# Binary Reverse Engineering And Analysis

## Course 10: Exp++

Caragea Radu

April 20, 2021

# Recap for EXP

- Classes of vulnerabilities
- Compiler and system mitigations
- Bypasses

# Skipped topics

- Sandboxing binary applications and system-level defenses
- Symbolic execution in vulnerability discovery
- Application fuzzing

# Finding bugs in large code bases

- What to look for?
- Where to look?
- How to look?

# What to look for?

- Variant analysis
- Older code (written by security-oblivious dinosaurs)
- 3AM code (written by night owls)
- Probabilistic approach (whose code is more error-prone?)

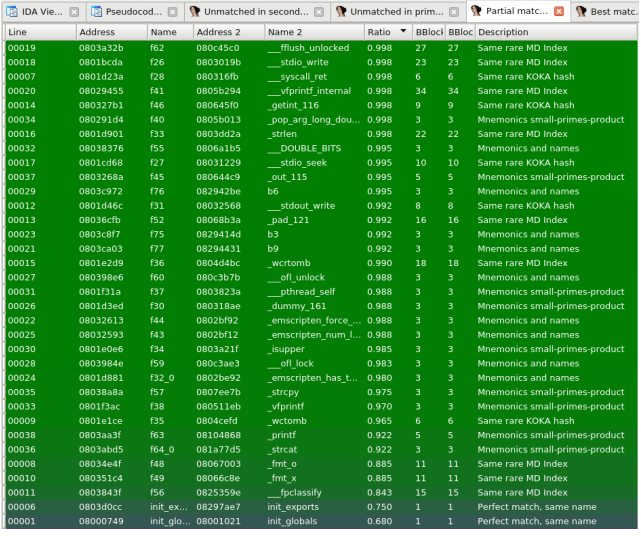
# Where to look? (open source code)

- Full commit log and Issue Tracker should be available
- Security issues should also be public eventually
- Straight-forward approach

# Where to look? (closed source code)

- Full commit log unavailable
- Individual release binaries available (sometimes Release Notes)
- Binary Diffing
- Security issues usually appear as fixed
- Harder approach but not impossible

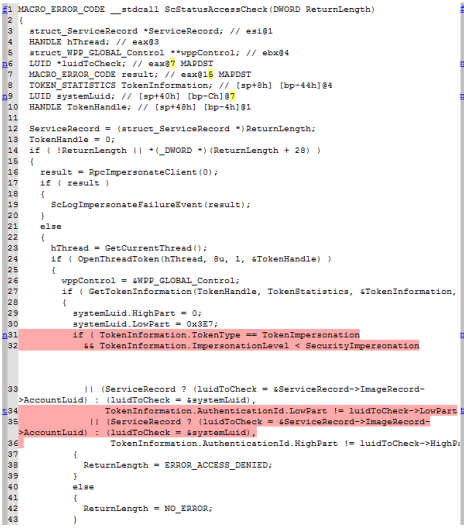
# Binary Diffing with Diaphora (1)



IDA Vie... | Pseudocod... | Unmatched in second... | Unmatched in prim... | Partial matc... | Best matc.

| Line | Address | Name | Address 2 | Name 2 | Ratio | BBlocl | BBloc | Description |
|---|---|---|---|---|---|---|---|---|
| 00019 | 0803a32b | f62 | 080c45c0 | __fflush_unlocked | 0.998 | 27 | 27 | Same rare MD Index |
| 00018 | 0801bcda | f26 | 0803019b | __stdio_write | 0.998 | 23 | 23 | Same rare MD Index |
| 00007 | 0801d23a | f28 | 080316fb | __syscall_ret | 0.998 | 6 | 6 | Same rare KOKA hash |
| 00020 | 08029455 | f41 | 0805b294 | __vfprintf_internal | 0.998 | 34 | 34 | Same rare MD Index |
| 00014 | 080327b1 | f46 | 080645f0 | _getint_116 | 0.998 | 9 | 9 | Same rare KOKA hash |
| 00034 | 080291d4 | f40 | 0805b013 | _pop_arg_long_dou... | 0.998 | 3 | 3 | Mnemonics small-primes-product |
| 00016 | 0801d901 | f33 | 0803dd2a | _strlen | 0.998 | 22 | 22 | Same rare MD Index |
| 00032 | 08038376 | f55 | 0806a1b5 | __DOUBLE_BITS | 0.995 | 3 | 3 | Mnemonics and names |
| 00017 | 0801cd68 | f27 | 08031229 | __stdio_seek | 0.995 | 10 | 10 | Same rare KOKA hash |
| 00037 | 0803268a | f45 | 080644c9 | _out_115 | 0.995 | 5 | 5 | Mnemonics small-primes-product |
| 00029 | 0803c972 | f76 | 082942be | b6 | 0.995 | 3 | 3 | Mnemonics and names |
| 00012 | 0801d46c | f31 | 08032568 | __stdout_write | 0.992 | 8 | 8 | Same rare KOKA hash |
| 00013 | 08036cfb | f52 | 08068b3a | _pad_121 | 0.992 | 16 | 16 | Same rare MD Index |
| 00023 | 0803c8f7 | f75 | 0829414d | b3 | 0.992 | 3 | 3 | Mnemonics and names |
| 00021 | 0803ca03 | f77 | 08294431 | b9 | 0.992 | 3 | 3 | Mnemonics and names |
| 00015 | 0801e2d9 | f36 | 0804d4bc | _wcrtomb | 0.990 | 18 | 18 | Same rare MD Index |
| 00027 | 080398e6 | f60 | 080c3b7b | __ofl_unlock | 0.988 | 3 | 3 | Mnemonics and names |
| 00031 | 0801f31a | f37 | 0803823a | __pthread_self | 0.988 | 3 | 3 | Mnemonics small-primes-product |
| 00026 | 0801d3ed | f30 | 080318ae | _dummy_161 | 0.988 | 3 | 3 | Mnemonics small-primes-product |
| 00022 | 08032613 | f44 | 0802bf92 | _emscripten_force_... | 0.988 | 3 | 3 | Mnemonics and names |
| 00025 | 08032593 | f43 | 0802bf12 | _emscripten_num_l... | 0.988 | 3 | 3 | Mnemonics and names |
| 00030 | 0801e0e6 | f34 | 0803a21f | _isupper | 0.985 | 3 | 3 | Mnemonics small-primes-product |
| 00028 | 0803984e | f59 | 080c3ae3 | __ofl_lock | 0.983 | 3 | 3 | Mnemonics and names |
| 00024 | 0801d881 | f32_0 | 0802be92 | _emscripten_has_t... | 0.980 | 3 | 3 | Mnemonics and names |
| 00035 | 08038a8a | f57 | 0807ee7b | _strcpy | 0.975 | 3 | 3 | Mnemonics small-primes-product |
| 00033 | 0801f3ac | f38 | 080511eb | _vfprintf | 0.970 | 3 | 3 | Mnemonics small-primes-product |
| 00009 | 0801e1ce | f35 | 0804cefd | _wctomb | 0.965 | 6 | 6 | Same rare KOKA hash |
| 00038 | 0803aa3f | f63 | 08104868 | _printf | 0.922 | 5 | 5 | Mnemonics small-primes-product |
| 00036 | 0803abd5 | f64_0 | 081a77d5 | _strcat | 0.922 | 3 | 3 | Mnemonics small-primes-product |
| 00008 | 08034e4f | f48 | 08067003 | _fmt_o | 0.885 | 11 | 11 | Same rare MD Index |
| 00010 | 080351c4 | f49 | 08066c8e | _fmt_x | 0.885 | 11 | 11 | Same rare MD Index |
| 00011 | 0803843f | f56 | 0825359e | __fpclassify | 0.843 | 15 | 15 | Same rare MD Index |
| 00006 | 0803d0cc | init_ex... | 08297ae7 | init_exports | 0.750 | 1 | 1 | Perfect match, same name |
| 00001 | 08000749 | init_glo... | 08001021 | init_globals | 0.680 | 1 | 1 | Perfect match, same name |

# Binary Diffing with Diaphora (2)



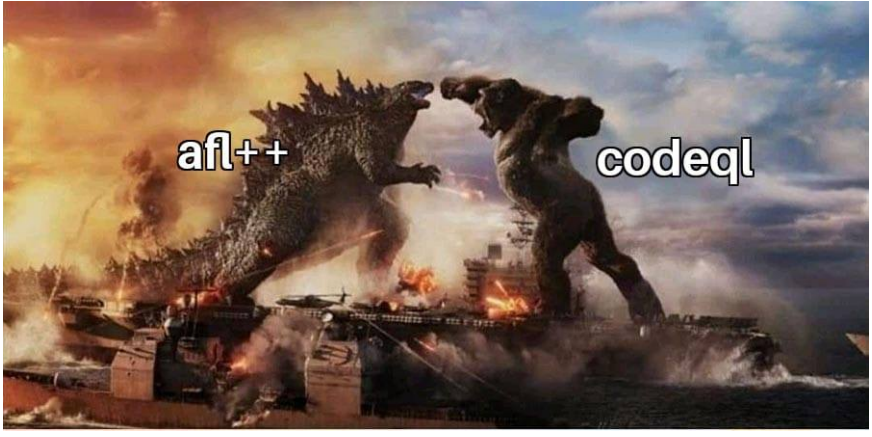
```
MACRO_ERROR_CODE __stdcall ScStatusAccessCheck(DWORD ReturnLength)
{
  struct_ServiceRecord *ServiceRecord; // esi@1
  HANDLE hThread; // eax@3
  struct_WPP_GLOBAL_Control **wppControl; // ebx@4
  LUID *luidToCheck; // eax@7 MAPDST
  MACRO_ERROR_CODE result; // eax@15 MAPDST
  TOKEN_STATISTICS TokenInformation; // [sp+8h] [bp-44h]@4
  LUID systemLuid; // [sp+40h] [bp-Ch]@7
  HANDLE TokenHandle; // [sp+48h] [bp-4h]@1

  ServiceRecord = (struct_ServiceRecord *)ReturnLength;
  TokenHandle = 0;
  if ( !ReturnLength || *(_DWORD *)(ReturnLength + 28) )
  {
    result = RpcImpersonateClient(0);
    if ( result )
    {
      ScLogImpersonateFailureEvent(result);
    }
    else
    {
      hThread = GetCurrentThread();
      if ( OpenThreadToken(hThread, 8u, 1, &TokenHandle) )
      {
        wppControl = &WPP_GLOBAL_Control;
        if ( GetTokenInformation(TokenHandle, TokenStatistics, &TokenInformation,
        {
          systemLuid.HighPart = 0;
          systemLuid.LowPart = 0x3E7;
          if ( TokenInformation.TokenType == TokenImpersonation
            && TokenInformation.ImpersonationLevel < SecurityImpersonation


              || (ServiceRecord ? (luidToCheck = &ServiceRecord->ImageRecord-
  >AccountLuid) : (luidToCheck = &systemLuid),
                  TokenInformation.AuthenticationId.LowPart != luidToCheck->LowPart
              || (ServiceRecord ? (luidToCheck = &ServiceRecord->ImageRecord-
  >AccountLuid) : (luidToCheck = &systemLuid),
                  TokenInformation.AuthenticationId.HighPart != luidToCheck->HighPa
          {
            ReturnLength = ERROR_ACCESS_DENIED;
          }
          else
          {
            ReturnLength = NO_ERROR;
          }
```

```
MACRO_ERROR_CODE __stdcall ScStatusAccessCheck(DWORD ReturnLength)
{
  struct_ServiceRecord *ServiceRecord; // esi@1
  HANDLE hThread; // eax@3
  struct_WPP_GLOBAL_Control **wppControl; // ebx@4
  LUID *luidToCheck; // eax@6 MAPDST
  MACRO_ERROR_CODE result; // eax@14 MAPDST
  TOKEN_STATISTICS TokenInformation; // [sp+8h] [bp-44h]@4
  LUID systemLuid; // [sp+40h] [bp-Ch]@8
  HANDLE TokenHandle; // [sp+48h] [bp-4h]@1

  ServiceRecord = (struct_ServiceRecord *)ReturnLength;
  TokenHandle = 0;
  if ( !ReturnLength || *(_DWORD *)(ReturnLength + 28) )
  {
    result = RpcImpersonateClient(0);
    if ( result )
    {
      ScLogImpersonateFailureEvent(result);
    }
    else
    {
      hThread = GetCurrentThread();
      if ( OpenThreadToken(hThread, 8u, 1, &TokenHandle) )
      {
        wppControl = &WPP_GLOBAL_Control;
        if ( GetTokenInformation(TokenHandle, TokenStatistics, &TokenInformation,
        {
          systemLuid.HighPart = 0;
          systemLuid.LowPart = 0x3E7;
          if ( ServiceRecord )
            luidToCheck = &ServiceRecord->ImageRecord->AccountLuid;
          else
            luidToCheck = &systemLuid;
          if ( TokenInformation.AuthenticationId.LowPart != luidToCheck->LowPart
            || (ServiceRecord ? (luidToCheck = &ServiceRecord->ImageRecord-
  >AccountLuid) : (luidToCheck = &systemLuid),


                TokenInformation.AuthenticationId.HighPart != luidToCheck->HighPa
          {
            ReturnLength = ERROR_ACCESS_DENIED;
          }
          else
          {
            ReturnLength = NO_ERROR;
          }
```

# How to look?

- Manual inspection
  - Art/Difficult to master
  - Time consuming
  - Success is not guaranteed either
- Fuzzing
  - Custom fuzzers can be repurposed
  - Limited only by creativity
  - Throw more CPUs, more results
  - Unattended until triage
- Other approaches?

# Battle of the giants



# CodeQL

- Semantic code analysis engine
- Transforms a given codebase into a database
- Finds instances of vulnerabilities according to a query
- Useful if you already know what you're looking for
- Q: Where would this fit? (discussed today)

# CodeQL

- Semantic code analysis engine
- Transforms a given codebase into a database
- Finds instances of vulnerabilities according to a query
- Useful if you already know what you're looking for
- Q: Where would this fit? (discussed today)
- A: Variant analysis on Open Source code

# CodeQL applicability

- C/C++
- C#
- Go
- Java
- Javascript
- Python
- TypeScript
- COBOL (deprecated)
- Ruby (soon)

# CodeQL tutorials

Free resources:

- help.semmle.com/QL/learn-ql/index.html
- tiny.cc/offensivecon
- securitylab.github.com/ctf
- lab.github.com/githubtraining/codeql-u-boot-challenge-(cc++)
- cyberark.com/resources/threat-research-blog/make-memcpy-safe-again-codeql

# Going further: Resources

- Modern Binary Exploitation from RPISEC: `github.com/RPISEC/MBE`
- Google Project Zero: `googleprojectzero.blogspot.com`
- LiveOverflowCTF: `www.youtube.com/c/LiveOverflowCTF/videos`
- gamozolabs: `www.youtube.com/user/gamozolabs/videos`
- Orange's blog: `blog.orange.tw`

# Going further: Conferences

Most have videos on YouTube

- DEF CON: `https://www.defcon.org/`
- CCC: `media.ccc.de/c/35c3`
- Hack in the Box: `conference.hitb.org`
- OffensiveCon: `www.offensivecon.org`
- RECON: `recon.cx/2020/montreal/video/`
- Usenix ENIGMA: `www.youtube.com/c/USENIXEnigmaConference/videos`

# Practice

- https://pwnthybytes.ro/unibuc_re/10-lab.html

# Thanks

- Thank you for attending this course!
- Feedback appreciated: `https://forms.gle/a6cPHxkmvJTiRgdb9`
- Contact:
  - Twitter @mztropics
  - LinkedIn: `https://linkedin.com/in/radu-caragea`