


# MMOGLider

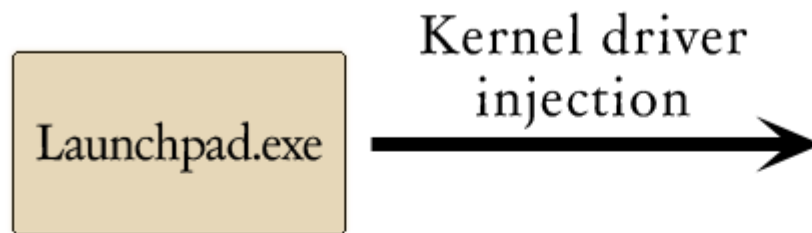
Presented by  
Alex Ten and Dan Lake  
for CS592 Computer  
Security Practicum

# Glider System Overview

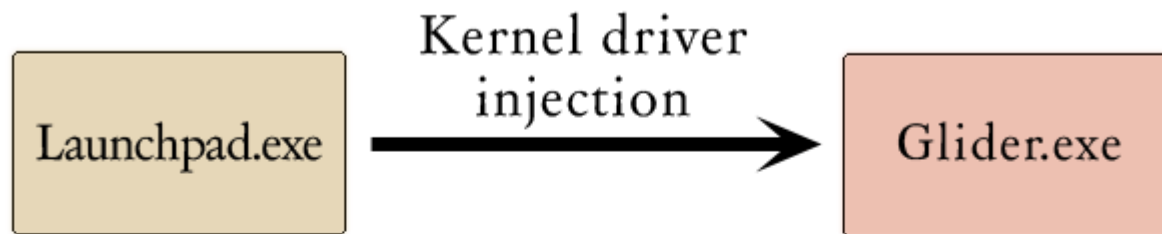


Launchpad.exe

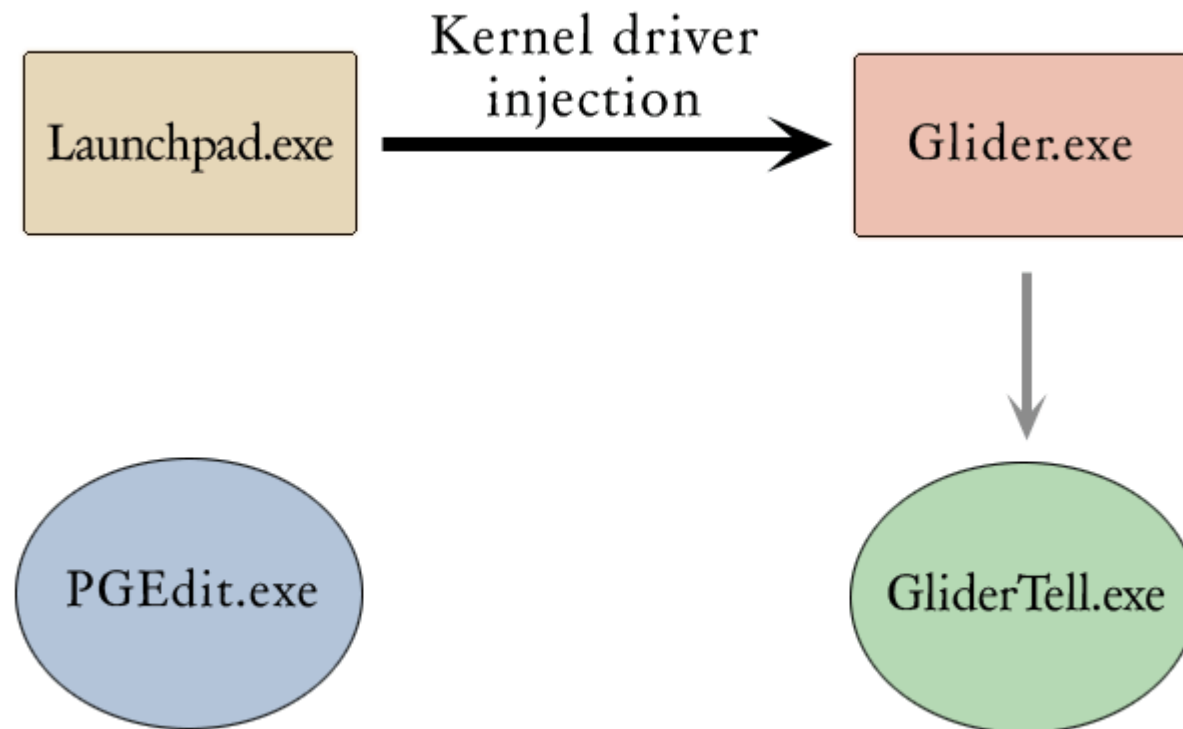
# Glider System Overview



# Glider System Overview



# Glider System Overview



# Into the cheat, Launchpad.exe

The screenshot displays a debugger interface with assembly code on the left and two windows on the right.

**Assembly Code:**

```
.text:110C6F65      call    sub_110C74D3
.text:110C6F6A      jmp     loc_110C70AB
.text:110C6F6F      ; -----
.text:110C6F6F      ; CODE XREF: .text:110C6F6F
.loc_110C6F6F:
.text:110C6F6F      push   11D4Ch
.text:110C6F74      jmp     loc_110C70A3
.text:110C6F79      ; -----
.text:110C6F79      ; CODE XREF: .text:110C6F79
.loc_110C6F79:
.text:110C6F79      call   sub_110C6D25
.text:110C6F7E      jmp     loc_110C70AB
.text:110C6F83      ; -----
.text:110C6F83      ; CODE XREF: .text:110C6F83
.loc_110C6F83:
.text:110C6F83      cmp    ds:12B90h, edi
.text:110C6F83      jnz   short loc_110C6F95
.text:110C6F89      push  11D1Ch
.text:110C6F8B      jmp     loc_110C70A3
.text:110C6F90      ; -----
.text:110C6F95      ; CODE XREF: .text:110C6F95
.loc_110C6F95:
.text:110C6F95      push  dword ptr ds:12B58h
.text:110C6F9B      mov   ecx, [esi+0Ch]
.text:110C6F9E      push  11D04h
.text:110C6FA3      call  nullsub_1
.text:110C6FA8      mov   eax, ds:12B58h
.text:110C6FAD      push  dword ptr [eax+18h]
```

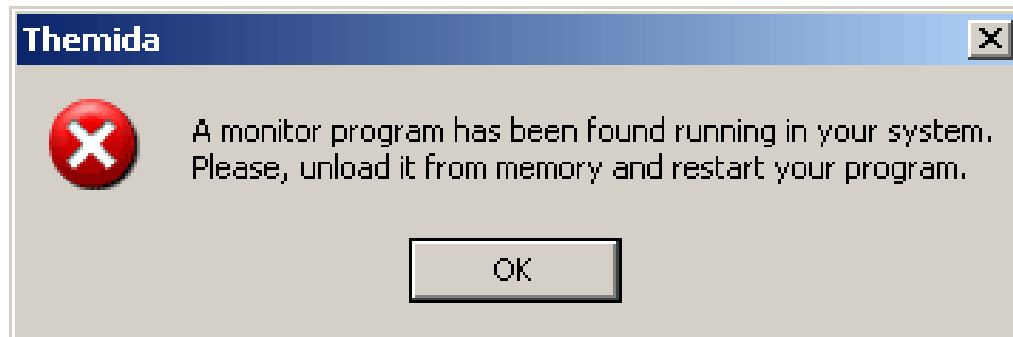
**Names window:**

Name	Address
aDrawimage	1109AD92
aDriverNixedFro	110C8A82
aDriverStarting	110C8895
aDriverUnloaded	110C8761
aDruid	1109D84A
aDs	1109DF69
aDt	1109DF6C
aDtype	1109D286
aDu	1109DF6F
aDumpdebug	1109D4EE

**Strings window:**

Address	Length	Type	String
... .text:11...	0000000D	C	SetWindowPos
... .text:11...	0000000E	C	SetWindowText
... .text:11...	00000011	C	SetWindowsHookEx
... .text:11...	00000017	C	Shadow table -> 0x%08x
... .text:11...	00000007	C	Shaman
... .text:11...	00000008	C	ShiftState
... .text:11...	00000005	C	Show
... .text:11...	00000012	C	ShowConfiguration
... .text:11...	00000008	C	ShowDialog
... .text:11...	00000009	C	ShowHelp

# Themida



# Themida

- A commercial packer, originated from Xtreme-Protector
- Main Features:
  - Patching, i.e. removing certain commands from the original file and adding them to the packer's code.
  - Mutation, i.e. real and garbage instructions that are added are getting mutated.



# Themida

Target File before protecting

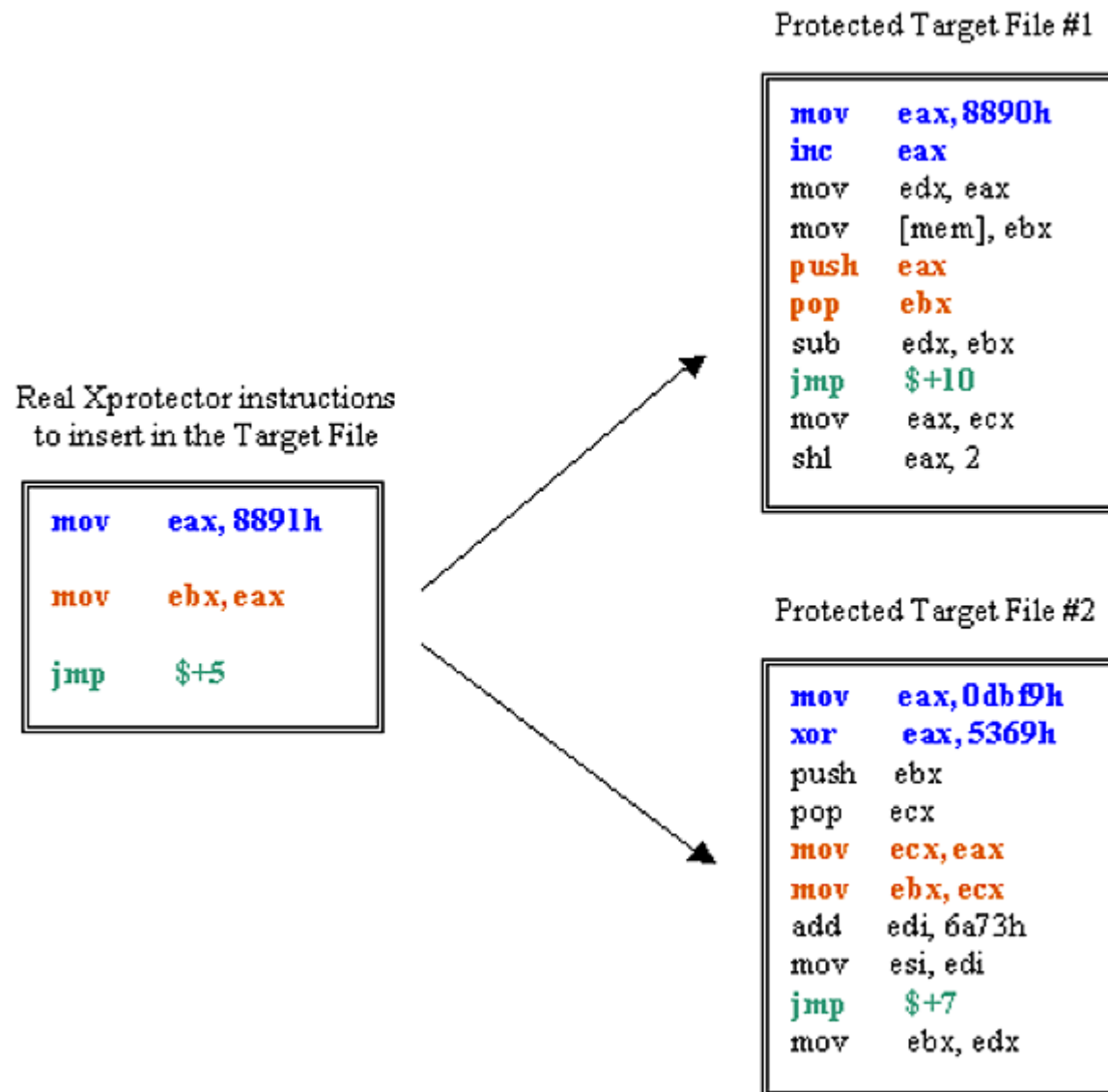
```
mov    eax, ecx
shr    eax, 2
push   eax
mov    eax, fs:[18h]
mov    eax, [eax+30h]
mov    eax, [eax+08h]
call   eax
mov    [ebx], eax
mov    esi, eax
mov    ecx, 100h
rep    movsd
```



Target File after protecting

```
mov    eax, ecx
shr    eax, 2
garbage code
garbage code
garbage code
garbage code
garbage code
garbage code
mov    esi, eax
mov    ecx, 100h
rep    movsd
```

# Themida



# Themida

- Run-time decryption, i.e. all decryption is happening at run-time, block by block. There are 3 'flavors' of decrypting/encrypting:
  - Poly-Decrypting, the engine generates thousands of combinations in the way that the code is decrypted at runtime.
  - DrX-Decrypting, the engine decrypts the code using Debug Control registers, which makes debugging virtually impossible.
  - Thread-Decrypting, the engine decrypts the code using more than one thread.

# Themida

- Anti-debugger techniques that detect/fool any kind of debugger
- Anti-memory dumpers techniques for any Ring3 and Ring0 dumpers
- Different encryption algorithms and keys in each protected application
- Automatic decompilation and scrambling techniques in target application
- Anti-disassembler techniques for any static and interactive disassemblers
- Anti-monitors techniques against file and registry monitors
- Random garbage code insertion between real instructions
- Advanced Entry point protection

# Custom Classes

Using provided API you can program your own class behavior (woo-hoo).

That's probably one of the reasons why Launchpad.exe (which loads those classes) is packed, but not obfuscated.

Any class has to be written in C#.

# Into the cheat, Glider.exe

The screenshot displays the Immunity Debugger interface for the process Glider.exe. The main window shows a memory dump of the 'Sparky\_\_' variable, with addresses ranging from 110F9F0D to 110F9F2A. The data is shown in hexadecimal and ASCII. The Names window on the right lists various string literals found in the memory, such as 'alSj', 'aSS', 'aFz', 'aXh5', 'aSlS', 'aHE', 'aVsi', 'aRsts', 'aBaL', and 'aTsbI'. The Strings window at the bottom shows a list of strings with their addresses, lengths, and types, including 'Mb406', 'PHkXCZ[', '{U4c.', 'U{=24V', 'VNt4A', 'M^,q\t8', 'd\rs~Cx', 'M^,q\t', 'Qg3\w1Bd', and 'KFBdb'.

Name	Address
alSj	11105C37
aSS	11105C7D
aFz	11106441
aXh5	1110666E
aSlS	11106674
aHE	11106A2D
aVsi	11107159
aRsts	11107258
aBaL	1110725E
aTsbI	111073D5

Address	Length	Type	String
110F9F0D	00000005	C	Mb406
110F9F0E	00000007	C	PHkXCZ[
110F9F0F	00000005	C	{U4c.
110F9F10	00000006	C	U{=24V
110F9F11	00000006	C	VNt4A
110F9F12	00000006	C	M^,q\t8
110F9F13	00000006	C	d\rs~Cx
110F9F14	00000005	C	M^,q\t
110F9F15	00000005	C	Qg3\w1Bd
110F9F16	00000005	C	KFBdb

```

:110EF2EB: Can't find name (hint: use manual arg)
:110EF2C2: Can't find name (hint: use manual arg)
:110EF2B5: Can't find name (hint: use manual arg)
:110EF334: Can't find name (hint: use manual arg)
:110EF33E: Can't find name (hint: use manual arg)
7FDB: Can't find name (hint: use manual arg)
:110EEE4: Can't find name (hint: use manual arg)
:110E550: Can't find name (hint: use manual arg)

```

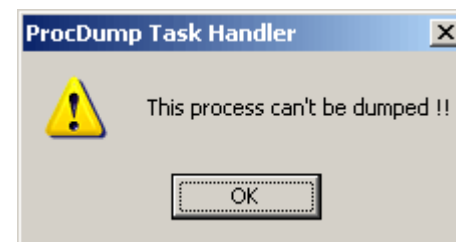
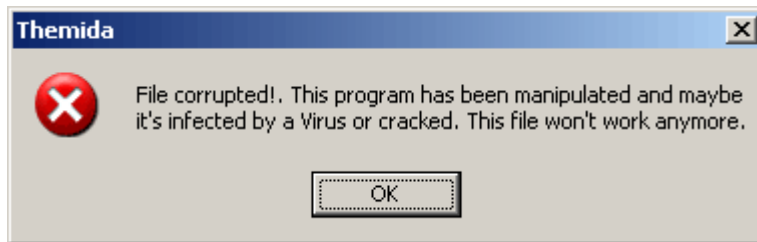
# Deobfuscating Glider.exe

It wasn't clear whether Glider.exe uses Themida's encrypting tool, anything else or both (or more than just two tools).

Dumping strings from the binary didn't help, since there was a lot of junk and all un-obfuscated strings seemed unrelated to a packer(s) name.

# Deobfuscating Glider.exe

An attempt to dump/unpack the executable yielded this:



The only thing that was verified was the fact that the binary was packed with something.



# Deobfuscating Glider.exe

Since there was no clue in Glider.exe for its deobfuscation, we went back and decided to review strings dump for Launchpad.exe. After awhile, we've found this:

```
iphttpapi.dll  
ISXWardenLink.dll  
gdi32.dll  
psapi.dll  
User32.dll  
ntdll.dll  
Advapi32.dll  
Userenv.dll  
mscorlib  
DotfuscatorAttribute  
GObject  
Glider.Common.Objects  
GUnit  
GLocation  
GItemHelper  
GPlayer  
GGameCamera  
GPartyHelper  
PartyBuff
```

# Dotfuscator

.NET specific commercial obfuscator that tampers the binary and makes it **very** hard to decompile.

Dotfuscator Community Edition comes for free with any version of Visual Studio .NET, except Express Edition, and is to date the most common obfuscator for .NET applications.

# Dotfuscator

It uses the Overload Induction technique to rename all program identifiers into small meaningless names, which are divided into colliding sets.

# Dotfuscator

```
float getCheckingBalance(int acctnmbr)
void setCheckingBalance(int acctnmbr, float value)
int getCheckingStatus(int acctnmbr)
```

becomes

```
float a(int a)
void a(int a, float b)
int b(int a)
```

if standard Overload Instruction is used

# Dotfuscator

```
float getCheckingBalance(int acctnmbr)
void setCheckingBalance(int acctnmbr, float value)
int getCheckingStatus(int acctnmbr)
```

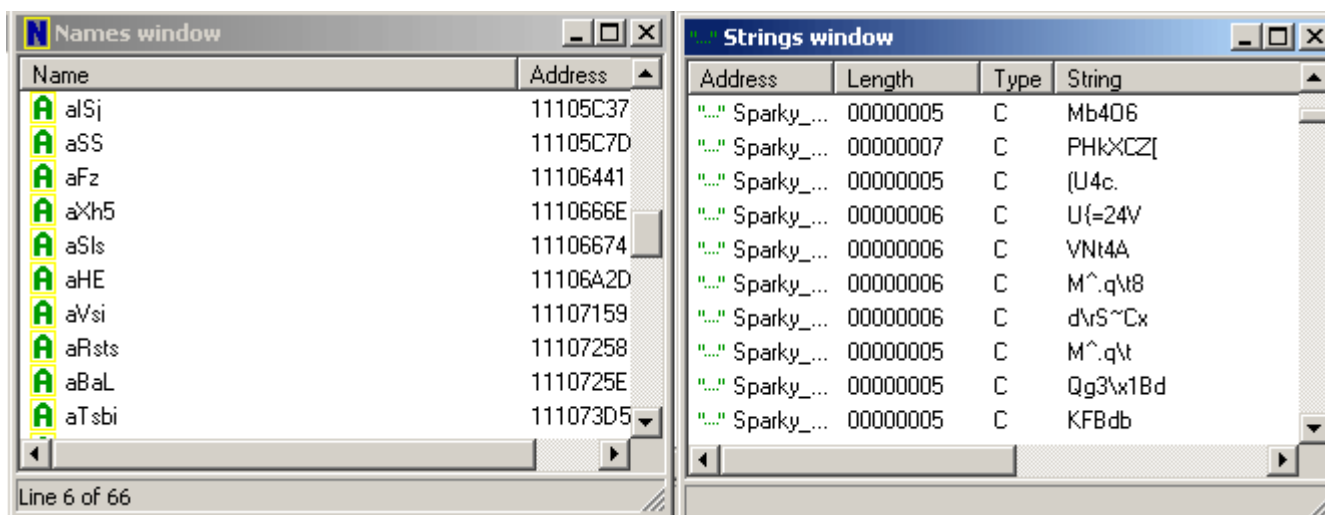
becomes

```
float a(int a)
void a(int a, float b)
int a(int a)
```

if Enhanced Overload Instruction is used

# Dotfuscator

The fact that Glider.exe doesn't have any function called "a" or "b" arouse suspicion that it was obfuscated not only with Dotfuscator. Most probably an addition tool was involved (such as Themida).



# Applications used

- Debuggers/Decompilers - IDA Pro, OllyDBG, SoftICE (completely broken on XP SP2), Kernel Debugger (LiveKD version of it)
- Unpackers - PeiD, procDump, XprotStripper
- Process monitors - SysAnalyzer, FileMon, RegMon, Axe 3