CS80S - Theory and practice of secure systems

Fall 2024

Tue / Thu : 2 pm to 3:30 pm

Shravan Narayan

UT Austin

Announcements

Sign up to present a paper

Minor changes to papers. Please check what you've signed up for!

Think about your projects! Did you figure out compute resources?

Writeups submission links will be posted later today

Writeups and attendance starting Tuesday

Today

1. How to present a paper

2. Background on memory safety

3. Background on side-channels

4. Security News

Paper presentation

A good resource

(An Opinionated Talk) On Preparing Good Talks – Ranjit Jhala (2018)

Paper presentation: preparation

1. Watch the talk from the previous slide
2. Read the paper
3. Repeat step 1 until comfortable
4. Skim papers that are cited by the paper
5. Skim papers that have cited your paper (Use Google Scholar)
6. Watch any videos from authors/others presenting this work
7. Concisely state the high-level problem and the solution.
8. Shortlist important points – Do not try to cover everything :)

Come talk to me if you need help understanding the paper.

Paper presentation: Slides

1. You can make slides or give a whiteboard talk
2. Plan for a 30 min talk: typical average is 1 to 2 slides per minute
3. Make sure the talk clarifies
   - What the problem is
   - Why the problem matters
   - Why existing solutions won't help
   - What is the new approach proposed by authors
   - Relevant implementation details that can fit within the time of the talk
   - Evaluation of the new idea
4. You can borrow slides/figures from the paper if you cite sources
   - Original slides for paper are usually too dense

Background on memory safety

Why does memory safety matter?

From Pearl to Pegasus

Bahraini Government Hacks Activists with NSO Group Zero-Click iPhone Exploits

By Bill Marczak, Ali Abdulemam[1], Noura Al-Jizawi, Siena Anstis, Kristin Berdan, John Scott-Railton, and Ron Deibert

[1] Red Line for Gulf

August 24, 2021

Summary & Key Findings

- We identified nine Bahraini activists whose iPhones were successfully hacked with NSO Group's Pegasus spyware between June 2020 and February 2021. Some of the activists were hacked using two zero-click iMessage exploits: the 2020 ***KISMET*** exploit and a 2021 exploit that we call ***FORCEDENTRY***.

About the Citizen Lab

The Citizen Lab is an interdisciplinary laboratory based at the Munk School of Global Affairs & Public Policy, University of Toronto, focusing on research, development, and high-level strategic policy and legal engagement at the intersection of information and communication technologies, human rights, and global security.

We use a "mixed methods" approach to research combining practices from political science, law, computer science, and area studies. Our research includes: investigating digital es-

https://citizenlab.ca/2021/08/bahrain-hacks-activists-with-nso-group-zero-click-iphone-exploits/

Why does memory safety matter?

When the ***FORCEDENTRY*** exploit was being fired at a device, the device logs showed crashes associated with *IMTranscoderAgent*. The crashes appeared to be segfaults generated by invoking the *copyGifFromPath:toDestinationPath:error* function on files received via iMessage.

The crashes appeared to be of two types. Type one crashes indicate that the chain of events set off by invoking *copyGifFromPath:toDestinationPath:error* ultimately crashed while apparently invoking ImageIO's functionality for rendering Adobe Photoshop PSD data.

```
Thread 1 name: Dispatch queue: com.apple.root.default-qos
Thread 1 Crashed:
 0: ImageIO                 0x181b326f0 __ZN13PSDReadPlugin12GetRangeInfoEP9LayerInfoP19IIOImageReadSessionRK13PSDPluginDatat + 244
 1: ImageIO                 0x181b326ec __ZN13PSDReadPlugin12GetRangeInfoEP9LayerInfoP19IIOImageReadSessionRK13PSDPluginDatat + 240
 2: ImageIO                 0x181b32abc __ZN13PSDReadPlugin11DecodeLayerEP9LayerInfoRK6CGRectP19IIOImageReadSessionRK14ReadPluginDataRK13PSDPluginData + 448
 3: ImageIO                 0x181b32d40 __ZN13PSDReadPlugin11MergeLayersEP19IIOImageReadSessionR20IIODecodeFrameParamsRK14ReadPluginDataRK13PSDPluginData + 276
 4: ImageIO                 0x181b33248 __ZN13PSDReadPlugin11DecodeBlockEP19IIOImageReadSessionR20IIODecodeFrameParamsRK14ReadPluginDataRK13PSDPluginDatajj + 232
 5: ImageIO                 0x181b34c4c ____ZN13PSDReadPlugin12DecodeBlocksEP12IIOImageReadRK14ReadPluginDataRK13PSDPluginDataRNSt3__16vectorI20IIODecodeFrameParamsNS8_9allocatorISA_EEEE_block_invoke + 116
 6: libdispatch.dylib       0x18004b860 __dispatch_client_callout2 + 20
 7: libdispatch.dylib       0x18005f988 __dispatch_apply_serial + 120
 8: libdispatch.dylib       0x18004b81c __dispatch_client_callout + 20
 9: libdispatch.dylib       0x180050c7c __dispatch_sync_function_invoke + 56
10: libdispatch.dylib       0x18005f844 _dispatch_apply_f + 908
11: ImageIO                 0x181b34bcc __ZN13PSDReadPlugin12DecodeBlocksEP12IIOImageReadRK14ReadPluginDataRK13PSDPluginDataRNSt3__16vectorI20IIODecodeFrameParamsNS8_9allocatorISA_EEEE + 140
12: ImageIO                 0x181b34364 __ZN13PSDReadPlugin17copyImageBlockSetEP7InfoRecP15CGImageProvider6CGRect6CGSizePK14__CFDictionary + 1028
13: ImageIO                 0x181a28388 __ZN10IIO_Reader21CopyImageBlockSetProcEPvP15CGImageProvider6CGRect6CGSizePK14__CFDictionary + 152
14: ImageIO                 0x181a36510 __ZN20IIOImageProviderInfo28copyImageBlockSetWithOptionsEP15CGImageProvider6CGRect6CGSizePK14__CFDictionary + 756
15: ImageIO                 0x181a33e74 __ZN20IIOImageProviderInfo28CopyImageBlockSetWithOptionsEPvP15CGImageProvider6CGRect6CGSizePK14__CFDictionary + 584
16: CoreGraphics            0x182004c68 _imageProvider_retain_data + 92
17: CoreGraphics            0x181decaa8 _CGDataProviderRetainData + 80
18: CoreGraphics            0x181ff1e04 _CGAccessSessionCreate + 108
19: CoreGraphics            0x181e21bc8 _CGDataProviderCopyData + 168
20: CoreGraphics            0x181d95230 _CGImageGetDataProviderInternal + 268
21: CoreGraphics            0x181eccc10 _img_image + 600
22: CoreGraphics            0x181ecc6e8 _CGSImageDataLock + 1004
23: CoreGraphics            0x181ceb2bc _ripc_AcquireRIPImageData + 716
24: CoreGraphics            0x181eed25c _ripc_DrawImage + 1152
25: CoreGraphics            0x181ed3b40 _CGContextDrawImageWithOptions + 1216
26: ImageIO                 0x181ae2730 __ZN14GIFWritePlugin16writeSingleFrameEv + 1016
27: ImageIO                 0x181ae3890 __ZN14GIFWritePlugin8writeAllEv + 580
28: ImageIO                 0x181a6d76c __ZN14IIO_Writer_GIF5writeEPvS0_ + 36
29: ImageIO                 0x181aae554 __ZN19IIOImageDestination19finalizeDestinationEv + 548
30: ImageIO                 0x181aaffd4 _CGImageDestinationFinalize + 132
31: IMSharedUtilities       0x18f77c9b4 writeNewFileAtPath:withProperties:fromImageSource:error: + 236
32: IMSharedUtilities       0x18f77cd88 copyGifFromPath:toDestinationPath:error: + 348
33: IMTranscoderAgent       0xf08e58c8
```

https://citizenlab.ca/2021/08/bahrain-hacks-activists-with-nso-group-zero-click-iphone-exploits/

Goal – Before next class, you should know

1. What is a memory safety attack?

2. How does it compromise the system?

But first: basic OS concepts

Model of process – stack, heap, code, functions calls

Virtual memory vs physical memory

Memory pages, page permissions, and segmentation faults

But first: basic OS concepts

Model of process – stack, heap, code, functions calls

Virtual memory vs physical memory

Memory pages, page permissions, and segmentation faults

C is a language to target these abstractions, but…

C/C++ is not “memory safe”

Designed > 30 years ago when performance was key
⇒ Not secure by default

Google Chrome: ~70% of bugs (2015–2020)

A problem for old and new software alike

Memory safety root cause: Missing checks in code

- Buffer overflows
- Pointer dereference out-of-bounds
- Use-after-free
- Type confusion

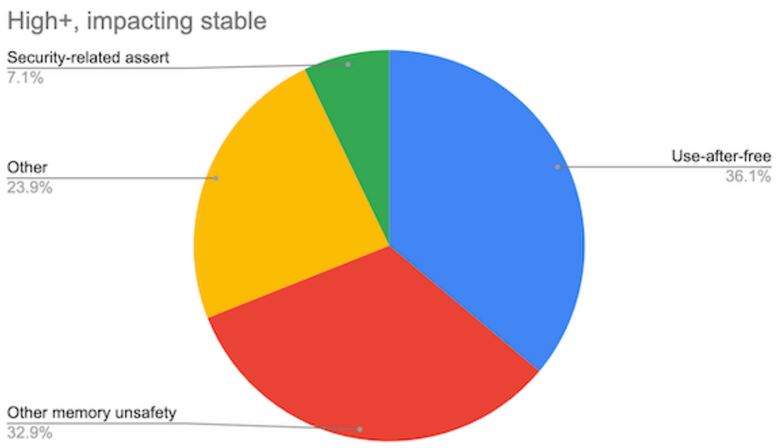

Image from the Chromium project blog
https://www.chromium.org/Home/chromium-security/memory-safety/

C/C++ is not “memory safe”

While a programmer must fix all possible bugs and defenses must cover all possible attack vectors to secure a system, an adversary only needs to find one exploitable bug to gain control of the system.

Memory safety

Classically divided into a few sub-areas

- Spatial safety
- Temporal safety
- Type safety

Memory safety: spatial safety violation

```
int main(int argc, char** argv) {
    char buffer[100];

    for(int i = 0; i <= 100; i++) {
        buffer[i] = 'a';
        // buffer[100] is OOB, buffer overflow, memory unsafe, spatially unsafe
    }

    // Extra question: this is C UB, how does this related to memory safety
}
```

Memory safety: temporal safety violation

```
int main(int argc, char** argv) {
    char* buffer = malloc(16);

    char* copy = buffer;
    free(copy);

    // violation as buffer no longer points to a valid object
    buffer[12] = 23;
}
```

Memory safety: type safety violation

```
class B {
public:
    int b_field;
};

class D {
public:
    virtual void d_func() {}
    int d_field;
};
```

```
int main(int argc, char** argv) {
    D* d_ptr = new D;

    // Bad cast
    B* b_ptr = (B*) d_ptr;

    // Type confusion!
    b_ptr->b_field = 0x43;
    b_ptr->d_func();
}
```

Mem safety attacks break program properties

1. Memory integrity
   - Data integrity
   - Data pointer integrity
   - Code pointer integrity

2. Memory confidentiality
   - Leak data
   - Leak location of code

Mem safety attacks break program properties

1. Memory integrity
   - Data integrity
   - Data pointer integrity
   - Code pointer integrity

2. Memory confidentiality
   - Leak data
   - Leak location of code

Background on side-channels

Transient-Execution Attacks: A Computer Architect Perspective

Luís Fiolhais, Leonel Sousa (2023)

Normal attacks

Learn data due to bugs in hw/sw (buffer overflows)

Side-channel attacks

Learn data by measuring indirect effects of hw/sw impls. (caches)

Architectural concepts

Translation lookaside buffer (TLB) and page table entries (PTE)
Superscalar, Instruction-level parallelism (ILP)
Out of order execution (OOO), Reorder buffer (ROB)
Branch predictors & Speculative execution
Instruction timing and microcode
Store to load forwarding
Frequency scaling

Caches and the cache hierarchy

- Ways and sets
- Inclusive vs Exclusive
- Writeback vs. Write-through

Prime + Probe cache attack

```
// I am sharing a single core computer with a webserver
void on_http_request(int user_id) { // Server code
    auto userinfo = user_data[user_id];
    // Some IO
}

void prime_and_probe() { // My code
    // 1. Create an array the size of the cache
    // 2. Access the entire array to fill up the cache and then sleep
    // 3. On wakeup, measure time to access each array element
    // 4. Slow-elements correspond to cache collisions
}
```

Question 1:
How can I use this to learn information

Question 2:
What are the ways this is not realistic?

Security News

Let's look at some news

Windows Patch Tuesdays

https://www.bleepingcomputer.com/news/microsoft/microsoft-august-2024-patch-tuesday-fixes-9-zero-days-6-exploited/

Google Chrome bugs

https://thehackernews.com/2024/08/google-fixes-high-severity-chrome-flaw.html