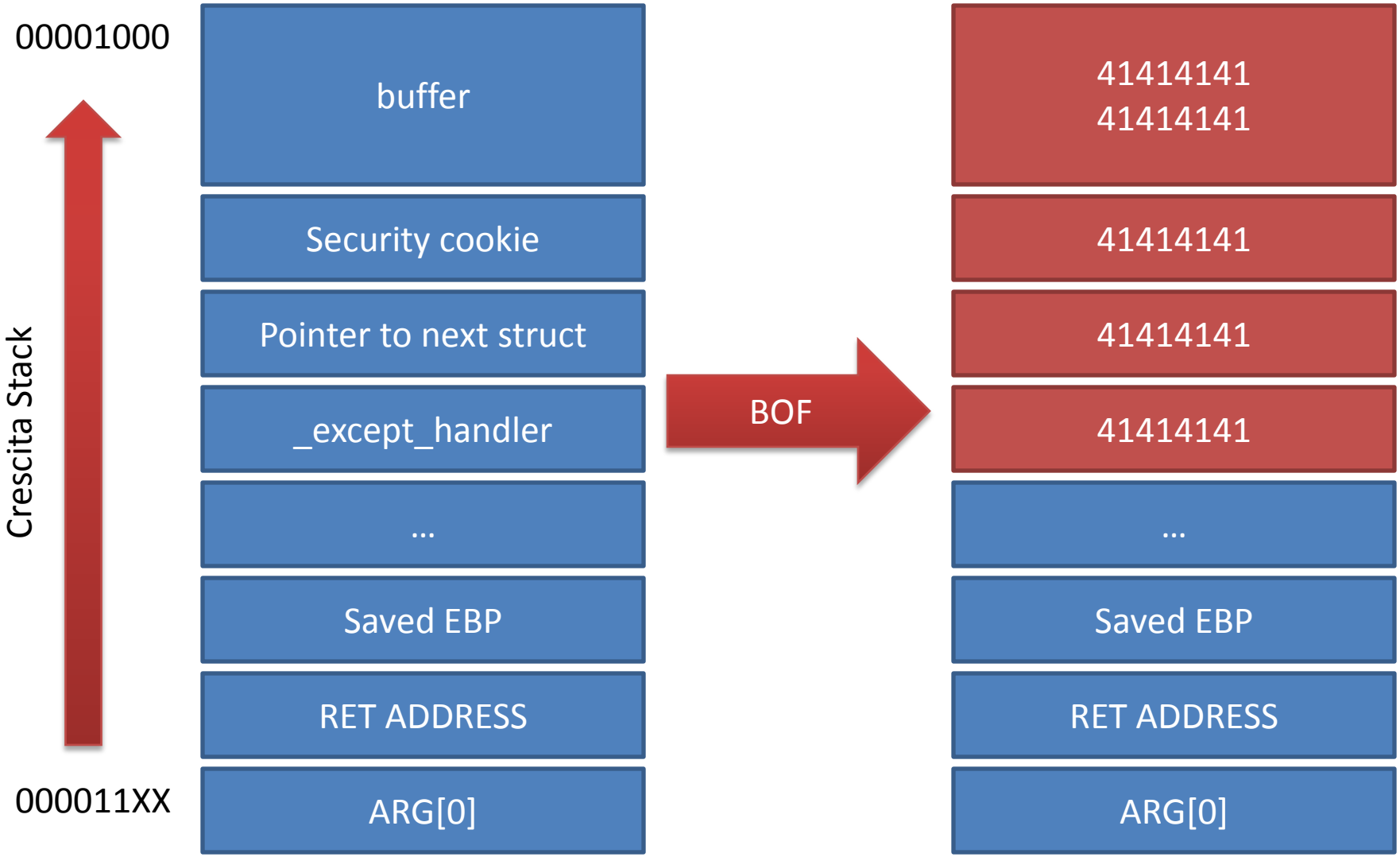
Introduzione al SEH

```
push    ebp
mov     ebp,esp
push    0FFFFFFFEh ← indica se sono all'interno di un blocco try-except
push    offset Test!__rtc_tzz+0x88 (00102398) ← Scope tables
push    offset Test!_except_handler4 (001018c5) ← indirizzo della mia funzione
mov     eax,dword ptr fs:[00000000h] ← indirizzo del primo handler
push    eax ← con questa push completo la struttura EXCEPTION_REGISTRATION_RECORD
sub     esp,8
push    ebx
push    esi
push    edi
mov     eax,dword ptr [Test!__security_cookie (00103000)]
xor     dword ptr [ebp-8],eax
xor     eax,ebp
push    eax
lea     eax,[ebp-10h] ← ottengo l'indirizzo della mia EXCEPTION_REGISTRATION_RECORD
mov     dword ptr fs:[00000000h],eax ← la registro aggiungendola in testa alla catena
```

Introduzione al SEH

```
mov     dword ptr [ebp-18h],esp
mov     dword ptr [ebp-4],0 ← indica che sono all'interno del blocco try-except
xor     eax,eax
call    eax
jmp     Test!B+0x62 (00101062) ← se tutto va a buon fine salto il blocco except
push    offset Test!`string' (00102104)
call    dword ptr [Test!_imp__wprintf (001020b0)]
add     esp,4
xor     eax,eax
ret
```

Introduzione al SEH



Introduzione al SEH

- Dopo l'overflow, viene sovrascritta la struttura EXCEPTION_REGISTRATION_RECORD presente sullo stack con il valore *41414141*.
- Se una AV occorre viene richiamata la funzione all'indirizzo *41414141* (flag /SAFESEH permettendo)

Exploiting SEH Overwrite

- Lo scopo finale è ovviamente fare in modo che EIP ricada all'interno del nostro buffer.
- Per prima cosa otteniamo il controllo di EIP attraverso la sovrascrittura dell'exception handler.
- Il prototipo dell'exception handler è:

```
void _except_handler(struct _EXCEPTION_RECORD *, struct
_EXCEPTION_REGISTRATION_RECORD *, struct _CONTEXT *, void *)
```

per cui quando una AV occorre, sullo stack abbiamo un puntatore alla nostra EXCEPTION_REGISTRATION_RECORD, nello specifico è situato a ESP+8.

Exploiting SEH Overwrite

- Una volta controllato EIP, dato che ESP+8 punta alla nostra EXCEPTION_REGISTRATION_RECORD, la tecnica più usata per ricadere nel nostro buffer è eseguire un codice *pop pop ret*.
- In seguito viene generalmente utilizzato uno short jump per saltare l'exception handler e ricadere nel nostro codice.

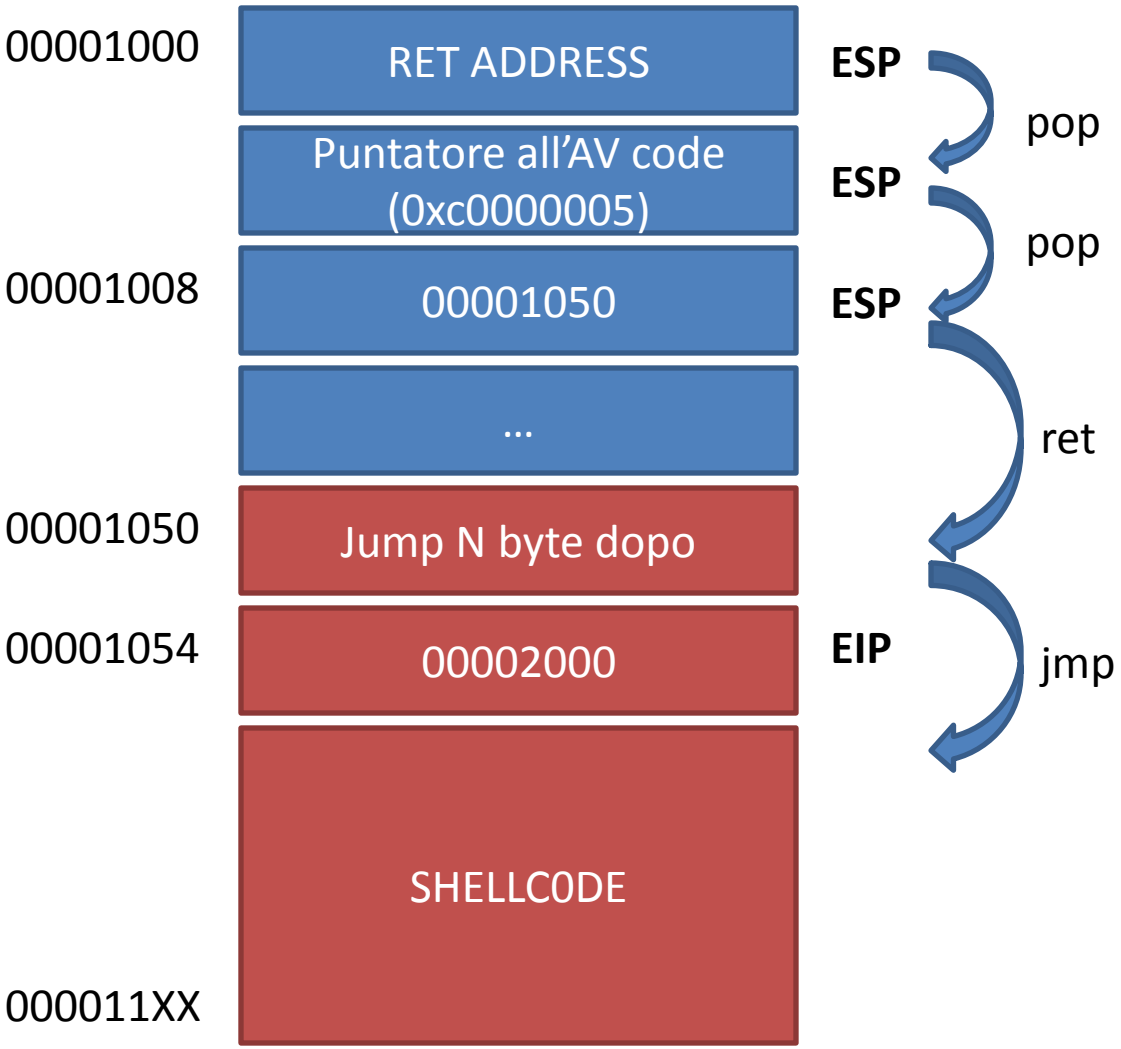
Exploiting SEH Overwrite

- La *pop pop ret* viene usata per far riferimento ad aree di memoria di codice le quali generalmente risultano essere sempre allo stesso indirizzo (flag /DYNAMICBASE permettendo)
- La short jump viene utilizzata perché il suo opcode non necessita del completo indirizzo ma solo del numero di byte da saltare a partire dalla locazione corrente (ci si astrae dall'indirizzo di memoria dello shellcode)

N.B.

Il check sui cookie viene effettuato all'uscita della funzione, per cui il nostro intento è quello di ottenere il controllo prima del check. *Microsoft afferma che prima di chiamare l'exception handler il cookie venga controllato (nel caso di compilazione con /GS + /SAFESEH), ciò è solo parzialmente vero ;)*

Exploiting SEH Overwrite



Una AV viene sollevata:

1. ESP punta a 00001000 ed EIP punta a 00001054 il codice della mia routine di exception.
2. Eseguo le istruzioni pop, pop.
3. ESP punta a 00001008 l'indirizzo dove si trova la prossima struttura EXCEPTOIN_REGISTRATION da me sovrascritta con il l'opcode della short jump.
4. Eseguo la ret, EIP assume il valore 00001050.
5. Eseguo la jump (per saltare il puntatore) ed arrivo nel mio shellc0de.

Exploiting SEH Overwrite

TIME OF DEMO

Windows Protections

- In Visual studio 2008 (il compilatore da noi preso in esame) sono abilitate di default quasi tutte le misure di sicurezza disponibili, in particolare:
  - **Buffer Security Check:** nella linea di comando il flag appare solo se disabilitato con la dicitura /GS-, può essere disabilitato tramite menù
  - **SafeSEH:** abilitato di default, per disabilitarlo aggiungere il flag /SAFESEH:NO nella sezione Link

Windows Protections

- **Randomized Base Address:** crea un binario che viene casualmente rilocato a “load Time” utilizzando la feature ASLR. Abilitato di default (flag /DYNAMICBASE) nella sezione Link
- **Data Execution Prevention (DEP):** previene l’esecuzione di codice residente sullo stack o su heap. Abilitato di default (flag /NXCOMPAT) nella sezione Link

/SafeSEH + /GS internals

- Come già accennato entrambi i flag sono abilitati di default.
- In particolare a seconda del fatto che il flag /GS sia abilitato o meno, vengono invocate le seguenti funzioni:
  - _except_handler4: flag /GS abilitato
  - _except_handler3: flag /GS non abilitato
- Noi vedremo solamente la funzione _except_handler4 essendo _except_handler3 ormai considerata obsoleta.

/SafeSEH + /GS internals

- Quando viene generata un'eccezione, il kernel esegue la funzione ntdll!KiUserExceptionDispatcher che a sua volta chiama la ntdll!RtlDispatchException.
- ntdll!RtlDispatchException chiama per prima la ntdll!RtlCallVectoredExceptionHandlers per richiamare i Vectored Exception Handler
- In seguito se l'eccezione non è ancora stata gestita le Structured Exception Handler vengono esaminate.

/SafeSEH + /GS internals

- Vengono invocate le funzioni ntdll!RtlpGetRegistrationHead (che esegue il seguente codice mov eax,dword ptr fs:[00000000h], ovvero ottiene l'indirizzo della struttura EXCEPTION_REGISTRATION_RECORD in testa alla lista) e ntdll!RtlpGetStackLimits per ottenere i limiti dello stack.
- Ciò viene fatto perché la ntdll!RtlDispatchException controlla che il mio EXCEPTION_REGISTRATION_RECORD sia presente sullo stack e l'handler fuori dallo stack.

/SafeSEH + /GS internals

call ntdll!RtlpGetStackLimits (76fa2923) ← mette in [ebp+8] il limite superiore e in [ebp-8] il limite inferiore
call ntdll!RtlpGetRegistrationHead (76fe118d)
mov ebx,eax
xor edi,edi
cmp ebx,0FFFFFFFFh
je ntdll!RtlDispatchException+0x114 (76f894b6)
cmp ebx,dword ptr [ebp+8] ss:0023:0016f5c0=00166000 ← il puntatore al prox record deve stare sullo stack
jb ntdll!RtlDispatchException+0x110 (76f897da)
lea eax,[ebx+8]
cmp eax,dword ptr [ebp-8]
ja ntdll!RtlDispatchException+0x110 (76f897da)
test bl,3
jne ntdll!RtlDispatchException+0x110 (76f897da)
mov eax,dword ptr [ebx+4] ← controllo che l'exception handler non risieda sullo stack
cmp eax,dword ptr [ebp+8]
jb ntdll!RtlDispatchException+0x72 (76fa29b8) ← se vero salto a RtlIsValidHandler perché si trova fuori dallo stack
cmp eax,dword ptr [ebp-8]
jb ntdll!RtlDispatchException+0x110 (76f897da) ← Errore il codice è sullo stack
push eax
call ntdll!RtlIsValidHandler

/SafeSEH + /GS internals

- Se tutto è andato a buon fine viene chiamata la ntdll!RtlIsValidHandler con parametro l'indirizzo della funzione dell'handler.
- In ntdll!RtlIsValidHandler viene chiamata la funzione ntdll!RtlLookupFunctionTable che ritorna l'elenco delle funzioni registrate.
- Se il binario è compilato con il flag /SAFESEH:NO tale funzione ritorna null, ciò significa che non esiste una lista di exception handler registrati, per cui l'exception handler corrente è considerato safe e quindi viene invocato (dopo ulteriori check non relativi al /SAFESEH e qui non trattati).

/SafeSEH + /GS internals

- Se invece il binario è compilato con il flag /SAFESEH viene ritornata la tabella dei seh registrati (indicata con il simbolo __safe_se_handler_table, in forma di RVA). I seh registrati sono hard coded all'interno del formato PE.
- L'exception handler corrente viene confrontato con quelli della lista per verificarne la validità (il confronto viene fatto sui RVA).
- Se l'handler non è nella lista ntdll!RtlIsValidHandler ritorna FALSE e viene invocata la ntdll!RtlInvalidHandlerDetected.

/SafeSEH + /GS internals

- Se l'handler è nella lista viene invocata la ntdll!RtlpExecuteHandlerForException che si compone di due fasi:
  - Si cerca un filtro che gestisca l'eccezione. Ciò viene fatto chiamando la MSVCR90!_EH4_CallFilterFunc per l'invocazione dei filtri ovvero gli exception handler (le informazioni vengono ricavate dalla Scope Table)
  - Se un filtro esiste (EAX=1), richiamare tutti i body __except dei vari EXCEPTION_RECORD attraversati. Ciò viene fatto dalla MSVCR90!_EH4_GlobalUnwind. Dopo di chè viene chiamata la ntdll!NtContinue per continuare l'esecuzione.
- È in questa fase che viene invocata la _except_handler4, ed è qui che i cookie dello stack frame vengono controllati.

Conclusioni

- Il connubio SafeSEH e Cookie pone dei seri limiti alla possibilità di “exploitare” un programma.
- Esistono comunque delle tecniche per bypassare SafeSEH (e cookie):
  - Abusare di un handler esistente in una delle librerie disponibili
  - Utilizzare (se esistono) delle librerie caricate in memoria non compilate con /SAFESEH
  - Far riferimento ad una zona di memoria non associata ad alcun modulo (l’effetto ottenuto è lo stesso del punto precedente)
- Nel caso in cui tutte le misure di protezione fossero abilitate (ovvero ASLR e NX) ottenere il controllo risulta un compito veramente difficile, ma non impossibile ;)

Riferimenti e Ringraziamenti

- Chris Anley et All, "*The Shellcoders Handbook – Second Edition*"
- Mario Hewardt and Daniel Pravat, "*Advanced Windows Debugging*"
- Michael Howard and David LeBlanc, "*Writing Secure Code for Windows Vista*"
- MSDN Site sulla Structured Exception Handling, http://msdn.microsoft.com/en-us/library/ms680657(VS.85).aspx
- MSDN Site sul /SAFESEH flag, http://msdn.microsoft.com/en-us/library/9a89h429(VS.80).aspx
- SEH Explained, http://www.openrce.org/articles/full_view/21
- Mark Down, Neel Mehta and John McDonald, "*Breaking C++ Applications*". BlackHat 2007

Thanks to:

KJK::Hyperion per la parte di SEH Internals