Armor Within: Defending against Vulnerabilities in Third-Party Libraries

Sameed Ali, Prashant Anantharaman, Sean Smith
Dartmouth College, NH, USA
sameed.ali.gr@dartmouth.edu

DARTMOUTH

Outline

Motivation

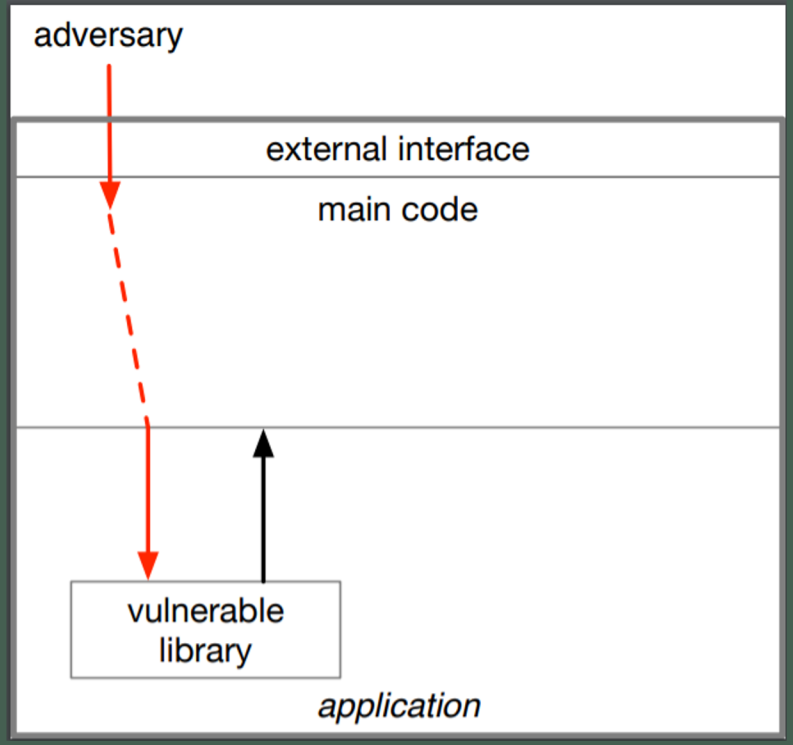
Our Approaches

Evaluation

Conclusions

Crafted input attacks on libraries

- Third-party library input not validated by main application
- App and library in same address space
- Otherwise secure software compromised by a crafted input attack on a third-party library



CVE-2019-7317: Use-after-free in png_image_free of libpng library

Reporter OSS-Fuzz, Eddie Lee

Impact high

Description

A use-after-free vulnerability was discovered in the `png_image_free` function in the libpng library. This could lead to denial of service or a potentially exploitable crash when a malformed image is processed.

References

Bug 1542829

CVE

Repor

Impac
Descr

A use-

This c

proces

Refere

Bug 15

libpng integer overflow

Announced	February 16, 2012
Impact	**Critical**
Products	Firefox, Firefox ESR, SeaMonkey, Thunderbird, Thunderbird ESR
Fixed in	Firefox 10.0.2 Firefox 3.6.27 Firefox ESR 10.0.2 SeaMonkey 2.7.2 Thunderbird 10.0.2 Thunderbird 3.1.19 Thunderbird ESR 10.0.2

Description

An integer overflow in the libpng library can lead to a heap-buffer overflow when decompressing certain PNG images. This leads to a crash, which may be potentially exploitable.

ary

library.

image is

libpng integer overflow

Security Update Guide > Details

CVE-2020-0938 | Adobe Font Manager Library Remote Code Execution Vulnerability

Security Vulnerability

Published: 04/14/2020 | Last Updated : 04/14/2020

MITRE CVE-2020-0938

A remote code execution vulnerability exists in Microsoft Windows when the Windows Adobe Type Manager Library improperly handles a specially-crafted multi-master font - Adobe Type 1 PostScript format.

For all systems except Windows 10, an attacker who successfully exploited the vulnerability could execute code remotely. For systems running Windows 10, an attacker who successfully exploited the vulnerability could execute code in an AppContainer sandbox context with limited privileges and capabilities. An attacker could then install programs; view, change, or delete data; or create new accounts with full user rights.

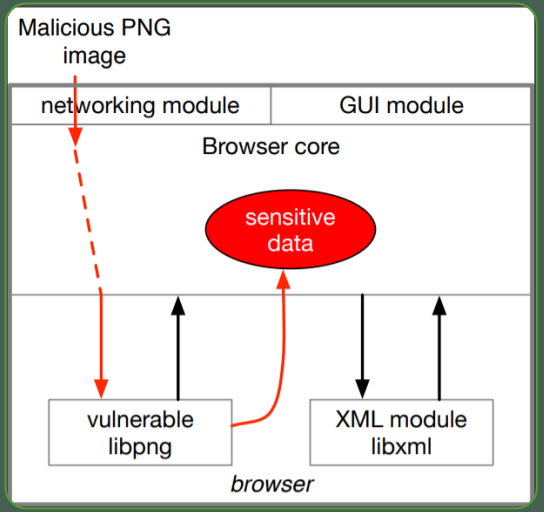
There are multiple ways an attacker could exploit the vulnerability, such as convincing a user to open a specially crafted document or viewing it in the Windows Preview pane.

The update addresses the vulnerability by correcting how the Windows Adobe Type Manager Library handles Type1 fonts.

CVE-2020-9391	An issue was discovered in the Linux kernel 5.4 and 5.5 through 5.5.6 on the AArch64 architecture. It ignores the top byte in the address passed to the brk system call, potentially moving the memory break downwards when the application expects it to move upwards, aka CID-dcde237319e6. This has been observed to cause heap corruption with the GNU C Library malloc implementation.
CVE-2020-9290	An Unsafe Search Path vulnerability in FortiClient for Windows online installer 6.2.3 and below may allow a local attacker with control over the directory in which FortiClientOnlineInstaller.exe and FortiClientVPNOnlineInstaller.exe resides to execute arbitrary code on the system via uploading malicious Filter Library DLL files in that directory.
CVE-2020-9287	An Unsafe Search Path vulnerability in FortiClient EMS online installer 6.2.1 and below may allow a local attacker with control over the directory in which FortiClientEMSOnlineInstaller.exe resides to execute arbitrary code on the system via uploading malicious Filter Library DLL files in that directory.
CVE-2020-8945	The proglottis Go wrapper before 0.1.1 for the GPGME library has a use-after-free, as demonstrated by use for container image pulls by Docker or CRI-O. This leads to a crash or potential code execution during GPG signature verification.
CVE-2020-8910	A URL parsing issue in goog.uri of the Google Closure Library versions up to and including v20200224 allows an attacker to send malicious URLs to be parsed by the library and return the wrong authority. Mitigation: update your library to version v20200315.
CVE-2020-8899	There is a buffer overwrite vulnerability in the Quram qmg library of Samsung's Android OS versions O(8.x), P(9.0) and Q(10.0). An unauthenticated, unauthorized attacker sending a specially crafted MMS to a vulnerable phone can trigger a heap-based buffer overflow in the Quram image codec leading to an arbitrary remote code execution (RCE) without any user interaction. The Samsung ID is SVE-2020-16747.
CVE-2020-8155	An outdated 3rd party library in the Files PDF viewer for Nextcloud Server 18.0.2 caused a Cross-site scripting vulnerability when opening a malicious PDF.
CVE-2020-8096	Untrusted Search Path vulnerability in Bitdefender High-Level Antimalware SDK for Windows allows an attacker to load third party code from a DLL library in the search path. This issue affects: Bitdefender High-Level Antimalware SDK for Windows versions prior to 3.0.1.204 .
CVE-2020-8093	A vulnerability in the AntivirusforMac binary as used in Bitdefender Antivirus for Mac allows an attacker to inject a library using DYLD environment variable to cause third-party code execution
CVE-2020-7618	sds through 3.2.0 is vulnerable to Prototype Pollution.The library could be tricked into adding or modifying properties of the 'Object.prototype' by abusing the 'set' function located in 'js/set.js'.
CVE-2020-7617	ini-parser through 0.0.2 is vulnerable to Prototype Pollution.The library could be tricked into adding or modifying properties of Object.prototype using a '__proto__' payload.
CVE-2020-7490	A CWE-426: Untrusted Search Path vulnerability exists in Vijeo Designer Basic (V1.1 HotFix 15 and prior) and Vijeo Designer (V6.9 SP9 and prior), which could cause arbitrary code execution on the system running Vijeo Basic when a malicious DLL library is loaded by the Product.

An example: CVE-2004-0597

- The adversary tricks the browser into sending a malicious PNG file into the libPNG library.
- The exploited software module can then access sensitive information in other parts of the address space.



Outline




Motivation

Our Approaches

Evaluation

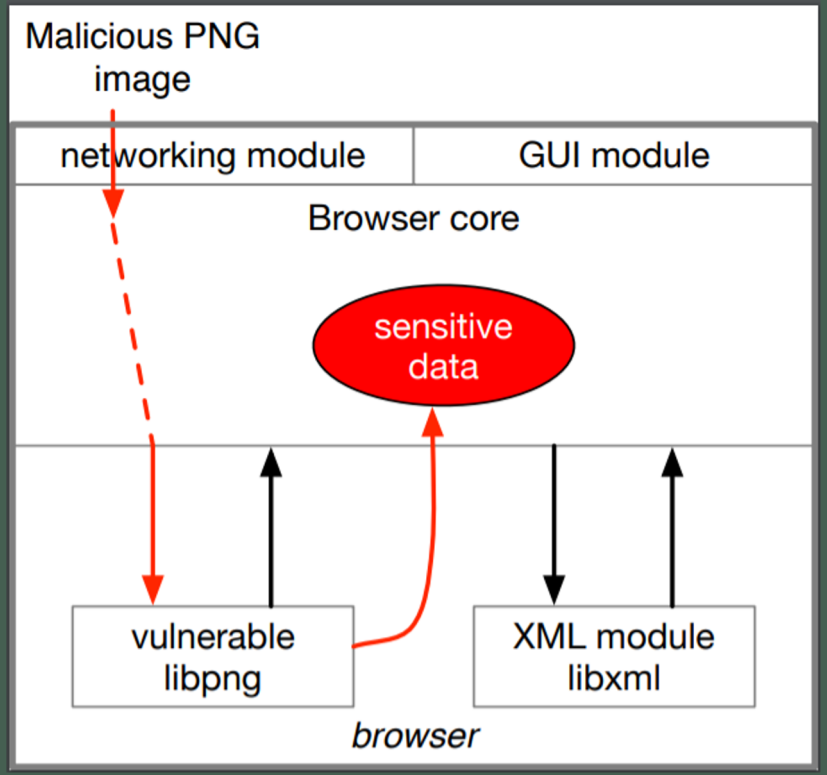
Conclusions

Proposed Solution

1. Compartmentalize application address space (via ELFbac)
2. LangSec validation applied to input of third-party software modules
3. Inject LangSec validation parser/filter in the software via
    a. Object rewriting
    b. Binary rewriting
4. Ensure CFI so validation not bypassed (via ELFbac policy)

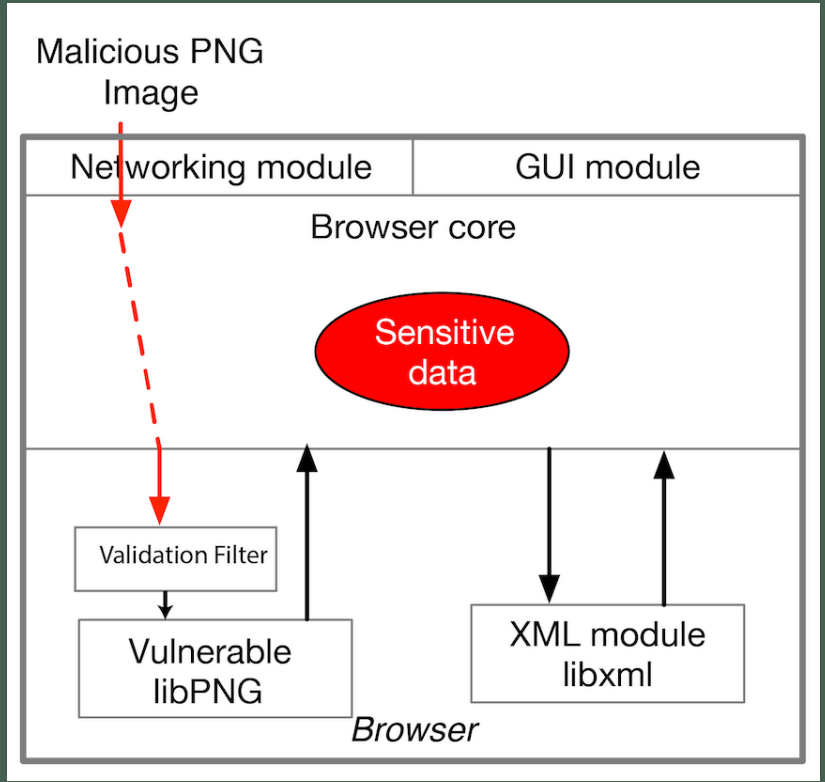
CVE-2004-0597: LibPNG

- The adversary tricks the browser into sending a malicious PNG file into the libpng library.
- The exploited software module can then access sensitive information in other parts of the address space.



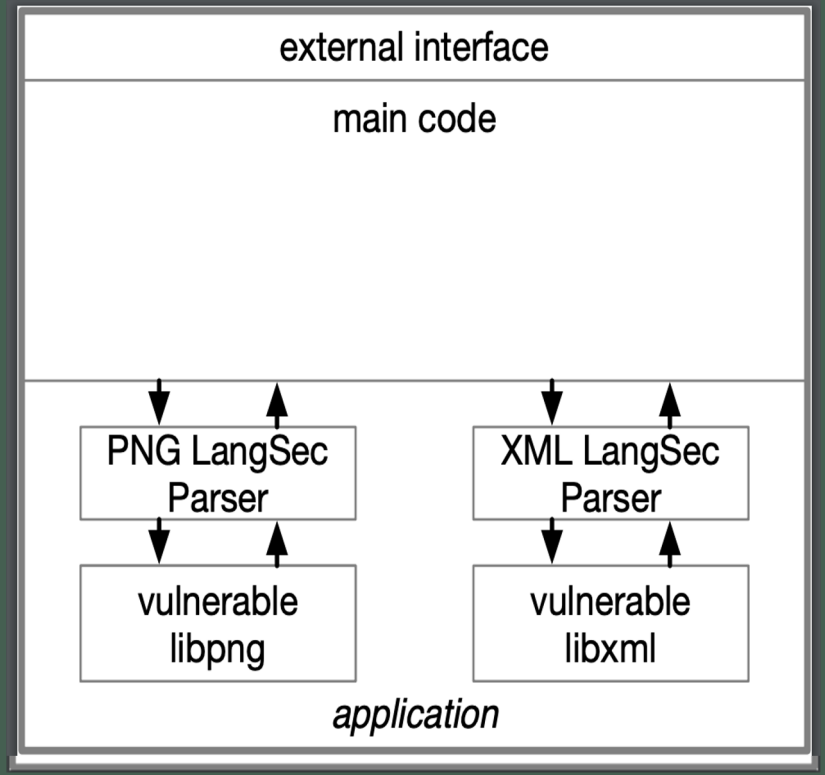
CVE-2004-0597: LibPNG

- The adversary tricks the browser into sending a malicious PNG file into the libpng library.
- The exploited software module can then access sensitive information in other parts of the address space.



CVE-2004-0597: LibPNG

- The adversary tricks the browser into sending a malicious PNG file into the libpng library.
- The exploited software module can then access sensitive information in other parts of the address space.



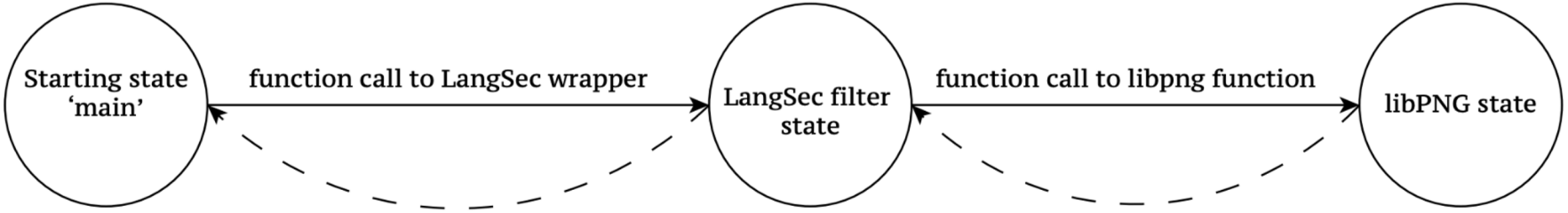
```
66 HParser* get_PNG_parser() {
67     const HParser* MAGIC = h_token("\x89\x50\x4E\x47\x0D\x0A\x1A\x0A", 8);
68     const HParser* IHDR_len = h_int_range(h_uint32(), 13, 13);
69     const HParser* IHDR_type = h_token("IHDR", 4);
70     const HParser* IHDR_chunk = h_sequence(h_uint32(), h_uint32(), h_repeat_n(h_int8(), 5), NULL)
71     const HParser* IHDR_crc = h_uint32();
72
73     const HParser* byte = h_uint8();
74
75     // tRNS chunk
76     const HParser* tRNS_chunk_type = h_token("tRNS", 4);
77     const HParser* tRNS_chunk_len = h_left(h_int_range(h_uint32(), 0, 255), tRNS_chunk_type);
78     const HParser* tRNS_chunk = h_length_value(tRNS_chunk_len, byte);
79
80     // any other chunk
81     const HParser* chunk_type = h_not(h_token("tRNS", 4));
82     const HParser* chunk_len = h_left(h_uint32(), chunk_type);
83     const HParser* chunk = h_length_value(chunk_len, byte);
84
85     const HParser* chunks = h_many1(chunk);
86
87     HParser* PNG_parser = h_sequence(MAGIC, IHDR_len, IHDR_type, IHDR_chunk, IHDR_crc, chunks,
            NULL);
88     return PNG_parser;
89 }
```

What does a LS parser/filter look like?

A simple PNG LS parser/filter using Hammer

```
66 HParser* get_PNG_parser() {
67     const HParser* MAGIC = h_token("\x89\x50\x4E\x47\x0D\x0A\x1A\x0A", 8);
68     const HParser* IHDR_len = h_int_range(h_uint32(), 13, 13);
69     const HParser* IHDR_type = h_token("IHDR", 4);
70     const HParser* IHDR_chunk = h_sequence(h_uint32(), h_uint32(), h_repeat_n(h_int8(), 5), NULL);
71     const HParser* IHDR_crc = h_uint32();
72
73     const HParser* byte = h_uint8();
74
75     // tRNS chunk
76     const HParser* tRNS_chunk_type = h_token("tRNS", 4);
77     const HParser* tRNS_chunk_len = h_left(h_int_range(h_uint32(), 0, 255), tRNS_chunk_type);
78     const HParser* tRNS_chunk = h_length_value(tRNS_chunk_len, byte);
79
80     // any other chunk
81     const HParser* chunk_type = h_not(h_token("tRNS", 4));
82     const HParser* chunk_len = h_left(h_uint32(), chunk_type);
83     const HParser* chunk = h_length_value(chunk_len, byte);
84
85     const HParser* chunks = h_many1(chunk);
86
87     HParser* PNG_parser = h_sequence(MAGIC, IHDR_len, IHDR_type, IHDR_chunk, IHDR_crc, chunks,
            NULL);
88     return PNG_parser;
89 }
```

Ensuring control flow integrity



Policy State Diagram

Filter Injection via object rewriting

- Assumption: Constituent software modules compiled objects available
- Rewrite the Symbol table of the target object
- Library symbols point to with LangSec filter functions
- Link the objects together to generate the binary
- Inject ELFbac policy

Filter Injection via LLVM

- Lift binary to LLVM IR code
- Insert LangSec validation filter via a custom LLVM IR pass
- Compile LLVM to generate required binary
- Inject ELFbac policy

Outline

Motivation

Our Approaches

Evaluation

Conclusions

Evaluation

To evaluate our system, we answer the following questions:

- Is *Armor Within* effective against known vulnerabilities?
- How much overhead do our LangSec filters add to existing binaries?
- Can *Armor Within* effectively inject parsers in existing binaries?

Evaluating against known vulnerabilities

Armor Within was able to successfully detect and mitigate the following vulnerabilities:

- CVE-2016-1838: Denial-of-service heap-based buffer over-read vulnerability in LIBXML
- CVE-2004-0597: Stack-Overflow remote code execution vulnerability in LibPNG
- CVE-2010-1205: Buffer overflow in LibPNG

We ran these experiments on a Desktop computer equipped with a Xeon E3-1245 processor and 8 Gigabytes of RAM. The computer ran Ubuntu Linux version 12.04 with the ELFbac Linux kernel patch.

Overheads added by our LangSec filters

PNG file size	No instrumentation	With LangSec filter	LangSec filter with ELFbac policy
100KB			
Page faults	273	660	1230
Instructions	28,974,520	35,669,663	2,050,000,720
Time	0.028951596	0.061482514	0.540514533
200KB			
Page faults	326	713	1,334
Instructions	45,926,310	52,487,839	2,414,208,538
Time	0.063752567	0.076081896	0.66811976
500KB			
Page faults	525	912	1,734
Instructions	111,338,010	119,492,290	4,070,225,945
Time	0.117933531	0.127791843	1.201331268

Outline

Motivation

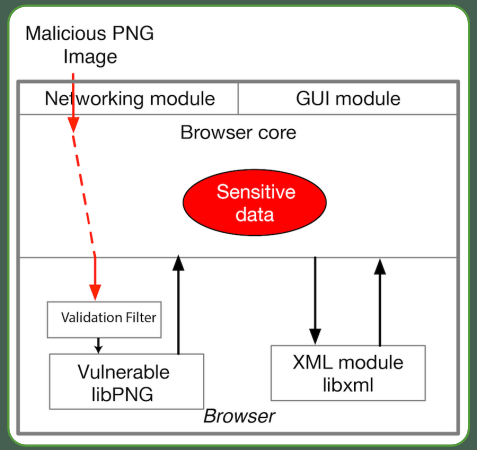
Our Approaches

Evaluation

Conclusions

Conclusions

- *Armor Within* comprises two techniques to inject LangSec parsers in binaries:
  - Object rewriting
  - Binary rewriting
- First technique is suited to dynamically linked libraries, whereas second technique works for statically linked libraries.
- Our tools were effective and added minimal overhead in terms of memory and CPU time to existing binaries.



Future Work

- *Armor Within,* works with Hammer parsers. We are working to make the tool more generic and can accept any parser combinator toolkit.

- For control-flow integrity, we used ELFbac in this paper. We are working to make our tools to be agnostic of the control-flow integrity techniques.

- We are working on a parser generator that converts BNF syntax to parser-combinator syntax.

Thank you!

Questions?

Sameed Ali sameed.ali.gr@dartmouth.edu

Prashant pa@cs.dartmouth.edu

Sean sws@cs.dartmouth.edu

Code available at:
https://bitbucket.org/sameed_ali/app-armor-poc/

Acknowledgements

This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-16-C-0179. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of United States Government or any agency thereof.