iOS Tutorial – Dumping the Application Memory Part 2

In my previous blog, iOS Tutorial – Dumping the Application Heap from Memory, I covered how to dump sensitive information from the heap of an iOS application using GDB. This time we will be covering how to use Cycript to accomplish the same goal but using the class-dump-z output to specifically pull out properties or instance variables. This round will be in a more automated fashion by automatically parsing a class dump of the binary and generating the necessary Cycript scripts to pull the specific properties from memory. I will also be releasing another tool to do all of this for you in the near future. Keep an eye on our NetSPI GitHub repo for the latest tools and scripts for when we release it.

If we do not have access to the source code then we must first decrypt the binary. We do this first to dump the class information about the binary. There are several guides out there for decryption but Clutch is my go-to tool for ease of use as it also regenerates an IPA file with the decrypted binary in it so you can install it again on a different device if you have to. After we extract/install the new decrypted binary, we can now run class-dump-z to get the header information with all the classes, properties, class methods, instance methods, etc.

```
MAPen-iPad-000314:~ root# ./class-dump-z -z TestApp

[TRUNCATED]

@interface CryptoManager : XXUnknownSuperclass {
@private
        NSData* key;
}
@property(retain, nonatomic) NSData* key;
+(id)CryptoManager;
-(id)init;
-(id)cipher:(id)cipher key:(id)key context:(unsigned)context;
-(id)cipher:(id)cipher key:(id)key context:(unsigned)context withIV:(BOOL)iv;
-(id)cipher:(id)cipher key:(id)key context:(unsigned)context withIV:(BOOL)iv
usingIV:(id)iv5;
-(id)cipher:(id)cipher key:(id)key context:(unsigned)context withIV:(BOOL)iv
usingIV:(id)iv5 withPad-ding:(BOOL)padding;
-(void)clearKey;
-(void)dealloc;
-(id)decryptData:(id)data;
-(id)decryptData:(id)data usingIV:(id)iv;
-(id)decryptData:(id)data usingIV:(id)iv withPadding:(BOOL)padding;
-(id)decryptData:(id)data withIV:(BOOL)iv;
-(id)decryptData:(id)data withIV:(BOOL)iv withHeader:(BOOL)header;
-(id)decryptData:(id)data withKey:(id)key;
-(id)decryptString:(id)string;
```

```
-(id)decryptString:(id)string withIV:(BOOL)iv;
-(id)decryptString:(id)string withIV:(BOOL)iv withHeader:(BOOL)header;
-(id)decryptString:(id)string withIV:(BOOL)iv withHeader:(BOOL)header
withKey:(id)key;
-(id)decryptString:(id)string withKey:(id)key;
-(id)encryptData:(id)data;
-(int)encryptData:(id)data AndAppendToFileAtPath:(id)path
initiatedByUnlockOperation:(BOOL)operation error:(id*)error;
-(id)encryptData:(id)data usingIV:(id)iv;
-(id)encryptData:(id)data withKey:(id)key;
-(id)encryptString:(id)string;
-(id)encryptString:(id)string withKey:(id)key;
-(id)hashString:(id)string;
-(id)hashString:(id)string salt:(id)salt;
-(BOOL)isHashOfString:(id)string equalToHash:(id)hash;
-(BOOL)isHeaderValid:(id)valid;
-(id)newHeader;
-(unsigned long)readEncryptedData:(void**)data atPath:(id)path
offset:(long)offset length:(unsigned long)length
initiatedByUnlockOperation:(BOOL)operation error:(id*)error;
@end

[TRUNCATED]
```

So you can see above that TestApp has a class called "CryptoManager" and has a property called "key". This looks interesting as there could be an encryption key sitting there in memory. We will now use Cycript to grab that specific property from memory. Note during runtime, the "CryptoManager" class is instantiated before login but only after a valid user has successfully logged in once before on the device. Also, the class is never cleared out even when it is no longer needed, such as a user logged out, which is where the vulnerability lies. In this instance, we have already logged in successfully during a previous session and therefore the class is already in memory before the user logs in.

First we will hook into the running TestApp process from an SSH session so we can leave the application running on the iOS device.

```
MAPen-iPad-000314:~ root# cycript -p TestApp
cy#
```

Now that we are hooked in, let's go ahead and talk about the "choose" method in cycript. The "choose" method scans the heap for the matching class name and returns an array of objects that match that class' structure. So, if we type "choose(MyClass)". It is going to contain an indexed array of all instantiated classes of MyClass that are currently in memory (or that match that structure). The below output is just calling out the first indexed object which is index "0" and storing it into a variable called "a". If you like GDB more, we can also take the memory location returned and go back to GDB for dumping out everything from that sub-region in memory or set breakpoints and watch the registers. See my previous blog on how to scan the heap here (https://blog.netspi.com/ios-tutorial-dumping-the-application-heap-from-memory/). Note however, that there can be more than one class instantiated in this array and you will to go through each index to get the properties of that instantiated class.

```
cy# a=choose(CryptoManager)
[#"< CryptoManager: 0x17dcc340>",#"< CryptoManager: 0x17f42ba0>"]
```

Now let’s dump the “key” property from memory so we can grab the key and decrypt any data in the app later on.

```
cy# a[0].key.hexString
@"6D2268CFFDDC16E890B365910543833190C9C02C4DCA2342A9AEED68428EF9B6"
```

Bingo! We now have the hexadecimal of the key we need to decrypt anything this application wants to keep encrypted.

Now let’s talk about how to automate this and go over what we know and what we have to figure out programmatically as we go. We know that the class-dump-z output contains the output of all the classes and their properties. What we don’t know is whether or not those classes are currently instantiated or not. We also don’t know how many times the classes are instantiated in memory. What we can do is parse the class-dump-z output and create a map of classes and their properties. Now that we have a map we can now create Cycript scripts to pull the information out for us. Note however, that this technique is for classes that are already instantiated and we won’t be covering how to make a new instance of an object in Cycript as there are many tutorials and books on how to do this.

So we have to read Cycript’s output from the choose method to figure out how many times the object is instantiated in memory. To do that we can use JavaScript to get the array length:

```
cy# choose(CryptoManager).length
2
cy#
```

Cool, now we know how many times to loop through the array to pull out all instantiated “CryptoManager” objects. Now let’s move on to cycript scripting.

Cycript can take a script as a parameter and a basic script just has to contain the commands we want to run like so:

```
MAPen-iPad-000314:~ root# cat dump.cy
a=choose(CryptoManager)[0]
a.key.hexString

MAPen-iPad-000314:~ root# cycript -p TestApp dump.cy
@"6D2268CFFDDC16E890B365910543833190C9C02C4DCA2342A9AEED68428EF9B6"
```

One issue that I can’t seem to figure out is Cycript only returns the last line of output to the terminal when you run a script and doesn’t return all output. So to pull out multiple classes and their properties from the terminal, you have to create a new script for each class and property combination. If anyone knows how to get around this limitation, please feel free to reach out to me on how to accomplish this. Or you can write everything in Cycript JavaScript if that is your preferred language.

Thanks for reading and hack responsibly