Cracking & Keygen

"If there's a will, there's a way"



Disclaimer

I am legally obligated to inform you: You shouldn't pirate, support the developers, this presentation is for educational purposes only

I am not a professional

I am not a Ph.D

All knowledge in this presentation is off the top of my head, not all of it may be correct.

THE GOOD OLD DAYS

Buy game disk Play game Enjoy game, tell friend about it Friend asks for game "Yeah I can just make a copy" Copy le game to blank disk Free game for friend!!!!!!!



NOW

DRM

"Please enter a serial key" "Please activate windows" "You MUST open Steam to play this game"

"Activate MSWord NOW!!!"

Denuvo

Copy protection Please pay \$200 for our screen reader because you are blind



NOW (ALSO)

Software cracking Game cracking Keygenning Reverse engineering WAREZ



	EXPRESSE ALL HATE & II	S NSULTS	
	DEPRESSE	8	
	IMPRESSE	S SS THE SEA	
Cre	-Game Info Red Dead Red -Protecti Digital.AI (form Socialclub + edits to Mr_Gold talclub emulator	o- emption 2 on- mer Arxan) + License berg for his r and game	
		7	



Simple, real-world example: Creeper World 3





If you played Flash games on ArmorGames, Kongregate, etc. in your youth, you might recognize this game.

Vulnerable and easy target of CRACKING:

- Openly accessible "re-download" page that allows you to download a key-protected game executable
- Unity Engine game, meaning the game logic is easily reversed engineered due to the ease of decompilation of C# scripts

reeper World 3		– 🗆 X
	Creeper World III Arc Eternal Enter your product key below. If you don't have a key, visit knucklecracker.com for purchase info.	
	Product Key	



Oh no! Product key!

Now I have to pay for the game!

Unless I bust out my 1337 h4xOr skillz and crack the game...



reeper World 3	
	Creeper World III Arc Eternal Enter your product key below. If you don't have a key, visit knucklecracker.com for purchase info.
	Product Key
6	ð
19	

How 2 crack? Think about it....

– 🗆 🗙

A (game) program is code that is executed by your computer.

Unfortunately, that code has truthfully determined that there is no product key on record, hence presenting us with this screen.

Know how it works, know how to circumvent it (in regards to copy protection)



There are two aspects of the game that are targets of knowing, which if known, can result in copy protection circumvention.

(i) The process by which the Product Key screen is determined to be shown, i.e. the game code that determines if there is a valid key on record,

(ii) The algorithm behind verifying the Product Key and any methods, if possible and tractable, of generating the key without knowing developer secrets.

The Process of Knowing - Reverse Engineering

Reverse Engineering (in the context of code): Figuring out what a piece of code does without the original source the code was generated from or any documentation or insights into what it is supposed to do and how it does it.

In this case: understanding the game's behavior through deduction from the compiled code available in the executable without access to the game's source code.

Also in this case: not as hard and we're cheating a bit by "reverse engineering" C# Intermediate Language

The Process of Knowing - What Must Be Known Beforehand What kind of code is being executed? Ultimately, this ends up being x86(_64) machine code on most

personal computers nowadays. But in some cases (this game being one of them), we are given a higher-level language like **JVM** (Java) **Bytecode** or **CIL** (Common Intermediate Language, Microsoft's version of JVM bytecode).

What does the code being executed actually do (at the micro level)?

Computers (including abstract machines executing CIL) execute instructions, which are well-defined in a specification and you can find out what they do and how they are encoded in a manual somewhere.



Giveaway of an Essential Fact To Be Known - What kind of code?

If you have UI that looks like this, it's probably a Unity game

The game logic is encoded in a .NET Assembly, meaning that Unity Engine essentially executes compiled C#

Essential Fact To Be Known - Unity / C# / CIL

This fact can also be verified by looking at the game's files. This gives it away, for sure.

dan@Dan-PC:/mnt/s/CW3Cracka/CW3_Data	Managed\$ 1s		
Assembly-CSharp.dll	Boo.Lang.dll	System.dll	UnityEngine.Networking.dll
Assembly-CSharp-firstpass.dll	Mono.Security.dll	System.Xml.dll	UnityEngine.UI.dll
Assembly-UnityScript.dll	mscorlib.dll	System.Xml.Linq.dll	UnityEngine.xml
Assembly-UnityScript-firstpass.dll	System.Core.dll	UnityEngine.dll	UnityScript.Lang.dll
dan@Dan-PC:/mnt/s/CW3Cracka/CW3_Data	Managed\$ file As	sembly-CSharp.dll	
Assembly-CSharp.dll: PE32 executable	e (DLL) (console) I	ntel 80386 Mono/.Net	assembly, for MS Windows

Assembly-CSharp.dll is the main .NET Assembly (like a dynamically linked library file but for CIL and the .NET Runtime) in the game that encodes game logic, which probably also includes the product key check.

The Process of Knowing - Code Analysis in Order to Know

How to figure out what the game does: look at the CIL in the game binary (Assembly-CSharp) and decode and understand the instructions to form a bigger picture of

W	hat they	do elasta do elasta elasta	C 06 2A 00 00 36 02 02 7 02 2B C8 02 00 06 38 2 00 02 02 7B B7 02 00 D 02 7B BF 02 08 04 02	22 12 6 6 6 6 6 4 70 BE 92 66 64 2 A 68 66 13 35 7 66 8 4 2 6 6 6 0 7 66 60 11 82 78 BA 20 66 64 35 68 86 66 66 6 6 80 6 80 6 28 A 6 92 6 6 45 7 8 9 2 2 6 94 35 7 8 92 6 6 4 35 7 8 96 6 2 6 7 8 B 6 2 8 0 6 4 2 2 6 6 5 2 6 4 4 2 A 60 60 6 8 6 2 78 BE 6 2 6 9 4 6 F 13 6 9 6 0 A 2 5 5 6 1 9 0 A 6 8 1 2 1 3 6 F 6 1 6 0 6 4 2 2 6 F 12 8 3 A 5 A 7 1 B 6 2 6 9 4 7 8 B 6 2 8 6 6 5 1 3 6 9 6 A 6 7 7 8 B 6 2 A 6 6 5 7 8 B 6 2 A 6 6 6 7 8 B 6 2 A 6 6 7 8 B 6 2 A 6 7 8 B 6 7 8 8 6 7 8 B 6 7 8 B 6 7 8 B 6 7 8 B 6 7 8 B 6 7 8 B 6 7 8 B 6 7 8 B 6 7 8 B 6 7 8 B 6 7 8 B 6 7 8 B 6 7 8 8 6 7 8 B 6 7 8 8 8 8
List of	CIL instructions		文A 1 language	e V 82 78 BE 82 08 84 6F 13 90 08 8A 28 52 01 90 0A 28 52 01 90 0A 9C 12 82 28 6F 01 90 0A 44 63 00 00 00 02 78 BE 82 00 00 00 02 78 BD 62 00 04 39 0C 00 00 02 78 BD 62 00 04 14 28
Article Talk	1	Read E	Edit View history Tool	02 78 8C 02 00 04 28 5F 00 00 4 3A 13 00 02 78 8B 02 00 40 02 78 8C 02 78 8C 02 00 04 02 78 8C 02 00 04 02 17 6F 08 00 00 IS 🛩 EF 00 00 00 02 78 8F 02 00 04 22 00 00 00 06 40 24 00 00 06 02 02 78 87 02 00 04 02 78 8C 02 08 04 6F
From Wikiped	dia, the free encyclopedia			00 12 03 28 6F 01 00 04 04 22 5F 12 83 34 54 70 8F 02 08 04 02 78 8E 02 00 04 02 78 8E 02 00 04 05 07 8E 02 00 04 65 48 01 78 88 02 00 04 06 28 F6 02 00 06 6F 04 01 06 04 02 78 8F 02 00 04 02 78 70 2 00 04 02 78 8F 02 00 04 04 13 04 12 04 28 F6 01 00 03 44 63 00 00 06 02 78 8F 02 00 04 04 27 77 70 2 00 04 02 78 50 02 78
Main artic	de: Common Intermediate Languag	e		02 78 80 02 00 04 02 6F D4 1F 00 06 02 78 88 02 00 04 14 28 14 00 80 0A 39 23 00 00 00 02 78 8C 02 00
This is a list of Opcode a Base inst Object mo	of the instructions in the instruction abbreviated from operation code is <i>ructions</i> form a Turing-complete ins odel instructions provide an implem	set of the Common Intermediate Language bytecode. the portion of a machine language instruction that specifies the operation to b truction set. entation for the Common Type System.	pe performed.	B - S(TAP), w ProveR BREA, ONE MoveR BREA, TEPM MoveR BREA, TEP
Opcode \$	Instruction +	Description	♦ Type of instruction	novem (REG, TERM COMP CREG, No. 040
0x58	add	Add two values, returning a new value.	Base instruction	BL LE INDIAN
0xD6	add.ovf	Add signed integer values with overflow check.	Base instruction	PRZNT RESULT
0xD7	add.ovf.un	Add unsigned integer values with overflow check.	Base instruction	STOP
0x5F	and	Bitwise AND of two integral values, returns an integral value.	Base instruction	A ZO W
0xFE 0x00	arglist	Return argument list handle for the current method.	Base instruction	NE OL 17 ³
0x3B	<pre>beq <int32 (target)=""></int32></pre>	Branch to target if equal.	Base instruction	TERM DS 1
0x2E	<pre>beq.s <int8 (target)=""></int8></pre>	Branch to target if equal, short form.	Base instruction	END
0x3C	<pre>bge <int32 (target)=""></int32></pre>	Branch to target if greater than or equal to.	Base instruction	aboution lade and lede
0x2F	<pre>bge.s <int8 (target)=""></int8></pre>	Branch to target if greater than or equal to, short form.	Base instruction	Mnemeril Or Ander Hy Miles (2).
0x41	bge.un <int32 (target)=""></int32>	Branch to target if greater than or equal to (unsigned or unordered)	Base instruction	Coll STOP 3



The Process of Knowing - Tools to Aid in Knowing



dnSpy(Ex) - Popular tool for disassembly, decompilation, debugging, and analysis of .NET binaries and libraries (This is basically easy mode, more on that later)



Ghidra - RE tool developed by the NSA. Supports disassembly, decompilation, analysis of many popular executable/library formats and instruction sets



IDA Pro - RE tool by Hexrays. Costs too much.

The Process of Knowing - Tools to Aid in Knowing - dnSpy

Disassemble

Token: 0x06000FF6 RID: 4086	RVA: 0x00096D50 File	Offset: 0x00094F50
ethod public hidebysig stati	c	
string key,		
[out] uint32& id,		
[out] uint32& randomNu	mber	
// neader Size: 12 Dytes		
// Loop Non Size: 181 (0x65) 0	9003C8 BTD: 873	
// Localvarsig Token: 0x11	000508 RID: 672	
least init (
[1] uint8[]		
[2] int32		
[3] uint32		
[4] int 32		
[5] uint32.		
[6] uint32.		
[7] uint32		
/* 0X00094F5C 02	-/ IL_0000: Idang.0	
/* 0X00094F5D 28FD0F0006	7 IL_0001: call	uints[] Encoder::Decode(string)
/* 0x00094F62 0A	*/ TL_0000: SCIOC.0	"Comparent To dd o Doors 1 opportunition of Tabl
/* 0x00094F65 /200001/0	*/ TL_000/: Idstr	Supportingiebevelopersweweedit
/* 0v00094F08 00	*/ TL 0000: coll	wint%[] PC4. Ciphon(staing wint
/* 0x00034105 2014100000	*/ TL 0012: stlos 1	utito[] kc4ctpier(scring, utit
/* avagagatese a3	*/ TL 0012: Jdagg 1	
/* 0x00094F70 16	*/ TL 0014. 1dc i4 0	
/* 0x00094F71 54	*/ TL 0015: stind i4	
/* 0x00094F72 16	*/ TL 0016: 1dc. 14.0	
/* 0x00094E73 0C		
/* 0x00094F74 381B00000	*/ TL 0018: br	
// loop start (head: IL 00		
/* 0x00094F79 07		
/* 0x00094F7A 16		
/* 0x00094F7B 08		

Decompile

pul	plic static bool CheckKey(string key, out uint id, out uint randomNumber)
(
	<pre>byte[] array = Encoder.Decode(key);</pre>
	<pre>byte[] array2 = RC4.Cipher("SupportIndieDevelopersWeNeedIt", array);</pre>
	id = 0U;
	for (int i = 0; i < 32; i++)
	<pre>uint num = (uint)((uint)Encoder.ReadBit(array2, 0 + i * 2) << i);</pre>
	id = num;
	randomNumber = 0U;
	for (int j = 0; j < 32; j++)
	<pre>uint num2 = (uint)((uint)Encoder.ReadBit(array2, 1 + j * 2) << j);</pre>
	randomNumber = num2;
	}
	uint num5 = (uint)(((int)Encoder.Keadbyte(array2, 8) << 8) (int)Encoder.Keadb
	uint num4 = 10;
	10 /= 890;
	return num4 % 890 == 00 && (uint)Encoder.Digest(array2, 8) == num3;

Debug

e(array2, 9))

B	🧾 C# 🔄 🦻 🤆 🕨 Continue III 🔳 🕚	↓ ‡ ♀ 1 A				
ĸ	Locals					
	Name	Value				
	🔗 kev					
	🔗 id	0x00000000				
	andomNumber	0x00000000				
	array					
	Array2					
		0x00000000				
	🤗 num	0x00000000				
	01	0x00000000				
	🤗 num2	0x00000000				
	🤗 num3	0x00000000				
	🤗 num4	0x0000000				
	<pre>incoder ×</pre>					
	<pre>4 5 public class Encoder 6 { 7 public static void Validate() 7 9 Debug.tog("Validate Start"); 10 Debug.tog("Validate Finish"); 11 Debug.tog("Validate Finish"); 12 Debug.tog("Validate Finish"); 13 Debug.tog("Validate Finish"); 14 Debug.tog("Validate Finish"); 15 Debug.tog("Validate Finish"); 16 Debug.tog("Validate Finish"); 17 Debug.tog("Validate Finish"); 18 Debug.tog("Validate Finish");</pre>					
	12 13 13 mublic static bool CharkWay/steing kay out wint id out wint randombumbar)					
	14 14					
9	<pre>byte[] array = Encoder.Decode(key);</pre>					
	<pre>16 byte[] array2 = RC4.Cipher(75</pre>	SupportIndieDevelopersWeNeedIt", array);				
	10 = 60;					
	19 {					
	20 uint num = (uint)((uint)E 21 id = num;					
	22 } 23 randomNumber = 8U:					
	24 for (int j = 0; j < 32; j++)					
	<pre>25 { uint num2 - (uint)((uint)Encoder.ReadBit(array2, 1 + j * 2) << j); 27 randomNumber = num2; 28 }</pre>					
	100 % -					
	Call Stack					
	Nama					
	Arramble-CSham dillEncoder CheckKeu(ctring key, out unit id, out unit randomNumber) (II=0x0000)					
	Assembly Company and Contracting (Charge States (Ch					
	Assembly-CSharp.dlllkeyEntryGUI.ShowKeyEntryU (ILx0x04A3)					
	Assembly-CSharp.dll!KeyEntryGUI.OnGU() (Lx0x0016)					

The Process of Knowing - Tools to Aid in Knowing

Disassemble - Encoded instructions too hard to read? Turn them into human-readable mnemonics (assembly)





The Process of Knowing - Tools to Aid in Knowing Decompile - CIL assembly too hard too read? Turn it into a high-level code representation

<pre>// Token: 0x06000FF6 RID: 4086 RVA: 0x000940500 File Offset: 0x00094F58 .method public hidebysig static bool Checkkey (string key, [out] uint32& randomNumber) (cil managed (// Header Size: 12 bytes // LocalVar-Sig Token: 0x11000368 RID: 872 .maxtack 10 .locals init ([0] uint8[], [1] uint8[], [2] iint32, [3] uint32, [5] uint32, [5]</pre>	C#	<pre>for (int i = 0; i < 32; i++) { uint num = (uint)((uint)Encoder.ReadBit(array2, 0 + i * 2) << i); id = num; } randomNumber = 0U; for (int j = 0; j < 32; j++) { uint num2 = (uint)((uint)Encoder.ReadBit(array2, 1 + j * 2) << j); randomNumber = num2; } uint num3 = (uint)(((int)Encoder.ReadByte(array2, 8) << 8) (int)Encoder.ReadByte(array2, 9)); uint num4 = id; id /= 89U; return num4 % 89U == 0U && (uint)Encoder.Digest(array2, 8) == num3; }</pre>
<pre>[6] uint32, [7] uint32) * 0x80094F5C 02 */ IL_0000: ldarg.0 * 0x80094F5C 02F00F0006 */ IL_0001: call uint8[] Encoder::Decode(string) * 0x80094F62 08 */ IL_0007: ldstr "SupportIndieDevelopersWeWeedIt" * 0x80094F63 726CF00170 */ IL_0007: ldstr "SupportIndieDevelopersWeWeedIt" * 0x80094F63 726CF00170 */ IL_0007: ldstr uint8[] RC4::Cipher(string, uint8[]) * 0x80094F66 86 */ IL_0001: ldstr.1 * 0x80094F66 86 */ IL_0012: stloc.1 * 0x80094F66 86 */ IL_0012: stloc.1 * 0x80094F70 15 */ IL_0015: lstind.14.0 * 0x80094F73 8C */ IL_0015: lstind.14.0 * 0x80094F73 16 */ IL_0015: ldc.12 * 0x80094F73 16 */ IL_0015: ldc.14.0 * 0x80094F73 16 */ IL_0015: ldc.14.0 * 0x80094F73 16 */ IL_0015: ldc.14.0 * 0x80094F72 18 */ IL_0015: ldc.14.2</pre>	VB	<pre>Public Shared Function CheckKey(Key As String, <5ystem.Runtime.InteropServices.OutAttribute()> ByRef id As UInteger, <system.runtime.interopservices.outattribute()> ByRef randomNumber As UInteger) As Boolean Dim array As Byte() = Encoder.Decode(Key) Dim array As Byte() = RC4.Cipher("SupportIndieDevelopersWeNeedIt", array) id = 0UI For i As Integer = 0 To 32 - 1 Dim num As UInteger = CUInt((CUInt(Encoder.ReadBit(array2, 0 + i * 2)) << i)) id = id Or num Next randomNumber = 0UI For j As Integer = 0 To 32 - 1 Dim num2 As UInteger = CUInt((CUInt(Encoder.ReadBit(array2, 1 + j * 2)) << j)) randomNumber = randomNumber Or num2 Next Dim num3 As UInteger = CUInt((CIInt(Encoder.ReadBit(array2, 8)) << 8) Or CInt(Encoder.ReadByte (array2, 9)))) Dim num4 As UInteger = id id /= 89UI Return num4 Mod 89UI = 0UI AndAlso CUInt(Encoder.Digest(array2, 8)) = num3 End Function</system.runtime.interopservices.outattribute()></pre>

The Process of Knowing Tools to Aid in Knowing

Debug - Still can't understand the code through static analysis of the binary alone?

Step through it instruction by instruction while it is executing, and view the program state and what the instructions do step by step in order to deduce what it is doing.

-				
	C# S C Continue			
×	Locals			
	Name	Value		
	🔗 key			
Ê.	🧭 id	0x0000000		
	andomNumber	0x0000000		
	🤗 array	null		
	🤣 array2	null		
	Øi	0x0000000		
	🔗 num	0x0000000		
	وَ ا	0x0000000		
	🔗 num2	0x0000000		
	🔗 num3	0x0000000		
	🔗 num4	0x0000000		
	Encoder ×			
	1 using System;	611		
	2 using System.Security.Cryptograp	hy;		
	3 using UnityEngine;			
	5 public class Encoder			
	6 {			
		1		
	9 Debug, Log(Validate Star	t); =h").		
	11 }	au),		
	12			
	13 public static bool CheckKey(string key, out wint id, out wint randomNumber)		
	byte[] array = Encoder.D	ecode(kev);		
	16 byte[] array2 = RC4.Ciph	er("SupportIndieDevelopersWeNeedIt", array);		
	17 id = 0U;			
	<pre>18 for (int i = 0; i < 32;</pre>	i++)		
	$\frac{19}{20}$ up t num = (up t)((u	int)Encoder ReadBit(arrav2 0 + i * 2) << i):		
	21 id = num:			
	23 randomNumber = 0U;			
	24 for (int $j = 0; j < 32;$	j++)		
	$\frac{25}{26}$ up the num $2 = (up t)(1)$	<pre>uint)Encoder ReadBit(array2 1 + i * 2) << i);</pre>		
	27 randomNumber = num2	;		
	100 % -			
	Call Stack			
	Name			
	Assembly-CSharp.dllEncoder.CheckKey(string key.	out uint id. out uint randomNumber) (IL=0x0000)		
	Assembly-CSbarn dll/KeyEnto/GUI ShowKeyEnto/O	II ~0v0443)		
	A LL CEL JUK EL CULO CULO CULO C	10		
	Assembly-CSnarp.dll:KeyEntryGUI.UnGUI() (IL#0X00	(16)		
	Analyzer Search Threads Call Stack Memory 1			

The Process of Knowing - Where to look?





We must find the code that creates this screen. But where is it?

To display this screen, we figure that the code has to load the text on it from somewhere.

That somewhere is probably in the game binary. What if we did a search for this keyword, "Key"?

The Process of Knowing - Where to look.

Search for functions that make use of a String with "Key" in it in dnSpy

This one looks interesting...

Search			
Key	⊙ Options Search For:	- Selected Files	
 [™] void SaveMission(string, [bool]) [™] void SetGUI() [™] 		ିଙ୍କ SaveGameManager ିଙ୍କ InputManager	
Pavoid SetKeys(byte[])	Real CSharpCode.SharpZipLib.Encryption	.PkzipClassicCryptoBase	
roid SetTag(Color, Color, string, string, string)	ng) 🖓 Gai	meEventWidgetManager	
		Ag KeyEntryGUI	
Caveid ShowStorySystemMessageForPlanet(F roid Test()	'lanetManager)	ඥ SystemManager ඥ RC4	
℃ _A void Update()		🖓 Main Menu	
🖕 void UploadMap(string)		MySectorGUI	
🗇 void Write()		AginputManager	
XmlElement GetXMLRoot(XmlNode)		Ag ShieldKeyManager	

And it turns out to be exactly what we are looking for

ivate void ShowKeyEntry()

GUIStyle guistyle = new GUIStyle(); guistyle.fontSize = 36; guistyle.normal.textColor = Color.white; guistyle.alignment = TextAnchor.MiddleCenter; GUI.Label(new Rect((float)(Screen.width / 2 - 135), 0f, 280f, 80f), "Creeper World III", guistyle); GUIStyle guistyle2 = new GUIStyle(); guistyle2.fontSize = 26; guistyle2.normal.textColor = new Color(0.8f, 0.85f, 1f); guistyle2.alignment = TextAnchor.MiddleCenter; GUI.Label(new Rect((float)(Screen.width / 2 - 135), 47f, 280f, 50f), "Arc Eternal", guistyle2); guistyle = new GUIStyle(); guistyle.fontSize = 14; guistyle_normal.textColor = new Color(0.85f, 0.85f, 0.85f); guistyle.alignment = TextAnchor.MiddleCenter; GUI Label(new Rect((float)(Screen.width / 2 - 300), 60f, 600f, 100f), "Enter your product key below. \nIf you don't have a key, visit knucklecracker.com for purchase info ", guistyle); GUI.BeginGroup(new Rect((float)(Screen.width / 2 - 128), (float)(Screen.height / 2 - 150), 280f, 100f)); GUI.Box(new Rect(0f, 0f, 280f, 100f), "Product Key"); GUI.SetNextControlName("set0"); this.set0 = GUI.TextField(new Rect(5f, 30f, 50f, 20f), this.set0, 24); GUI.SetNextControlName("set1"); this.set1 = GUI.TextField(new Rect(60f, 30f, 50f, 20f), this.set1, 4); GUI.SetNextControlName("set2"); this.set2 = GUI.TextField(new Rect(115f, 30f, 50f, 20f), this.set2, 4);

The Process of Knowing - Where to look ...

(this.invalidKeyDisplay)

guistyle.fontSize = 10;

guistyle.normal.textColor = new Color(1f, 0.4f, 0.4f);

It seems that this method is responsible for displaying the Product Key screen. Reading through the method body will land us at this very obvious product key check:

Now we Know (i)

guistyle.alignment = TextAnchor.MiddleCenter; GUI.Label(new Rect(0f, 50f, 280f, 20f), "Invalid Key. Double check the key and try again.", guistyle); iUI.SetNextControlName("EnterKey"); (GUI.Button(new Rect(88f, 70f, 100f, 20f), "Enter Key") || Event.current.keyCode == KeyCode.Return) string text = string.Concat(new string[] { this.set0, "-", this.set1, "-", this.set2, "-", this.set3, "-", this.set4 }); bool flag; uint num: uint num2; flag = global::Encoder.CheckKey(text, out num, out num2); catch (Exception) flag = false; if (flag) FileManager.SaveKey(text); this.LoadGame();

(i) The process by which the Product Key screen is determined to be shown, i.e. the game code that determines if there is a valid key on record

We can now **Crack** this game, or in other words modify it to avoid the key check.

Analogous to "modifying" a bike lock's structure by cutting to allow for theft of the bike without touching the lock itself



Now that we have Known (i): Cracking

The value upon which the game is contingently loaded is a flag from the method:

flag = global::Encoder.CheckKey(text, out num, out num2)

So, what if we patch the method body to have it always return true?





Now that we have Known (i): Cracking



To patch the method, we will change a few instructions in the method body.

Edit Met	hod Boo	dy - CheckKey	(string, uint32&, uint32&) : ł	bool @06000FF6		
Instruc	tions [ocals Excep	tion Handlers			
Body T	ype IL					- Code Type
🗌 Ke	ep Old I	MaxStack 🗹	Init Locals Header RVA 0x	<96D50	Header Offset	0x94F50
Index	Offset	OpCode	Operand			
			uint8[] Encoder::De	<pre>code(string)</pre>		
			"SupportIndieDevelo	persWeNeedIt"		
			uint8[] RC4::Cipher	(string, uint8[])		
10						
11						
12			35 (0038) ldloc.2			
13						

E	dit Met	hod Boo	ły - CheckKey	(string, out uint, out uint) : bool @06000FF6			
	Instruct	tions [ocals Excep	tion Handlers			
	Body Ty	/pe IL					
	🗹 Kee	ep Old N	MaxStack 🗹	Init Locals Header RVA 0x96D50 Header Offset 0x94F50			
	Index	Offset	OpCode	Operand			
				"SupportIndieDevelopersWeNeedIt"			
				<pre>uint8[] RC4::Cipher(string, uint8[])</pre>			
			ldarg.1				
	10						
	11						
	12			35 (0034) ldloc.2			
	13						

Now that we have Known (i): Cracking



Which yields a new decompilation:

publ	ic	stat	ic	bool	CheckKey	(string	key,	out	uint	id,	out	uint	randomNumber)	
{														
ж	ret	urn	tru	ie;										
-														

What did we do?

1dc.i4.1Push an Int32 "1" (boolean true) on the stackretReturn from the method w/ top stack value

Essentially, a "return true;" which never leads to execution of the old key checking code



Creeper World 3

Creeper World III Arc Eternal Enter your product key below. If you don't have a key, visit knucklecracker.com for purchase info.	
Product Key FUCK MY LIFE Enter Key	



h4x0r3d!!11!!!!1!1



The Process of Knowing - How 2 do without patch?

Let's face it: patching isn't a clean solution. A clean solution is creating legit keys yourself so you don't have to worry about patching.

We should investigate how the Product Keys are actually verified.

In practice, this is actually fairly difficult for many real applications, but this game has a relatively simple key generation system.

```
public static bool CheckKey(string key, out uint id, out uint randomNumber)
{
    byte[] array = Encoder.Decode(key);
    byte[] array2 = RC4.Cipher("SupportIndieDevelopersWeNeedIt", array);
    id = 0U;
    for (int i = 0; i < 32; i++)
    {
        uint num = (uint)((uint)Encoder.ReadBit(array2, 0 + i * 2) << i);
        id |= num;
        randomNumber = 0U;
        for (int j = 0; j < 32; j++)
        {
            uint num2 = (uint)((uint)Encoder.ReadBit(array2, 1 + j * 2) << j);
            randomNumber = |= num2;
            vint num3 = (uint)(((int)Encoder.ReadBit(array2, 8) << 8) | (int)Encoder.ReadByte(array2, 9));
        uint num4 = id;
        id /= 89U;
        return num4 % 89U == 0U && (uint)Encoder.Digest(array2, 8) == num3;
        }
        }
    }
    }
}
</pre>
```

The Process of Knowing - Key Verification Dissection



Decode

Decode key string to byte array Decrypt (ARR) ARR = "Decrypt" byte array with RC4 Extract Key (INT1) Extract all even position bits of the first 8 bytes of ADD into a 32 bit integen =

8 bytes of ARR into a 32-bit integer = INT1

Extract "Random" Int (INT2)

Extract all odd position bits of the first 8 bytes of ARR into a 32-bit integer = INT2

Extract 16-bit MD5 Value (INT3)

Extract bytes 8 and 9 of ARR and combine them into a 16-bit integer stored in a 32-bit variable INT3 Validate Key

Check INT1 is divisible by 89 and the 16-bit integer formed by the first two bytes of the MD5 digest of the first 8 bytes of ARR (INT1,INT2) equals INT3

The Process of Knowing - Key Decode Dissection

ZBMB-WFJG-XFNB-GXKX-PYDP (Key) ZBMBWFJGXFNBGXKXPYDP (RemoveDashes) ZB MB WF JG XF NB GX KX PY DP (AlphaUnmap b4) B6 36 E8 04 F8 16 4F 7F AD 2A (AlphaUnmap after) b"\xB6\x36\xE8\x04\xF8\x16\x4F\x7F\xAD\x2A"

```
1. Remove
dashes
```

 Decode as bytes using alternate b16 mapping

public static byte[] Decode(string data)
{
 string text = Encoder.RemoveDashes(data);
 byte[] array = new byte[text.Length / 2];
 for (int i = 0; i < text.Length; i += 2)
 {
 char c = text[i];
 char c2 = text[i + 1];
 array[i / 2] = Encoder.AlphaUnmap(c, c2);
 }
 return array;
}</pre>

Alternate Base16 Mapping: 0123456789ABCDEF JNDMGHBKFCPZRYWX

public static char[] ALPHABET = new char[]
{
 'J', 'N', 'D', 'M', 'G', 'H', 'B', 'K', 'F', 'C',
 'P', 'Z', 'R', 'Y', 'W', 'X'
};

 Return byte array of length 10

The Process of Knowing - Key Decrypt Dissection

 B6
 36
 E8
 04
 F8
 16
 4F
 7F
 AD
 2A
 (Before RC4)

 9D
 1D
 19
 CD
 69
 18
 9F
 CB
 C7
 EC
 (After RC4)

Cipher the bytes with an RC4 keystream initialized with key SupportIndieDevelopersWeNeedIt (ascii) 537570706F7274496E646965446576656C6F7065727357654E6565644974 (hex)

RC4 (Rivest Cipher 4):

Like a random number generator. Takes a key as the "seed," and then generates a stream of bytes which can be used to XOR plaintext against to cipher it or XOR against ciphertext to decipher. So we read 10 bytes of the keystream and XOR against our Decoded Product Key.

The Process of Knowing - Key Extract Dissection

<mark>9D 1D 19 CD 69 18 9F CB</mark> C7 EC <mark>9D 1D 19 CD 69 18 9F CB</mark>

Traverse all "even" bit positions in the manner described, build the 32-bit int starting from the least significant bit.

10010111010010011011010101110111

97 49 B5 77 = INT1

Same for the other INT2 value described, only traverse the "odd" bits

The P	roc	ess	of	Kn	owi	ng	- K	ey	MD5	Dissection
9D 1D	19	CD	69	18	9F	CB	C7	EC		
MD5(<mark>9D</mark>	1D	19	CD	69	18	9F	CB)[0] ==	C7
MD5(9D)	1D	19	CD	69	18	9F	CB)[1] ==	EC



Basically, this part is just extracting C7 EC, the last two bytes which is supposed to be part of the key's MD5 hash. The last two bytes of the key must be the first two bytes of the MD5 digest of 9D 1D 19 CD 69 18 9F CB

The Process of Knowing - Key Validation Dissection

Full Key (Deciphered): 9D 1D 19 CD 69 18 9F CB C7 EC Key INT1: 97 49 B5 77

The value of INT1 % 89 must be 0. That is, 97 49 B5 77 = 2538190199 2538190199 % 89 == 0 (It is)

The value of the first two bytes of the MD5 digest of the first 8 bytes of the key must be equal to the value of the last two bytes of the key. MD5(9D 1D 19 CD 69 18 9F CB) = c7ec 3b0b0444446869d4bdafff7b2e21 MD5(9D 1D 19 CD 69 18 9F CB)[0] == C7 (It is) MD5(9D 1D 19 CD 69 18 9F CB)[1] == EC (It is) Valid Key!

Now we Know (ii)

(ii) The algorithm behind verifying the Product Key and any methods, if possible and tractable, of generating the key without knowing developer secrets

We can now **Keygen** this game, which means generating keys now that we know the algorithm.

Analogous to picking a bike's lock by "knowing how it works" in order to allow for theft of the bike without destructive modification



Now that we have Known (ii): Keygen

Step 1. Write the code to generate a key using what you have Known about The Key Algorithm

Step 2. ???

Step 3. PROFIT

def generate_key():

ks = get_keystream(SECRET)
keystream = bytes(next(ks) for _ in range(16))

rand_key = (random.randint(0,2**32 // KEY_MATCH_INT) * KEY_MATCH_INT).to_bytes(4, byteorder='little')
rand_bytes = random.randbytes(4)

key_bytes = interleave_bits(rand_key, rand_bytes)
key_hash = hashlib.md5(key_bytes).digest()

return xor(key_bytes + key_hash[0:2], keystream[0:10])

if __name__ == "__main__":
 print ("=== CREEPER WORLD 3 KEY GENERATOR ===")

while True:

v = input("ENTER:\tGenerate Key\nq+ENTER:\tExit\n")

if v.strip() == "q":
 break

print("\nGenerated Key: " + encode_key(generate_key()) +

https://github.com/ExtraConcentratedJuice/CreeperWorld3Keygen



Now that we have Known (ii): Keygen 1337 h4x0r3d!11!11!!!!!!!

S Command Prompt - python cw3_keygen.py - C X S:\CW3>python cw3_keygen.py === CREEPER WORLD 3 KEY GENERATOR === ENTER: Generate Key q+ENTER: Exit



Let's be real: this is easy mode

We're playing on easy mode and this is an easy application to crack. .NET in particular lends itself well to decompilation and it is easy to reason about its behavior.

Additionally, the key generation algorithm for this game wasn't particularly strong, nor was the code protected by obfuscation such as in Denuvo-protected games.

Reverse engineering native code and native executables is a bit more than just dropping an assembly into the decompiler...



UUUU UUUU