Writing Secure Software is hard, but at least add mitigations!

Simon Roses Femerling
CEO
VULNEX
@simonroses
ME?

- Simon Roses Femerling
  - Founder & CEO, VULNEX [www.vulnex.com](http://www.vulnex.com)
  - Blog: [www.simonroses.com](http://www.simonroses.com)
  - Twitter: @simonroses
  - Former Microsoft, PwC, @Stake
  - DARPA Cyber Fast Track award on software security project
  - Black Hat, RSA, OWASP, SOURCE, AppSec, DeepSec, MSFT TECHNET
BIG THANKS!

- DARPA Cyber Fast Track (CFT)
- Mudge
- The fine folks at BIT SYSTEMS
TALK OBJECTIVES

- Secure development
- Verify software security posture
AGENDA

1. Secure Development
2. Security Mitigations
3. BinSecSweeper
4. Case Studies
5. Conclusions
1. Secure Development
1. MICROSOFT SDL
1. OPENSAMM
1. IT’S ABOUT SAVING MONEY!

Figure 1: Cost of Bug Elimination in the Software Development Lifecycle [NIST 2002]
1. LET’S AVOID

D-LINK ROUTER BACKDOOR

CVE-2013-6462: Stack buffer overflow (20 years old)

Multiple CVEs:
• CVE-2013-5359
• CVE-2013-5358
• CVE-2013-5357
• CVE-2013-5349
1. BINARY INTELLIGENCE

**File Information**
- Size
- Hash
- Timestamp
- Strings

**Compiler**
- Name
- Version

**Security Mitigations**
- DEP
- ASLR
- Stack Cookies

**Vulnerabilities**
- Unsafe API
- Weak Crypto
- Backdoors
2. Security Mitigations
2. SOME COMPILERS OFFER GOOD SECURITY DEFENSES

- Visual Studio
- GCC
- LLVM (Xcode)
2. SDL MICROSOFT GUIDE 5.2

- “Use minimum code generation suite and libraries. For unmanaged, native C/C++ code, use Visual C++ 2010 as it offers all the SDL-mandated compiler and linker flags, including /GS, /DYNAMICBASE, /NXCOMPAT, and /SAFESEH. For managed code, use Visual Studio® 2008 SP1 or later. Use the currently required (or later) versions of compilers to compile options for the Win32®, Win64, WinCE, and Macintosh target platforms, as listed in Appendix E: SDL Required and Recommended Compilers, Tools, and Options for All Platforms. The biggest change in Visual Studio 2008 SP1 and later is Data Execution Prevention (DEP) support, enabled by default for all binaries, which can help protect against classes of buffer overrun.”

- “For unmanaged C or C++ code, BinScope must indicate a "Pass" in the compiler version field for all binaries. For managed code, an attestation is required that the compiler version used to ship the product is the version outlined in this document or later.”

- “Banned application programming interfaces (APIs). All native C and C++ code must not use banned versions of string buffer handling functions.”

2. Visual Studio Defenses

1) Only available in Visual Studio Ultimate
2) Defense enhanced
2. Visual Studio Defenses

- Stack buffer protection (/GS)
- Code Analysis
- Data Execution Prevention (DEP)
- Address Space Layout Randomization (ASLR)
- Security Development Lifecycle (/SDL)(VS 2012)
  - /sdl causes SDL mandatory compiler warnings to be treated as errors during compilation.
  - /sdl enables additional code generation features such as increasing the scope of stack buffer overrun protection and initialization or sanitization of pointers in a limited set of well-defined scenarios.
2. GCC Defenses

- **GCC 4.3**
  - `-Wall`
  - `-Wformat-security`
  - `-Wformat`
  - `-fstack-protector`
  - `-Wstack-protector`
  - `-fstack-protector-all`
  - `-Wstack-protector-all`
  - `-z relo`
  - `-fPIE`
  - `-pie`
  - `-D_FORTIFY_SOURCE=1`
  - `-D_FORTIFY_SOURCE=2`

- **GCC 4.4**
  - `-Wall`
  - `-Wformat-security`
  - `-Wformat`
  - `-fstack-protector`
  - `-Wstack-protector`
  - `-fstack-protector-all`
  - `-Wstack-protector-all`
  - `-z relo`
  - `-fPIE`
  - `-pie`
  - `-D_FORTIFY_SOURCE=1`
  - `-D_FORTIFY_SOURCE=2`

- **GCC 4.6**
  - `-Wall`
  - `-Wformat-security`
  - `-Wformat`
  - `-fstack-protector`
  - `-Wstack-protector`
  - `-fstack-protector-all`
  - `-Wstack-protector-all`
  - `-z relo`
  - `-fPIE`
  - `-pie`
  - `-D_FORTIFY_SOURCE=1`
  - `-D_FORTIFY_SOURCE=2`

- **GCC 4.7**
  - `-Wall`
  - `-Wformat-security`
  - `-Wformat`
  - `-fstack-protector`
  - `-Wstack-protector`
  - `-fstack-protector-all`
  - `-Wstack-protector-all`
  - `-z relo`
  - `-fPIE`
  - `-pie`
  - `-D_FORTIFY_SOURCE=1`
  - `-D_FORTIFY_SOURCE=2`
2. GCC SECURITY

- Decent security defenses by not enabled by default

- Mudflap Pointer Debugging (removed in GCC 4.9, in favor of Address Sanitizer)
  - Instruments for buffer overflows

- Address Sanitizer ([http://code.google.com/p/address-sanitizer/](http://code.google.com/p/address-sanitizer/)) GCC 4.8
  - It finds use-after-free and {heap, stack, global}-buffer overflow bugs in C/C++ programs.

- -fstack-protector strong included in GCC 4.9, previous version as a patch
2. LLVM Defenses

1) Enhanced defense

- GCC compatibility defenses
  -- Analyze (Clang)

LLVM 2.9

LLVM 3.0

LLVM 3.1

LLVM 3.2

GCC compatibility defenses
-- Analyze (Clang)(1)
2. LLVM SECURITY

- Some mitigations enabled by default

- Clang Static Analyzer
  - http://clang-analyzer.llvm.org/available_checks.html
2. DEVELOPERS! DEVELOPERS!

- No excuse, build & ship software with defenses enabled
3. BinSecSweeper
3. Why BinSecSweeper?

- BinSecSweeper is VULNEX binary security verification tool to ensure applications have been built in compliance with Application Assurance best practices
- The goal for BinSecSweeper is a tool:
  - Developers can use to verify that their output binaries are safe after compilation and before releasing their products
  - IT security pros to scan their infrastructure to identify binaries with weak security defenses or vulnerabilities.
- BinSecSweeper is a cross platform tool (works on Windows and Linux) and can scan different file formats: PE and ELF.
3. FEATURES

- 100% open source
- Easy to use
- Cross-platform works on Windows & Linux
- Scans Windows (PE) and Unix (ELF) files for security checks
- Configurable
- Analysis Engine
- Extensible by plugins
- Reporting
3. BinSecSweeper in Action

<table>
<thead>
<tr>
<th>Filename</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Size</td>
<td>128965</td>
</tr>
<tr>
<td>File Type</td>
<td>PE32 executable for MS Windows (GUI) Intel 80386 32-bit</td>
</tr>
<tr>
<td>MD5</td>
<td>550005223c0ebef60c2376ca549782d</td>
</tr>
<tr>
<td>SHA1</td>
<td>c13e3fa7a3a3be5c5b939f5f825b9a5c1638</td>
</tr>
<tr>
<td>SHA256</td>
<td>d2-8f-24e1cf5f593e-3e-1e780c-4f91931bc8292039-bc288677e9c-c3a8</td>
</tr>
<tr>
<td>Entry Point</td>
<td>0x140d</td>
</tr>
<tr>
<td>Type:</td>
<td>EXECUTABLE</td>
</tr>
<tr>
<td>Timestamp</td>
<td>4</td>
</tr>
</tbody>
</table>

**File Info**
- **Risk Level:**
  - File: DIRTY
  - Desc: File: DIRTY

**Summary**
- **Risk Level:**
  - Windows Unsafe API
  - Unsafe API Detected
  - Potential Unsafe API (4):
    - 0x0040f368: __swprintf
    - 0x0040f3b4: strlen
    - 0x0040f3d4: wcscpl
    - 0x0040f3a0: memcpy

- **Risk Level:**
  - Test Plugin
  - Desc: Hello from the test plugin

- **Risk Level:**
  - Adobe Malware Classifier
  - File: DIRTY

- **Risk Level:**
  - Windows ASLR Detection
  - ASLR Detected

- **Risk Level:**
  - Windows Packer Detection
  - No packer found

- **Risk Level:**
  - Windows Stack Cookies (GS) Detection
  - NO Stack Cookie Detected
3. BinSecSweeper in Action II
### 3. Current Windows Checks

<table>
<thead>
<tr>
<th>CHECK</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address space layout randomization (ASLR)</td>
<td>Checks if binary has opted the ASLR. Link with /DYNAMICBASE</td>
</tr>
<tr>
<td>Stack Cookies (GS)</td>
<td>Verifies if binary was compiled with Stack Cookies protection. Compile with /GS</td>
</tr>
<tr>
<td>HotPatch</td>
<td>Checks if binary is prepared for hot patching. Compile with /hotpatch</td>
</tr>
<tr>
<td>Compatible with Data Execution Prevention (NXCOMPAT)</td>
<td>Validates if binary has opted hardware Data Execution Prevention (DEP). Link with /NXCOMPAT</td>
</tr>
<tr>
<td>Structured Exception Handling (SEH)</td>
<td>Checks if binary was linked with SafeSEH. Link with /SAFESEH</td>
</tr>
<tr>
<td>Abobe Malware Classifier</td>
<td>Analyzes binary for malware behavior using machine learning algorithms</td>
</tr>
<tr>
<td>Packer</td>
<td>Checks if binary has been packed</td>
</tr>
<tr>
<td>Insecure API</td>
<td>Check if binary uses banned API</td>
</tr>
</tbody>
</table>
# RSAC

## 3. Current Linux Checks

<table>
<thead>
<tr>
<th>CHECK</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortify Source</td>
<td>Checks if binary was compiled with buffer overflow protection (bounds checking). Compile with <code>–D_FORTIFY_SOURCE=X</code></td>
</tr>
<tr>
<td>Never eXecute (NX)</td>
<td>Verifies if binary was compiled with NX to reduce the area an attacker can use to perform arbitrary code execution.</td>
</tr>
<tr>
<td>Position Independent Code (PIE)</td>
<td>Checks if binary was compiled with PIE to protects against &quot;return-to-text&quot; and generally frustrates memory corruption attacks. Compile with <code>–fPIE -pie</code></td>
</tr>
<tr>
<td>RELocation Read-Only (RELRO)</td>
<td>Validates if binary was compiled with RELRO (partial/full) to harden data sections. Compile with <code>–z,relro,-z,now</code></td>
</tr>
<tr>
<td>Stack Canary</td>
<td>Checks if binary was compiled with stack protector to protect against stack overflows. Compile with <code>–fstack-protector</code></td>
</tr>
</tbody>
</table>
3. Plugin Example: Windows ASLR

class win_aslr_detect(scanpluginclass):
    def __init__(self):
        super(win_aslr_detect, self).__init__()

        self.RegisterPlugin()

    def RegisterPlugin(self):
        d = {
            "name": "Windows ASLR Detection",
            "os": "Windows",
            "arch": "any",
            "code": "native"
        }

        self.SetPluginInfoNew(d)

    def ActivatePlugin(self):

        safe = self.risk_red
        istr = ""

        pe_class = self.GetFileParser()
        pe = pe_class.GetFP()

        if pe == None: return

        if pe.OPTIONAL_HEADER.DllCharacteristics & pe_class.DYNAMICBASE_FLAG:
            istr = "ASLR Detected"
            safe = self.risk_green
        else:
            istr = "NO ASLR Detected"
            safe = self.risk_red

        dl = {
            "name": self.GetPluginInfoData(),
            "safe": safe,
            "category": "info",
            "title": "Windows ASLR Detection",
            "desc": istr,
        }

        self.SetPluginResultsNew(dl)
3. Plugin Example: Linux fortify_source

```python
def ActivatePlugin(self):
    fs = 1
    add_data = []
    fs_funcs = []
    count_fs = 0

    elf_class = self.GetFileParser()
    elf = elf_class.GetPP()

    if elf == None: return

    for section in elf.iter_sections():
        if not isinstance(section, SymbolTableSection):
            continue

        if section['sh_entsize'] == 0:
            continue

        for nsym, symbol in enumerate(section.iter_symbols()):
            ss = bytes2str(symbol.name)
            if not "_stack_chk_fail" in ss and "_chk" in ss and not "LIBC" in ss:
                fs = 0
            fs_funcs.append(ss)
            count_fs++=1

        if fs == 0:
            t = "Fortify Source Functions (%s)" % str(count_fs)
            add_data.append((t, fs_funcs))
            dl = {"name": self.GetPluginInfoData(),
                  "safe": self.risk_green,
                  "category": "info",
                  "title": "Fortify Source Detection",
                  "desc": "Fortify Source Detected",
                  "add_data": add_data}

        else:
            dl = {"name": self.GetPluginInfoData(),
                  "safe": self.risk_red,
                  "category": "info",
                  "title": "Fortify Source Detection",
                  "desc": "NO Fortify Source Detected"
            }

    self.SetPluginResultsNew(dl)
```
3. Reporting
3. BinSecSweeper: what’s next!

- More plugins:
  - Windows, Linux, etc.
  - Mobile
  - Malware
  - Backdoors
  - Compilers
  - Packers
- Metrics panel
- Diff across product / versions
3. BinSecSwepeper: where?

- Download BinSecSweeper software from [www.vulnex.com](http://www.vulnex.com)
- After RSA USA (please give us a couple of weeks to finish up doc 😊)
4. Case Studies
4. Remember Picassa?

Missing: ASLR + DEP
Good: Stack Cookies
But was still exploitable!
4. Are you compiling your app with zlib.dll?
4. Are your 3rd party components improving?

- Python 2.7 -> sqlite3.dll

<table>
<thead>
<tr>
<th>Risk Level:</th>
<th>Windows NXCOMPAT (DEP) Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tile:</td>
<td>Windows NXCOMPAT (DEP) Detection</td>
</tr>
<tr>
<td>Desc:</td>
<td>NO NXCOMPAT (DEP) Detected</td>
</tr>
</tbody>
</table>

- Python 3.3 -> sqlite3.dll

<table>
<thead>
<tr>
<th>Risk Level:</th>
<th>Windows ASLR Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tile:</td>
<td>Windows ASLR Detection</td>
</tr>
<tr>
<td>Desc:</td>
<td>NO ASLR Detected</td>
</tr>
</tbody>
</table>
4. A DLL inside a well-known software

<table>
<thead>
<tr>
<th>Risk Level:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tile:</td>
<td>Windows ASLR Detection</td>
</tr>
<tr>
<td>Desc:</td>
<td>NO ASLR Detected</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Risk Level:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tile:</td>
<td>Windows NXCOMPAT (DEP) Detection</td>
</tr>
<tr>
<td>Desc:</td>
<td>NO NXCOMPAT (DEP) Detected</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Risk Level:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tile:</td>
<td>Windows Unsafe API</td>
</tr>
<tr>
<td>Desc:</td>
<td>Unsafe API Detected</td>
</tr>
</tbody>
</table>

Potential Unsafe API (14):

- 0x00407170: strcpy
- 0x004071bc: strncpy
- 0x00407168: strcat
- 0x00407164: strncat
- 0x00407244: wprintfA
- 0x004071d8: snprintf
- 0x004071c8: _vsnprintf
- 0x004071b8: _snprintf
- 0x004071c8: _vsnprintf
- 0x004071bc: strncpy
- 0x00407164: strncat
- 0x004071cc: sscanff
- 0x0040716c: strlen
- 0x0040715c: memcpy
4. The most common word inside a Microsoft binary?

<table>
<thead>
<tr>
<th>Total N-Grams</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Top 10 N-Grams</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-gram</td>
</tr>
<tr>
<td>on</td>
</tr>
<tr>
<td>ti</td>
</tr>
<tr>
<td>et</td>
</tr>
<tr>
<td>ic</td>
</tr>
<tr>
<td>io</td>
</tr>
<tr>
<td>at</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>Co</td>
</tr>
<tr>
<td>ro</td>
</tr>
</tbody>
</table>
5. Conclusions
5. Verifying Software Security Posture Matters!

- Binaries contain a lot of information!

- The security posture of the software developed by you is important:
  - Security improves Quality
  - Branding (shows you care about security)

- How is the security posture of software vendors you use?
5. Does your Software:

- Has it been compiled with all possible mitigations?
- Use insecure APIs?
- Contain malware?
- Backdoors?
Q&A

- FIN

- Thanks!

- @simonroses