Welcome to CS 110L 👋

Thea Rossman
Winter 2022

Today

- Quick intros
- Why are we here? (issues motivating the class)
- About & plans for the course

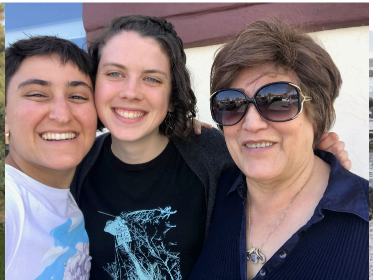
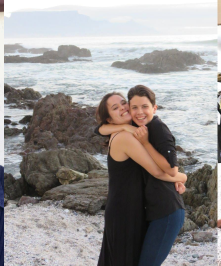
- *Zoom norms:*
  - *Please enable video (if you have one)*
  - *Try to mute yourself when not speaking*
  - *Please ask and answer questions! Feel free to just unmute, but chat is fine if you can't do that.*

Who are we?

This course and all material were put together by Ryan Eberhardt and Armin Namavari, with support from Will Crichton and Julio Ballista

Thea (pronounced thee-uh)

- MS/coterm focused on computer networking and systems
- Interest in the Internet / systems grew from CS110 & CS144; + being adjacent to community broadband projects; interest in security grew from being adjacent to social movement organizations navigating surveillance, doxxing, infiltration, etc.
- Knows about systems & teaching systems. Rust newbie.

Who are you?

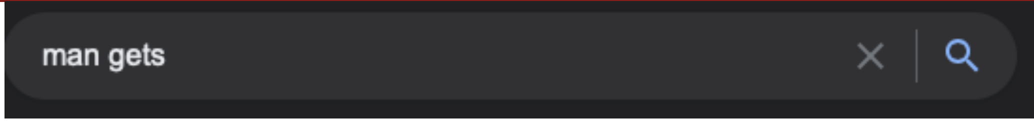
- Put in the chat...
  - Your name
  - What you're studying OR one fun fact about yourself
  - (Optionally) one thing that intrigues you about the class

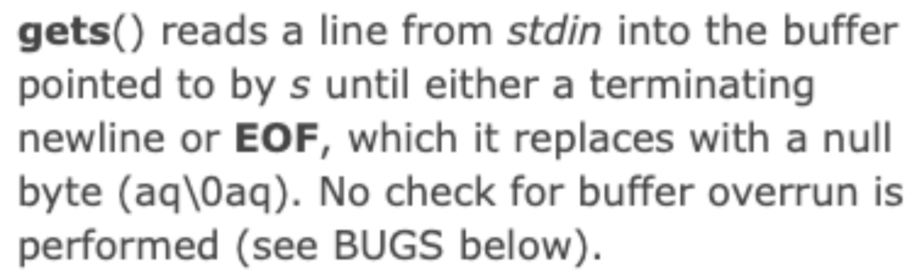
Why are we here?

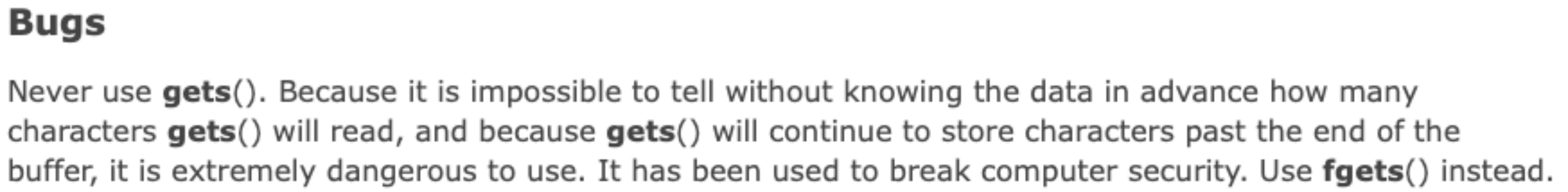
“Convert a String to Uppercase in C,” taken VERBATIM from Tutorials Point

```c
#include <stdio.h>
#include <string.h>
int main() {
   char s[100];
   int i;
   printf("\nEnter a string :  ");
   gets(s);
   for (i = 0; s[i]!='\0'; i++) {
      if(s[i] >= 'a' && s[i] <= 'z') {
         s[i] = s[i] -32;
      }
   }
   printf("\nString in Upper Case = %s", s);
   return 0;
}
```

From the documentation: https://linux.die.net/man/3/gets



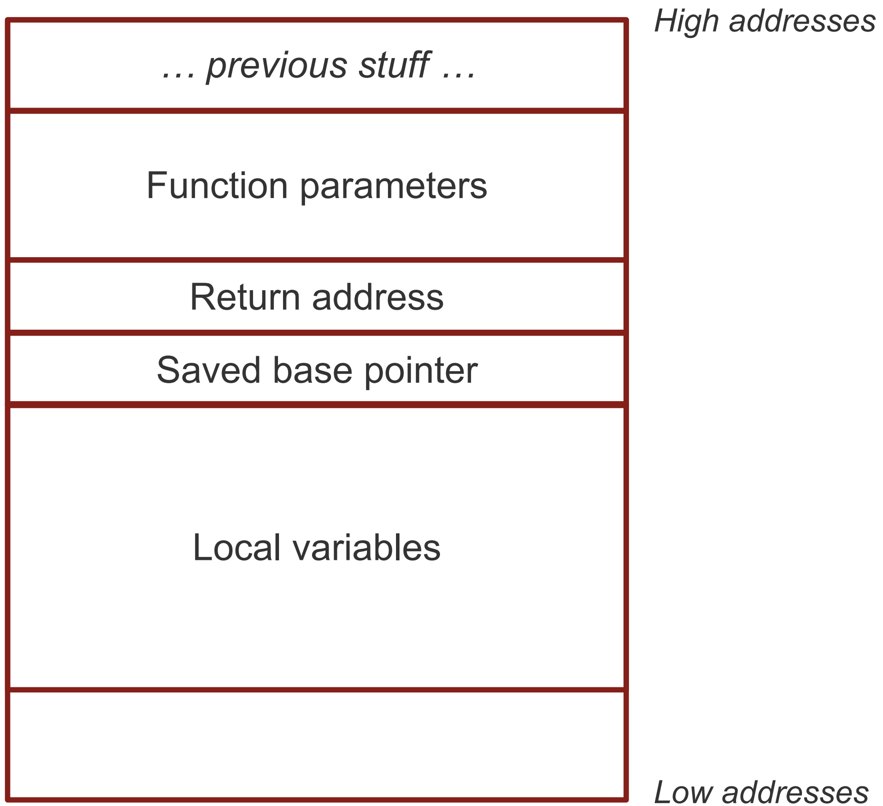

gets() reads a line from *stdin* into the buffer pointed to by *s* until either a terminating newline or **EOF**, which it replaces with a null byte (aq\0aq). No check for buffer overrun is performed (see BUGS below).


Bugs

Never use **gets**(). Because it is impossible to tell without knowing the data in advance how many characters **gets**() will read, and because **gets**() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use **fgets**() instead.

Anatomy of a Stack Frame

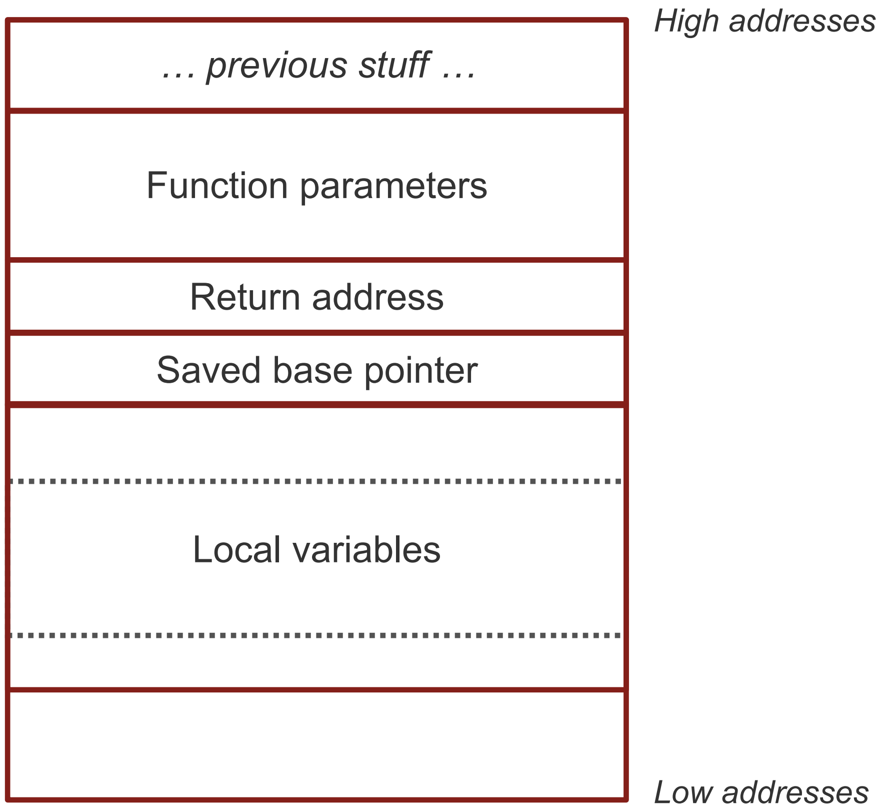
```
; push call arguments, in reverse
push    3
push    2
push    1
call    callee    ; call subroutine ‘callee'

    callee:
    push    ebp        ; save old call frame
    mov     ebp, esp   ; initialize new call frame
    ...do stuff...
    mov     esp, ebp
    pop     ebp        ; restore old call frame
    ret                ; return


add     esp, 12  ; remove call arguments from frame
```



From https://en.wikipedia.org/wiki/X86_calling_conventions#cdecl

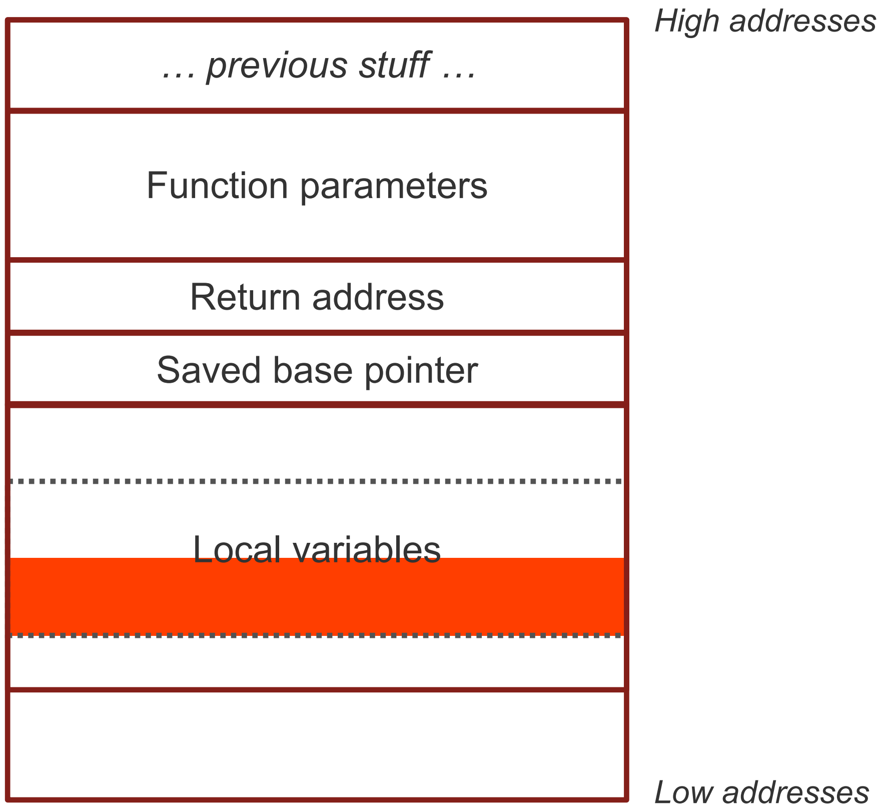
Anatomy of a Stack Frame

```
; push call arguments, in reverse
push    3
push    2
push    1
call    callee    ; call subroutine 'callee'

    callee:
    push    ebp        ; save old call frame
    mov     ebp, esp   ; initialize new call frame
    ...do stuff...
```



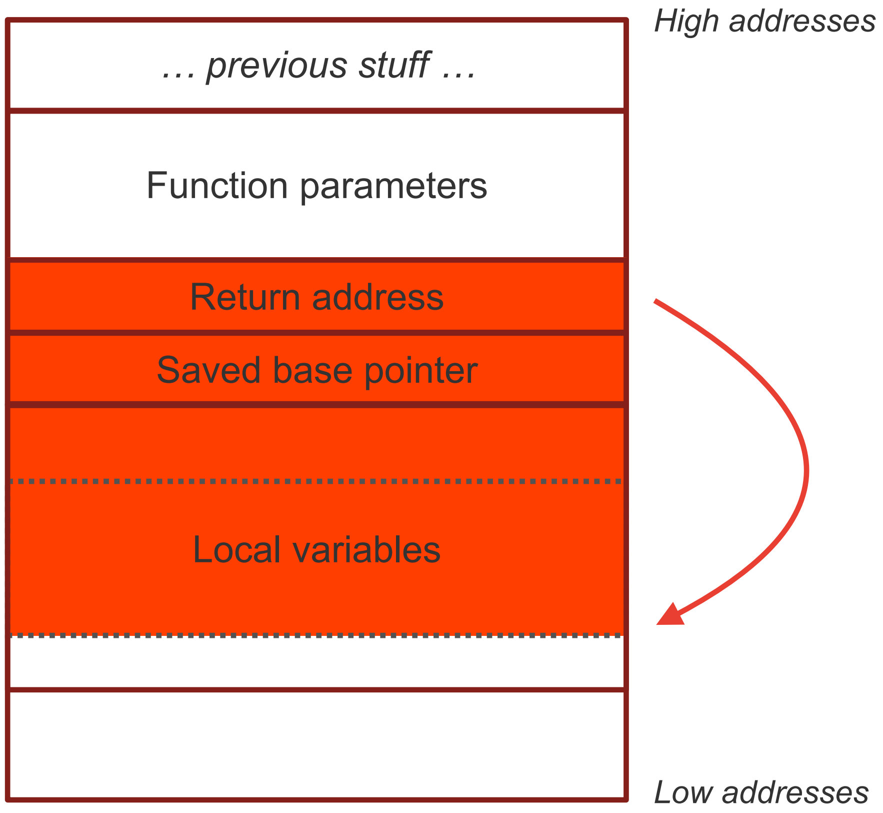
Anatomy of a Stack Frame

```
; push call arguments, in reverse
push    3
push    2
push    1
call    callee    ; call subroutine ‘callee'

    callee:
    push    ebp        ; save old call frame
    mov     ebp, esp   ; initialize new call frame
    ...do stuff...
```



Anatomy of a Stack Frame

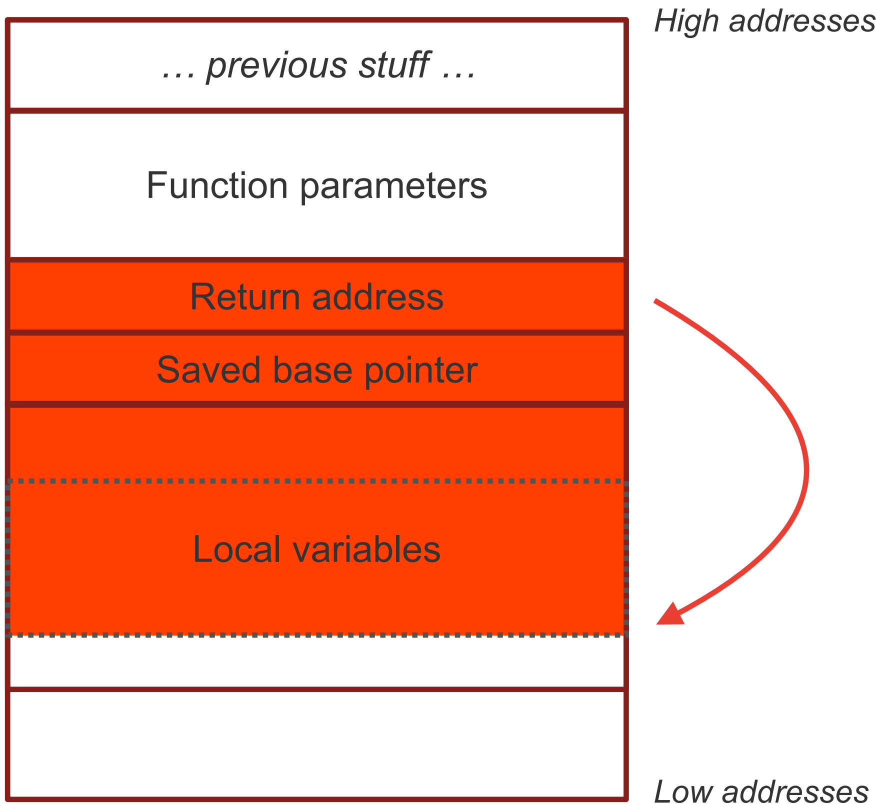
```
; push call arguments, in reverse
push    3
push    2
push    1
call    callee     ; call subroutine ‘callee'

    callee:
    push    ebp        ; save old call frame
    mov     ebp, esp   ; initialize new call frame
    ...do stuff...
```



Anatomy of a Stack Frame

```
; push call arguments, in reverse
push     3
push     2
push     1
call     callee     ; call subroutine ‘callee'

    callee:
    push     ebp         ; save old call frame
    mov      ebp, esp    ; initialize new call frame
    ...do stuff...
    mov      esp, ebp
    pop      ebp         ; restore old call frame
    ret                  ; return
```

💣😓

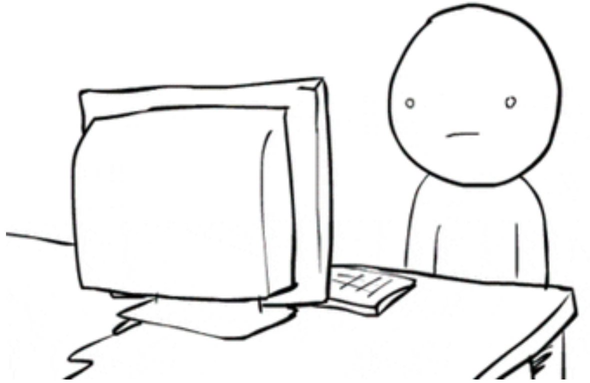


Morris Worm (circa 1988)

```
int main(int argc, char *argv[]) {
  char line[512];
  struct sockaddr_in sin;
  int i, p[2], pid, status;
  i = sizeof (sin);
  if (getpeername(0, &sin, &i) < 0) fatal(argv[0], "getpeername");
  if (gets(line) == NULL) exit(1);
  register char *sp = line;
  ...
  if ((pid = fork()) == 0) {
    close(p[0]);
    if (p[1] != 1) {
      dup2(p[1], 1);
      close(p[1]);
    }
    execv("/usr/ucb/finger", av);
    _exit(1);
  }
  ...
}
```

“Convert a String to Uppercase in C,” circa 2021

```
#include <stdio.h>
#include <string.h>
int main() {
   char s[100];
   int i;
   printf("\nEnter a string :  ");
   gets(s);
   for (i = 0; s[i]!='\0'; i++) {
      if(s[i] >= 'a' && s[i] <= 'z') {
         s[i] = s[i] -32;
      }
   }
   printf("\nString in Upper Case = %s", s);
   return 0;
}
```



Okay, well, I'd know better.

Professional engineers don’t make such silly mistakes, right?

Comprehensive Experimental Analyses of Automotive Attack Surfaces

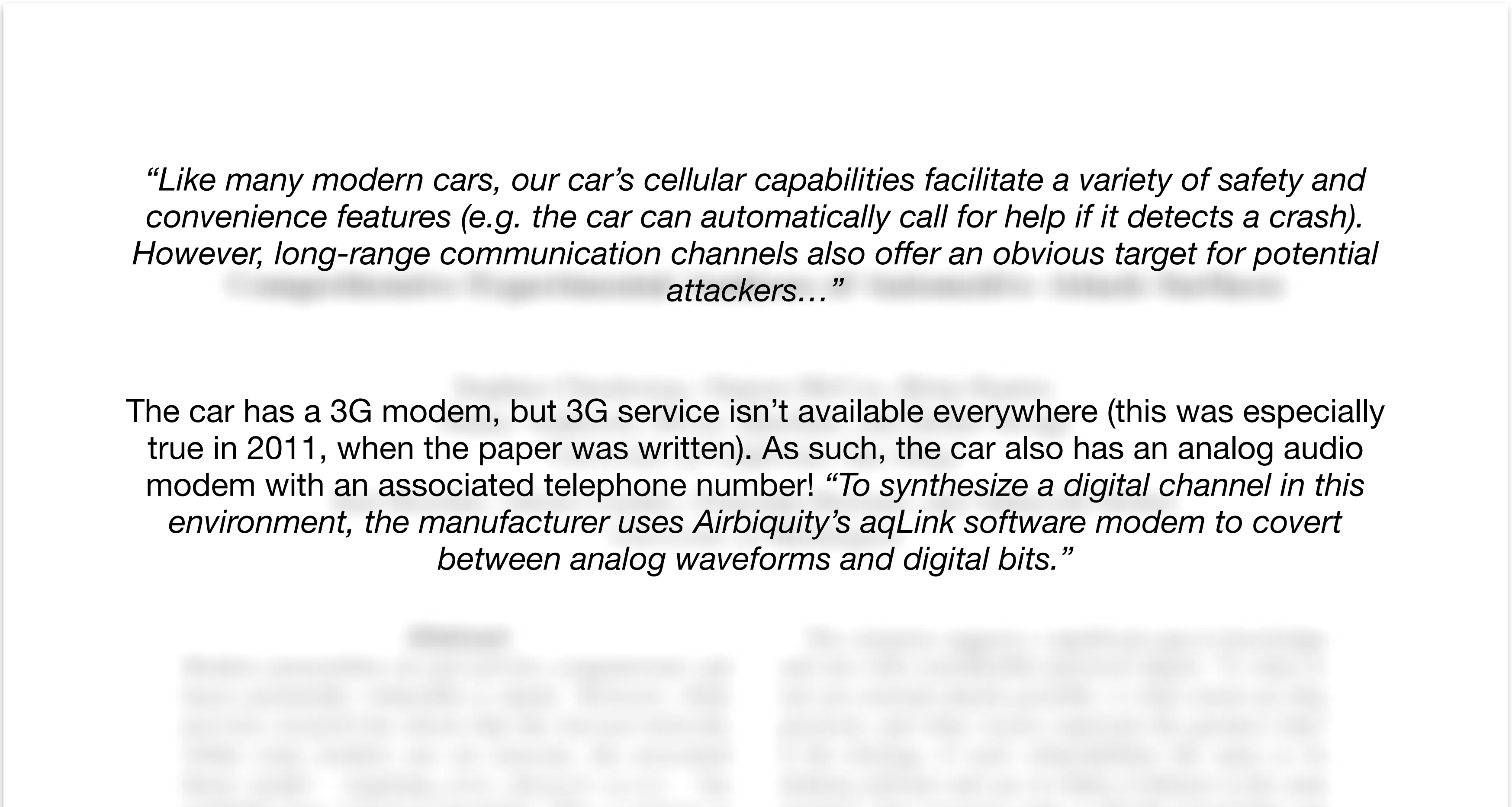
Stephen Checkoway, Damon McCoy, Brian Kantor,
Danny Anderson, Hovav Shacham, and Stefan Savage
University of California, San Diego

Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno
University of Washington

Abstract

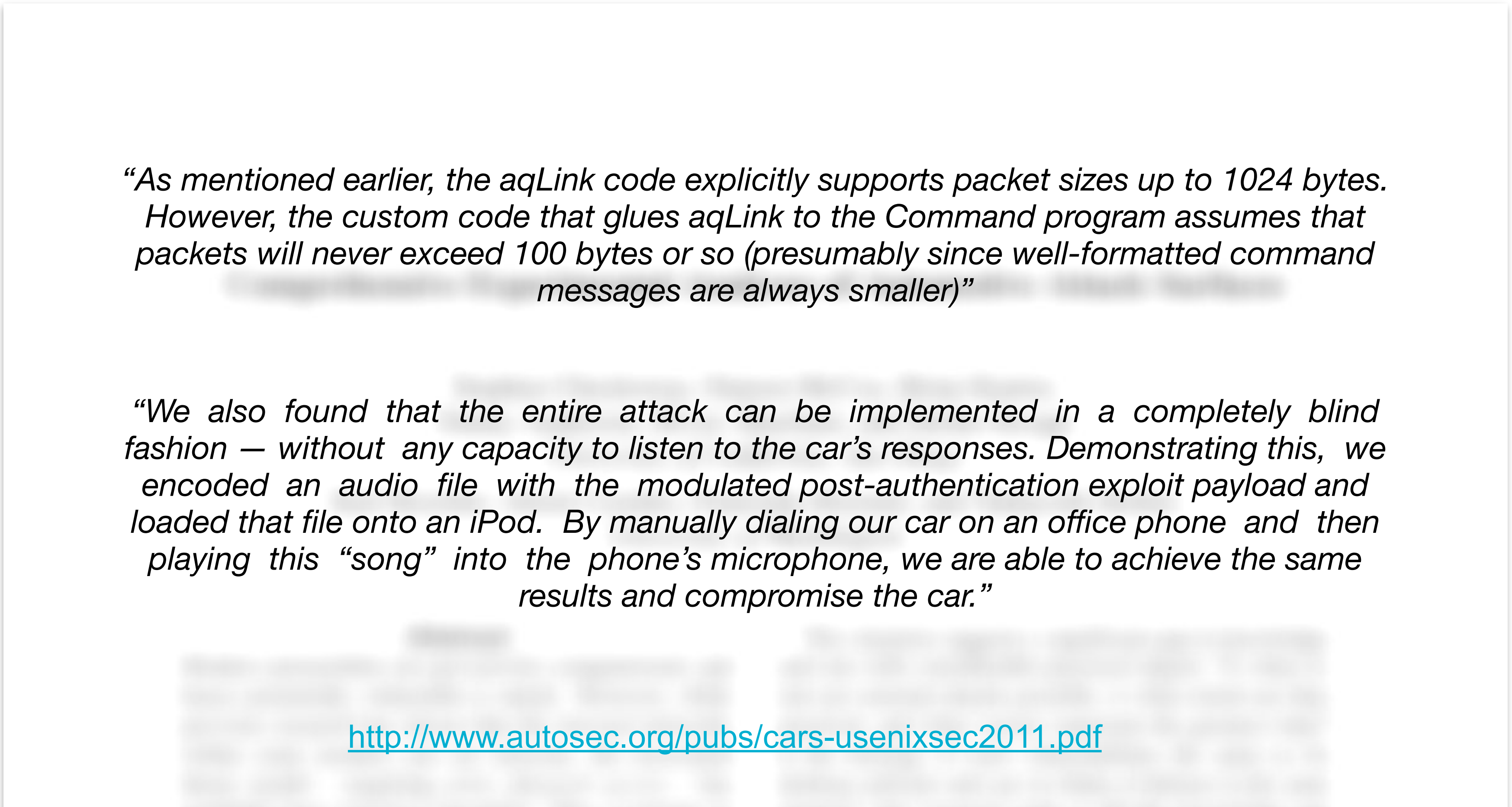
Modern automobiles are pervasively computerized, and hence potentially vulnerable to attack. However, while previous research has shown that the *internal* networks within some modern cars are insecure, the associated threat model — requiring *prior physical access* — has

This situation suggests a significant gap in knowledge, and one with considerable practical import. To what extent are external attacks possible, to what extent are they practical, and what vectors represent the greatest risks? Is the etiology of such vulnerabilities the same as for desktop software and can we think of defense in the same



“Like many modern cars, our car’s cellular capabilities facilitate a variety of safety and convenience features (e.g. the car can automatically call for help if it detects a crash). However, long-range communication channels also offer an obvious target for potential attackers…”

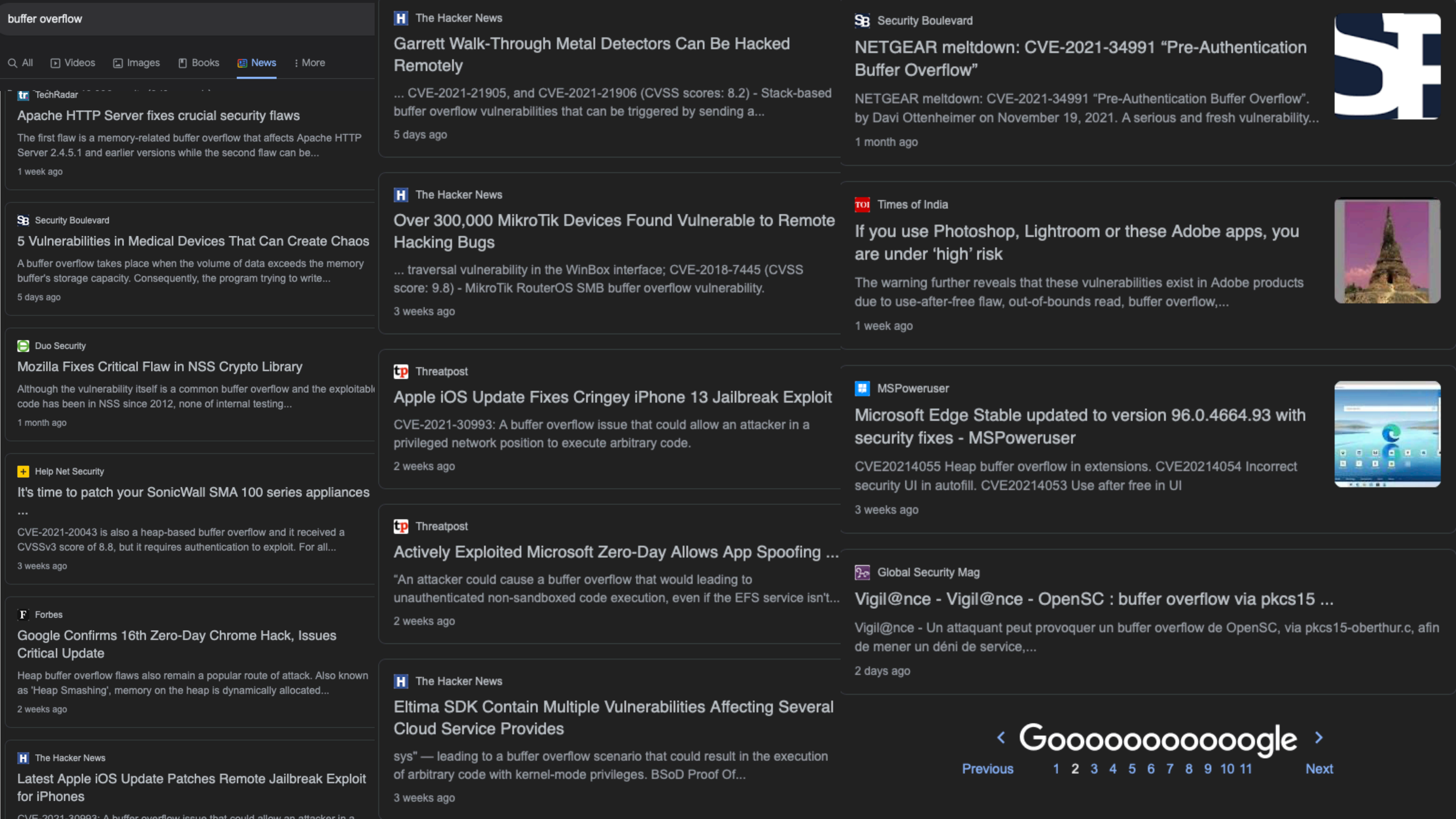
The car has a 3G modem, but 3G service isn’t available everywhere (this was especially true in 2011, when the paper was written). As such, the car also has an analog audio modem with an associated telephone number! *“To synthesize a digital channel in this environment, the manufacturer uses Airbiquity’s aqLink software modem to covert between analog waveforms and digital bits.”*



“As mentioned earlier, the aqLink code explicitly supports packet sizes up to 1024 bytes. However, the custom code that glues aqLink to the Command program assumes that packets will never exceed 100 bytes or so (presumably since well-formatted command messages are always smaller)”

“We also found that the entire attack can be implemented in a completely blind fashion — without any capacity to listen to the car’s responses. Demonstrating this, we encoded an audio file with the modulated post-authentication exploit payload and loaded that file onto an iPod. By manually dialing our car on an office phone and then playing this “song” into the phone’s microphone, we are able to achieve the same results and compromise the car.”

http://www.autosec.org/pubs/cars-usenixsec2011.pdf



buffer overflow

All · Videos · Images · Books · News · More

TechRadar
Apache HTTP Server fixes crucial security flaws
The first flaw is a memory-related buffer overflow that affects Apache HTTP Server 2.4.5.1 and earlier versions while the second flaw can be...
1 week ago

Security Boulevard
5 Vulnerabilities in Medical Devices That Can Create Chaos
A buffer overflow takes place when the volume of data exceeds the memory buffer's storage capacity. Consequently, the program trying to write...
5 days ago

Duo Security
Mozilla Fixes Critical Flaw in NSS Crypto Library
Although the vulnerability itself is a common buffer overflow and the exploitabl code has been in NSS since 2012, none of internal testing...
1 month ago

Help Net Security
It's time to patch your SonicWall SMA 100 series appliances ...
CVE-2021-20043 is also a heap-based buffer overflow and it received a CVSSv3 score of 8.8, but it requires authentication to exploit. For all...
3 weeks ago

Forbes
Google Confirms 16th Zero-Day Chrome Hack, Issues Critical Update
Heap buffer overflow flaws also remain a popular route of attack. Also known as 'Heap Smashing', memory on the heap is dynamically allocated...
2 weeks ago

The Hacker News
Latest Apple iOS Update Patches Remote Jailbreak Exploit for iPhones
CVE-2021-30993: A buffer overflow issue that could allow an attacker in a

The Hacker News
Garrett Walk-Through Metal Detectors Can Be Hacked Remotely
... CVE-2021-21905, and CVE-2021-21906 (CVSS scores: 8.2) - Stack-based buffer overflow vulnerabilities that can be triggered by sending a...
5 days ago

The Hacker News
Over 300,000 MikroTik Devices Found Vulnerable to Remote Hacking Bugs
... traversal vulnerability in the WinBox interface; CVE-2018-7445 (CVSS score: 9.8) - MikroTik RouterOS SMB buffer overflow vulnerability.
3 weeks ago

Threatpost
Apple iOS Update Fixes Cringey iPhone 13 Jailbreak Exploit
CVE-2021-30993: A buffer overflow issue that could allow an attacker in a privileged network position to execute arbitrary code.
2 weeks ago

Threatpost
Actively Exploited Microsoft Zero-Day Allows App Spoofing ...
"An attacker could cause a buffer overflow that would leading to unauthenticated non-sandboxed code execution, even if the EFS service isn't...
2 weeks ago

The Hacker News
Eltima SDK Contain Multiple Vulnerabilities Affecting Several Cloud Service Provides
sys" — leading to a buffer overflow scenario that could result in the execution of arbitrary code with kernel-mode privileges. BSoD Proof Of...
3 weeks ago

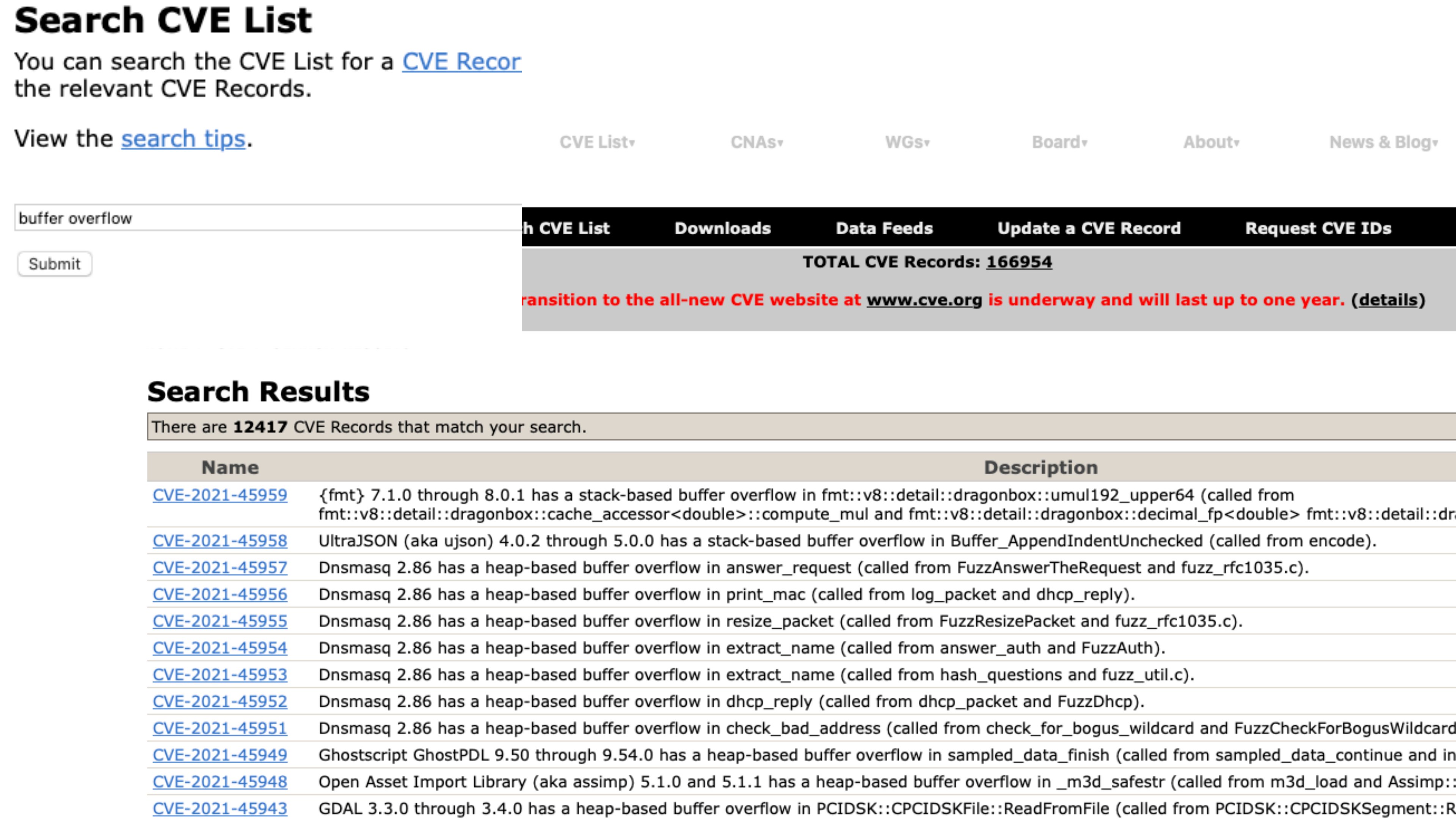
Security Boulevard
NETGEAR meltdown: CVE-2021-34991 "Pre-Authentication Buffer Overflow"
NETGEAR meltdown: CVE-2021-34991 "Pre-Authentication Buffer Overflow". by Davi Ottenheimer on November 19, 2021. A serious and fresh vulnerability...
1 month ago

Times of India
If you use Photoshop, Lightroom or these Adobe apps, you are under 'high' risk
The warning further reveals that these vulnerabilities exist in Adobe products due to use-after-free flaw, out-of-bounds read, buffer overflow,...
1 week ago

MSPoweruser
Microsoft Edge Stable updated to version 96.0.4664.93 with security fixes - MSPoweruser
CVE20214055 Heap buffer overflow in extensions. CVE20214054 Incorrect security UI in autofill. CVE20214053 Use after free in UI
3 weeks ago

Global Security Mag
Vigil@nce - Vigil@nce - OpenSC : buffer overflow via pkcs15 ...
Vigil@nce - Un attaquant peut provoquer un buffer overflow de OpenSC, via pkcs15-oberthur.c, afin de mener un déni de service,...
2 days ago

< Goooooooooooogle >
Previous 1 2 3 4 5 6 7 8 9 10 11 Next



Search CVE List

You can search the CVE List for a CVE Recor
the relevant CVE Records.

View the search tips.

buffer overflow

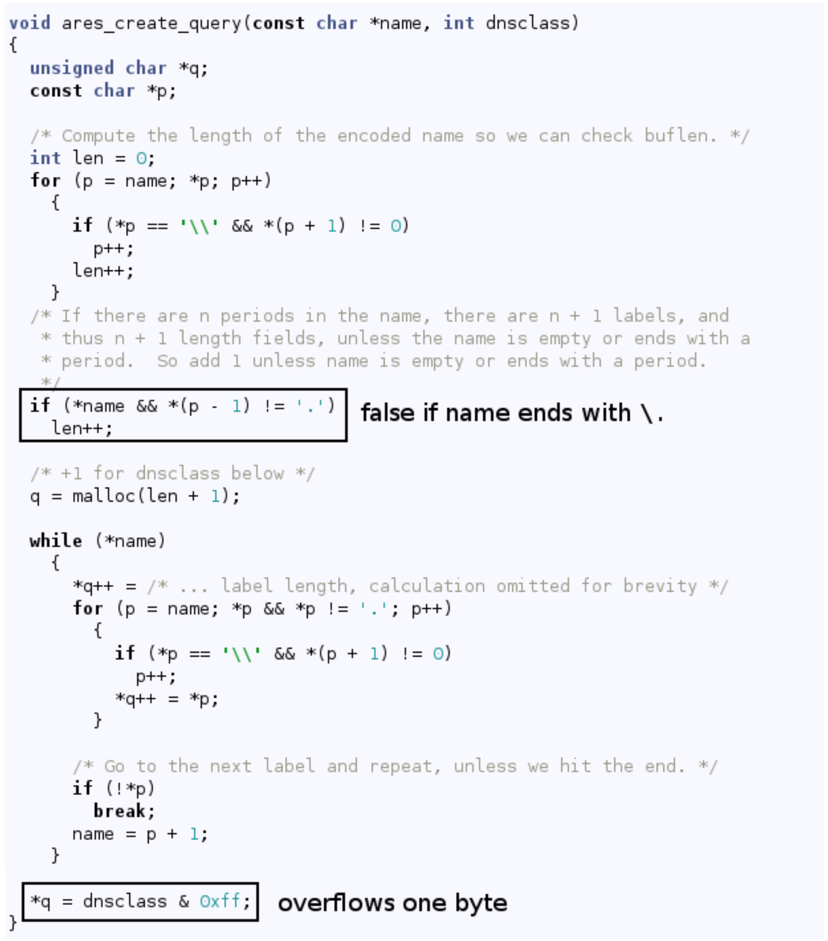
TOTAL CVE Records: 166954

ransition to the all-new CVE website at www.cve.org is underway and will last up to one year. (details)

Search Results

There are **12417** CVE Records that match your search.

Name	Description
CVE-2021-45959	{fmt} 7.1.0 through 8.0.1 has a stack-based buffer overflow in fmt::v8::detail::dragonbox::umul192_upper64 (called from fmt::v8::detail::dragonbox::cache_accessor<double>::compute_mul and fmt::v8::detail::dragonbox::decimal_fp<double> fmt::v8::detail::dr
CVE-2021-45958	UltraJSON (aka ujson) 4.0.2 through 5.0.0 has a stack-based buffer overflow in Buffer_AppendIndentUnchecked (called from encode).
CVE-2021-45957	Dnsmasq 2.86 has a heap-based buffer overflow in answer_request (called from FuzzAnswerTheRequest and fuzz_rfc1035.c).
CVE-2021-45956	Dnsmasq 2.86 has a heap-based buffer overflow in print_mac (called from log_packet and dhcp_reply).
CVE-2021-45955	Dnsmasq 2.86 has a heap-based buffer overflow in resize_packet (called from FuzzResizePacket and fuzz_rfc1035.c).
CVE-2021-45954	Dnsmasq 2.86 has a heap-based buffer overflow in extract_name (called from answer_auth and FuzzAuth).
CVE-2021-45953	Dnsmasq 2.86 has a heap-based buffer overflow in extract_name (called from hash_questions and fuzz_util.c).
CVE-2021-45952	Dnsmasq 2.86 has a heap-based buffer overflow in dhcp_reply (called from dhcp_packet and FuzzDhcp).
CVE-2021-45951	Dnsmasq 2.86 has a heap-based buffer overflow in check_bad_address (called from check_for_bogus_wildcard and FuzzCheckForBogusWildcard
CVE-2021-45949	Ghostscript GhostPDL 9.50 through 9.54.0 has a heap-based buffer overflow in sampled_data_finish (called from sampled_data_continue and in
CVE-2021-45948	Open Asset Import Library (aka assimp) 5.1.0 and 5.1.1 has a heap-based buffer overflow in _m3d_safestr (called from m3d_load and Assimp::
CVE-2021-45943	GDAL 3.3.0 through 3.4.0 has a heap-based buffer overflow in PCIDSK::CPCIDSKFile::ReadFromFile (called from PCIDSK::CPCIDSKSegment::R



```c
void ares_create_query(const char *name, int dnsclass)
{
  unsigned char *q;
  const char *p;

  /* Compute the length of the encoded name so we can check buflen. */
  int len = 0;
  for (p = name; *p; p++)
    {
      if (*p == '\\' && *(p + 1) != 0)
        p++;
      len++;
    }
  /* If there are n periods in the name, there are n + 1 labels, and
   * thus n + 1 length fields, unless the name is empty or ends with a
   * period.  So add 1 unless name is empty or ends with a period.
   */
  if (*name && *(p - 1) != '.')
    len++;

  /* +1 for dnsclass below */
  q = malloc(len + 1);

  while (*name)
    {
      *q++ = /* ... label length, calculation omitted for brevity */
      for (p = name; *p && *p != '.'; p++)
        {
          if (*p == '\\' && *(p + 1) != 0)
            p++;
          *q++ = *p;
        }

      /* Go to the next label and repeat, unless we hit the end. */
      if (!*p)
        break;
      name = p + 1;
    }

  *q = dnsclass & 0xff;
}
```

false if name ends with \.

overflows one byte

One-byte overflow in Chrome OS:
https://googleprojectzero.blogspot.com/2016/12/chrome-os-exploit-one-byte-overflow-and.html

Spot the overflow

```
char buffer[128];
int bytesToCopy = packet.length;
if (bytesToCopy < 128) {
    strncpy(buffer, packet.data, bytesToCopy);
}
```

Spot the overflow

```
char buffer[128];
int bytesToCopy = packet.length;
if (bytesToCopy < 128) {
    strncpy(buffer, packet.data, bytesToCopy);
}
```

✓ Proper bounds check

✓ Use of strncpy (avoiding unsafe strcpy)

Spot the overflow

Signed

```
char buffer[128];
int bytesToCopy = packet.length;
if (bytesToCopy < 128) {
    strncpy(buffer, packet.data, bytesToCopy);
}
```

Cast to size_t (unsigned)

How can we find and/or prevent problems like this?

This is the topic of this whole class :)

How can we find and/or prevent problems like this?

- Dynamic analysis: Run the program, watch what it does, and look for problematic behavior *[more in next lecture!]*
- Static analysis: read the source code and try to spot the issues *[more in next lecture!]*
- Write code differently: create habits and frameworks that make it harder to produce these kinds of mistakes *[more throughout the class!]*
- Sandbox: accept that these issues will happen, but try to minimize the consequences *[more in future lecture on browsers!]*

How can we find and/or prevent problems like this?

- Dynamic analysis: Run the program, watch what it does, and look for problematic behavior *[more in next lecture!]*
    - What if the problematic behavior occurs in some edge case that doesn't show up in testing?
- Static analysis: read the source code and try to spot the issues *[more in next lecture!]*
- Write code differently: create habits and frameworks that make it harder to produce these kinds of mistakes *[more throughout the class!]*
- Sandbox: accept that these issues will happen, but try to minimize the consequences *[more in future lecture on browsers!]*

How can we find and/or prevent problems like this?

- Dynamic analysis: Run the program, watch what it does, and look for problematic behavior *[more in next lecture!]*
- Static analysis: read the source code and try to spot the issues *[more in next lecture!]*
    - So you think you can spot every issue ever?
    - *(It's mathematically provable that you can't.)*
- Write code differently: create habits and frameworks that make it harder to produce these kinds of mistakes *[more throughout the class!]*
- Sandbox: accept that these issues will happen, but try to minimize the consequences *[more in future lecture on browsers!]*

How can we find and/or prevent problems like this?

- Dynamic analysis: Run the program, watch what it does, and look for problematic behavior *[more in next lecture!]*
- Static analysis: read the source code and try to spot the issues *[more in next lecture!]*
- Write code differently: create habits and frameworks that make it harder to produce these kinds of mistakes *[more throughout the class!]*
    - This is where Rust -- and thinking about the philosophy / design choices behind Rust -- comes in.
    - Possibly makes programming harder? Need to re-train engineers?
- Sandbox: accept that these issues will happen, but try to minimize the consequences *[more in future lecture on browsers!]*
    - Equally important!

About CS 110L 👋

Course outline

- Key question: How can we prevent common mistakes in systems programming?
    - This is not a Rust class, although almost all of our programming will be done in Rust
    - How do we find and prevent common mistakes in C/C++?
    - How does Rust's type system prevent common memory safety errors?
    - How do you architect good code?
    - Avoiding multiprocessing pitfalls
    - Avoiding multithreading pitfalls
    - Putting all of this into practice: Networked systems

Course outline

- Corequisite: CS 110
- Pass/fail
  - You will get out what you put in
- Components:
  - Lecture
  - Weekly exercises (40%)
  - Two projects (40%)
  - Participation (20%)
    - Coming to & participating in lecture
    - Asking/answering questions on Slack

Missing classes

- Class is officially in-person (if we can do so in a safer way)
- Communicate with me! Email or Slack.
- If we need a more rigorous hybrid option, I'll try to make one work
- We have recordings of lectures from previous quarters
- Happy to give extensions

Projects

- Project 1: Mini GDB
- Project 2: High-performance web server
- Functionality grading only
    - The Rust compiler will be your interactive style grader!
- These projects are intended to give you additional experience in building real systems, while having to think about some of the safety issues we’re discussing. These may seem intimidating, but they really aren't!
- Working in groups is encouraged!
- Have a different idea? Let me know!

Exercises

- Each week (ish), there will be small programming problems to reinforce the week’s lecture material
- Expected time: 1-3 hours
- In addition, you’ll be asked occasionally to complete an anonymous survey about how the class is going and how we/I can improve

Work for Wednesday

Fill out this intro form: https://forms.gle/gjep8hA4J637amC5A
Join the Slack (Canvas sidebar -> Slack -> Join. All communication will be there!)

(Slides will be posted on website shortly after class.)