datatheorem VIEW IN PORTAL

Switch to Secure Object Archiving APIs

CultureNEXT - PreProd 1.0.0 ID $153602 Severity: Medium No Priority

The following code locations leverage unsafe methods of the `NSKeyedArchiver` or `NSKeyedUnarchiver` APIs, which are vulnerable to object substitution attacks:

OPEN since 4/10/2021 `-[CTAppController applicationDidFinishLaunching:]` calls `-[NSKeyedUnarchiver unarchiveObjectWithData:]`
Stack Trace

OPEN since 4/13/2021 `-[PWRequestsCacheManager requestsQueue]` calls `-[NSKeyedUnarchiver unarchiveObjectWithFile:]`
Stack Trace

OPEN since 4/13/2021 `-[PWRequestsCacheManager save]` calls `-[NSKeyedArchiver archiveRootObject:toFile:]`
Stack Trace

View 14 more components

DESCRIPTION

The `NSKeyedArchiver` or `NSKeyedUnarchiver` methods used by the App are insecure because they are incompatible with the `NSSecureCoding` protocol. An attacker-controlled payload that is deserialized via these APIs may result in attacker-controlled code being executed.

`NSCoding` is an Objective-C protocol that interoperates with `NSKeyedArchiver` and `NSKeyedUnarchiver`. Together, these APIs allow serialization and deserialization of code objects. However, the `NSKeyedUnarchiver` methods used by the app, and the `NSCoding` protocol itself, do not verify the type of object upon deserialization. Thus, an attacker may craft a malicious payload that results in unexpected code being executed.

To mitigate this vulnerability, Apple introduced the `NSSecureCoding` protocol along with the following secure methods of `NSKeyedArchiver` and `NSKeyedUnarchiver`, which are robust against this type of attack:

```
// Secure NSKeyedUnarchiver methods
- (instancetype)initForReadingFromData:(NSData *)data error:(NSError **)error;
+ (id)unarchivedObjectOfClass:(Class)cls fromData:(NSData *)data error:(NSError **)error;
+ (id)unarchivedObjectOfClasses:(NSSet<Class> *)classes fromData:(NSData *)data error:(NSError **)error;

// Secure NSKeyedArchiver methods
- (instancetype)initRequiringSecureCoding:(BOOL)requiresSecureCoding;
+ (NSData *)archivedDataWithRootObject:(id)object requiringSecureCoding:(BOOL)requiresSecureCoding error:(NSError **)error;
```

These APIs protect against object substitution attacks by requiring the programmer to declare the expected type of object before deserialization completes. Thus, if an invalid object is deserialized, the error can be handled safely.

Apple provides more information in the WWDC20 session, 'Securing Your App'.

RECOMMENDATION

Locate all the classes in the App that conform to `NSCoding` and migrate them to `NSSecureCoding`. Then, replace the insecure usages of `NSKeyedArchiver` and `NSKeyedUnarchiver` with the secure APIs that perform error handling and validate the expected type of the deserialized objects.

Additionally, ensure all input data is validated before it is used, especially when dealing with data that becomes executable.

SECURE CODE

```
// Declare that your class conforms to NSSecureCoding
@interface MySecureObject : NSObject <NSSecureCoding>
@property (nonatomic, retain) NSDictionary *myData;
@end

@implementation MySecureObject
+ (BOOL)supportsSecureCoding {
    // Must override this class delegate method to reture YES
    return YES;
}
+ (MySecureObject*)deserializeFromData:(NSData*)data {
    // Inform the system of the object type we expect to be deserialized from the data
    // This method will return an error if the serialized data was invalid
    NSError* out_error = nil;
    [NSKeyedUnarchiver unarchivedObjectOfClass:[MySecureObject class] fromData:data error:&out_error];
    if (out_error != nil) {
        // Handle the error
        NSLog(@"Deserialization failed: %@", out_error);
    }
}
- (id)initWithCoder:(NSCoder *)decoder {
    if ((self = [super init])) {
        // When decoding sub-objects, use @selector(decodeObjectOfClass:forKey:)
        // This method will throw an exception if the deserialized object's class doesn't match the expected class
        self.myData = [decoder decodeObjectOfClass:[NSDictionary class] forKey:@"myData"];
    }
    return self;
}
- (void)encodeWithCoder:(NSCoder *)encoder {
    [encoder encodeObject:self.myData forKey:@"myData"];
}
@end
```

QUESTIONS & COMMENTS