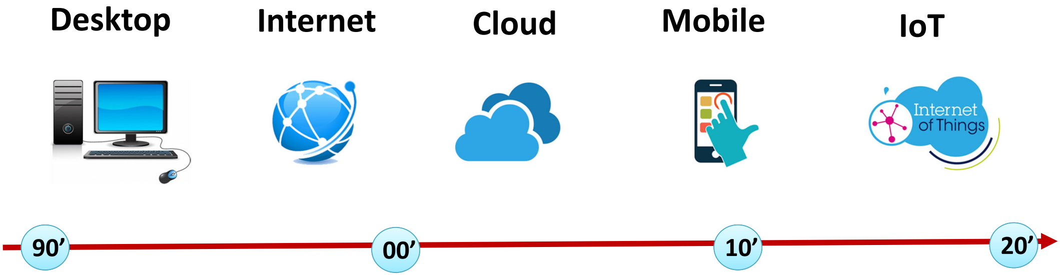
Software Security: Past, Present, and Future

Zhiqiang Lin

zlin@cse.ohio-state.edu

08/19/2021

Software is Everywhere



Desktop

Internet

Cloud

Mobile

IoT

90’

00’

10’

20’

Software is Everywhere

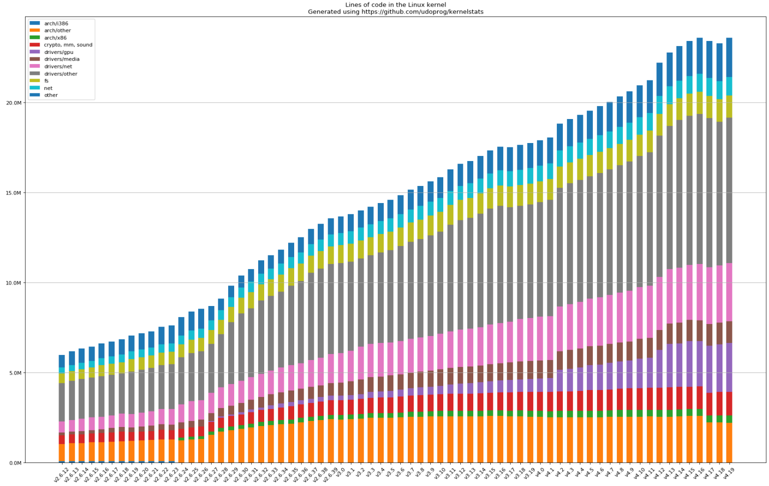
Source: https://iotool.io/news/industry-4-0/what-software-everywhere-means-for-iot-product-development

It has Become Inceasingly Sophiscated

It has Become Inceasingly Sophiscated

It has Become Inceasingly Sophiscated

It has Become Inceasingly Sophiscated

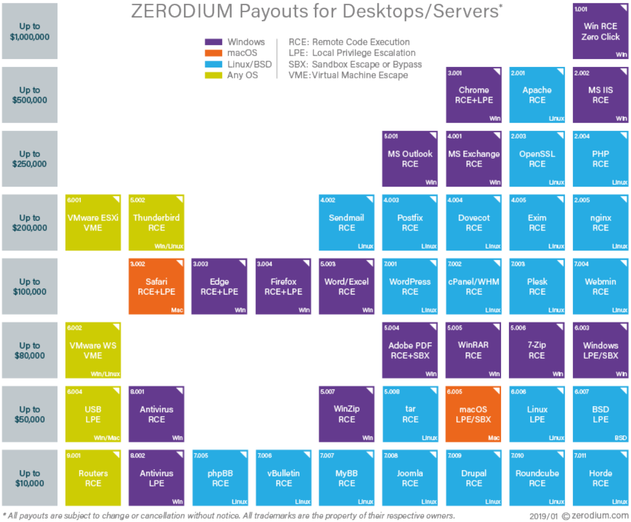


Source: https://www.reddit.com/r/linux/comments/9uxwli/lines_of_code_in_the_linux_kernel/

Software has Vulnerabilities

1. Buffer overflow (overrun), or over-reads
   - Stack overflow
   - Heap overflow
   - Global data (.got, .data, .bss) overflow
2. Format string (arbitrary write)
3. Integer overflow
4. User-after-free
5. Double free
6. ...

The high-end vulnerabilities: millions of dollars



ZERODIUM Payouts for Desktops/Servers*

- 🟪 Windows
- 🟧 macOS
- 🟦 Linux/BSD
- 🟨 Any OS

RCE: Remote Code Execution
LPE: Local Privilege Escalation
SBX: Sandbox Escape or Bypass
VME: Virtual Machine Escape

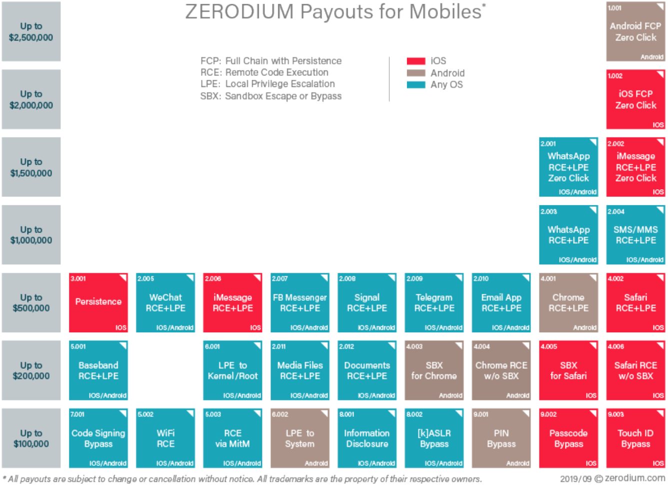
Up to $1,000,000									1.001 Win RCE Zero Click Win
Up to $500,000							3.001 Chrome RCE+LPE Win	2.001 Apache RCE Linux	2.002 MS IIS RCE Win
Up to $250,000						5.001 MS Outlook RCE Win	4.001 MS Exchange RCE Win	2.003 OpenSSL RCE Linux	2.004 PHP RCE Linux
Up to $200,000	6.001 VMware ESXi VME	5.002 Thunderbird RCE Win/Linux			4.002 Sendmail RCE Linux	4.003 Postfix RCE Linux	4.004 Dovecot RCE Linux	4.005 Exim RCE Linux	2.005 nginx RCE Linux
Up to $100,000		3.002 Safari RCE+LPE Mac	3.003 Edge RCE+LPE Win	3.004 Firefox RCE+LPE Win	5.003 Word/Excel RCE Win	7.001 WordPress RCE Linux	7.002 cPanel/WHM RCE Linux	7.003 Plesk RCE Linux	7.004 Webmin RCE Linux
Up to $80,000	6.002 VMware WS VME Win/Linux					5.004 Adobe PDF RCE+SBX Win	5.005 WinRAR RCE Win	5.006 7-Zip RCE Win	6.003 Windows LPE/SBX Win
Up to $50,000	6.004 USB LPE Win/Mac	8.001 Antivirus RCE Win			5.007 WinZip RCE Win	5.008 tar RCE Linux	6.005 macOS LPE/SBX Mac	6.006 Linux LPE Linux	6.007 BSD LPE BSD
Up to $10,000	9.001 Routers RCE	8.002 Antivirus LPE Win	7.005 phpBB RCE Linux	7.006 vBulletin RCE Linux	7.007 MyBB RCE Linux	7.008 Joomla RCE Linux	7.009 Drupal RCE Linux	7.010 Roundcube RCE Linux	7.011 Horde RCE Linux

* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/01 © zerodium.com

Source: https://zerodium.com/program.html

The high-end vulnerabilities: millions of dollars



ZERODIUM Payouts for Mobiles*

FCP: Full Chain with Persistence
RCE: Remote Code Execution
LPE: Local Privilege Escalation
SBX: Sandbox Escape or Bypass

🟥 iOS
🟫 Android
🟦 Any OS

Up to $2,500,000									1.001 Android FCP Zero Click Android
Up to $2,000,000									1.002 iOS FCP Zero Click iOS
Up to $1,500,000								2.001 WhatsApp RCE+LPE Zero Click iOS/Android	2.002 iMessage RCE+LPE Zero Click iOS
Up to $1,000,000								2.003 WhatsApp RCE+LPE iOS/Android	2.004 SMS/MMS RCE+LPE iOS/Android
Up to $500,000	3.001 Persistence iOS	2.005 WeChat RCE+LPE iOS/Android	2.006 iMessage RCE+LPE iOS	2.007 FB Messenger RCE+LPE iOS/Android	2.008 Signal RCE+LPE iOS/Android	2.009 Telegram RCE+LPE iOS/Android	2.010 Email App RCE+LPE iOS/Android	4.001 Chrome RCE+LPE Android	4.002 Safari RCE+LPE iOS
Up to $200,000	5.001 Baseband RCE+LPE iOS/Android		6.001 LPE to Kernel/Root iOS/Android	2.011 Media Files RCE+LPE iOS/Android	2.012 Documents RCE+LPE iOS/Android	4.003 SBX for Chrome Android	4.004 Chrome RCE w/o SBX Android	4.005 SBX for Safari iOS	4.006 Safari RCE w/o SBX iOS
Up to $100,000	7.001 Code Signing Bypass iOS/Android	5.002 WiFi RCE iOS/Android	5.003 RCE via MitM iOS/Android	6.002 LPE to System Android	8.001 Information Disclosure iOS/Android	8.002 [k]ASLR Bypass iOS/Android	9.001 PIN Bypass Android	9.002 Passcode Bypass iOS	9.003 Touch ID Bypass iOS

*All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/09 © zerodium.com

Source: https://zerodium.com/program.html

The Offense and Defense w/ Memory Corruptions

Offense

1. Code injection
   - NULL-free shellcode
   - NOP sled
   - JMP %ESP
2. Code reuse
   - Return-into-libc
   - ROP
   - JIT-ROP
   - BROP

The Offense and Defense w/ Memory Corruptions

Offense

1. Code injection
   - NULL-free shellcode
   - NOP sled
   - JMP %ESP
2. Code reuse
   - Return-into-libc
   - ROP
   - JIT-ROP
   - BROP

Defense

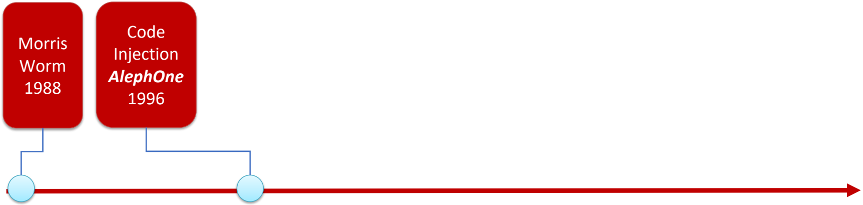
1. Stack canary
2. Non-executable memory
3. ASLR
   - Partial ASLR
   - Full ASLR
4. Control flow integrity
5. Fuzzing
6. Code hardening

The Arm Race Between Offense and Defense w/ Memory Corruptions

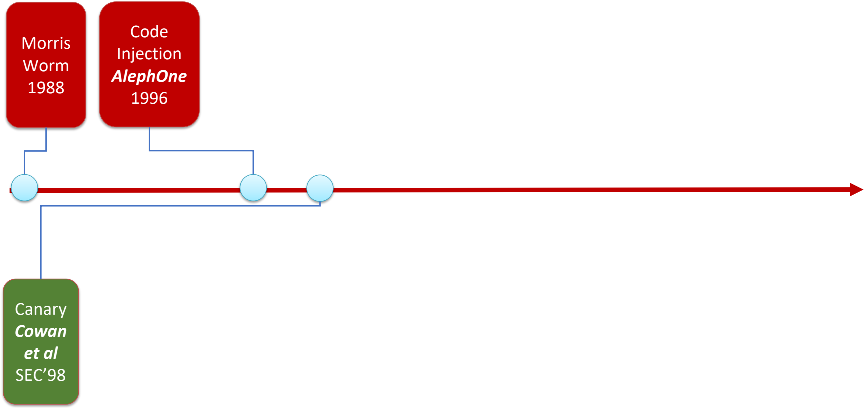
The Arm Race Between Offense and Defense w/ Memory Corruptions

Morris Worm 1988

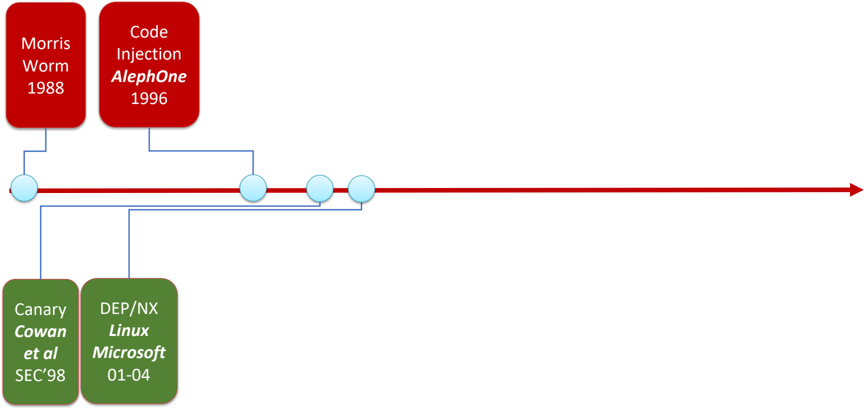
The Arm Race Between Offense and Defense w/ Memory Corruptions



The Arm Race Between Offense and Defense w/ Memory Corruptions



The Arm Race Between Offense and Defense w/ Memory Corruptions



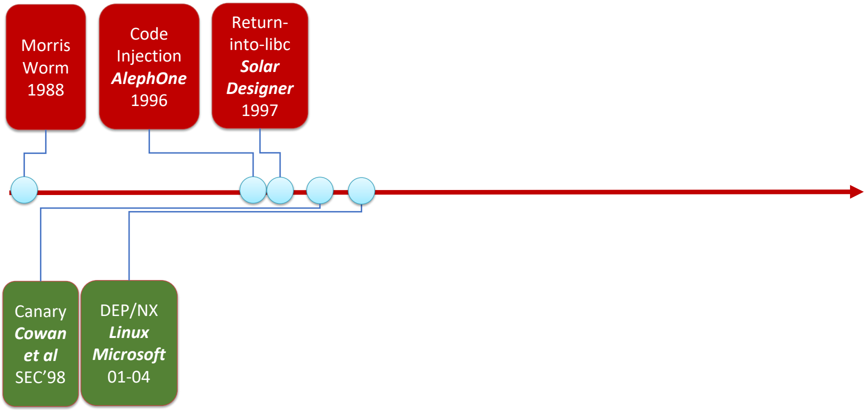
Morris Worm 1988

Code Injection ***AlephOne*** 1996

Canary ***Cowan et al*** SEC'98

DEP/NX ***Linux Microsoft*** 01-04

The Arm Race Between Offense and Defense w/ Memory Corruptions



Morris Worm 1988

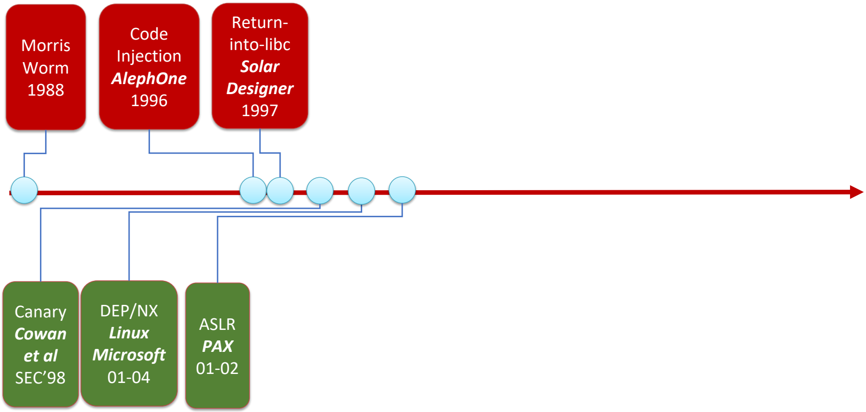
Code Injection ***AlephOne*** 1996

Return-into-libc ***Solar Designer*** 1997

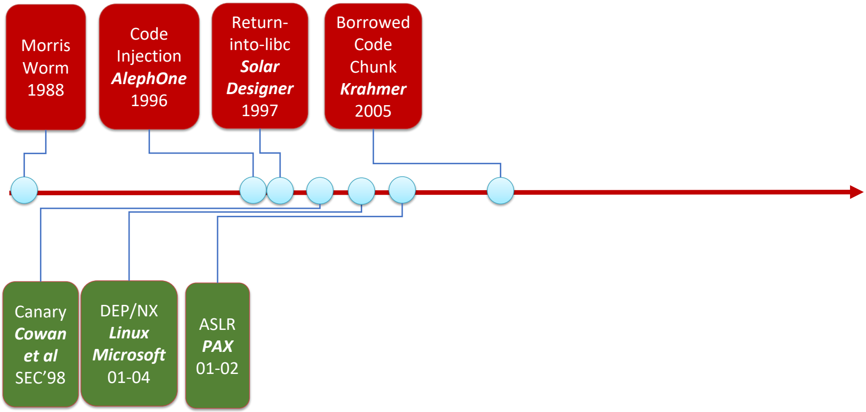
Canary ***Cowan et al*** SEC'98

DEP/NX ***Linux Microsoft*** 01-04

The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988
Code Injection ***AlephOne*** 1996
Return-into-libc ***Solar Designer*** 1997
Canary ***Cowan et al*** SEC'98
DEP/NX ***Linux Microsoft*** 01-04
ASLR ***PAX*** 01-02

The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988

Code Injection ***AlephOne*** 1996

Return-into-libc ***Solar Designer*** 1997

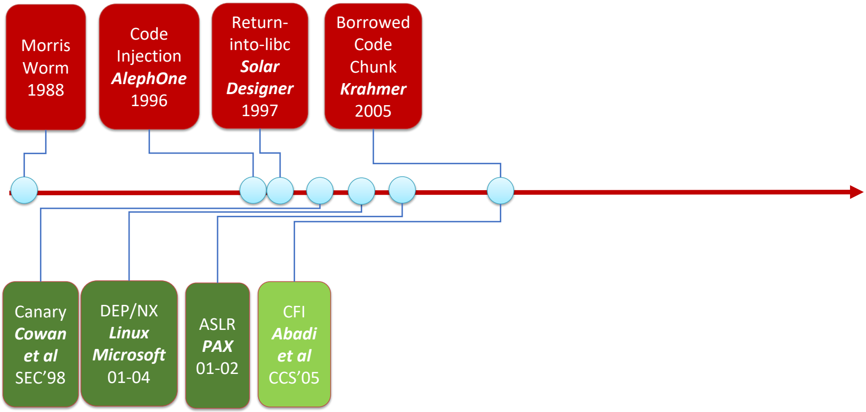
Borrowed Code Chunk ***Krahmer*** 2005

Canary ***Cowan et al*** SEC'98

DEP/NX ***Linux Microsoft*** 01-04

ASLR ***PAX*** 01-02

The Arm Race Between Offense and Defense w/ Memory Corruptions



Morris Worm 1988

Code Injection ***AlephOne*** 1996

Return-into-libc ***Solar Designer*** 1997

Borrowed Code Chunk ***Krahmer*** 2005

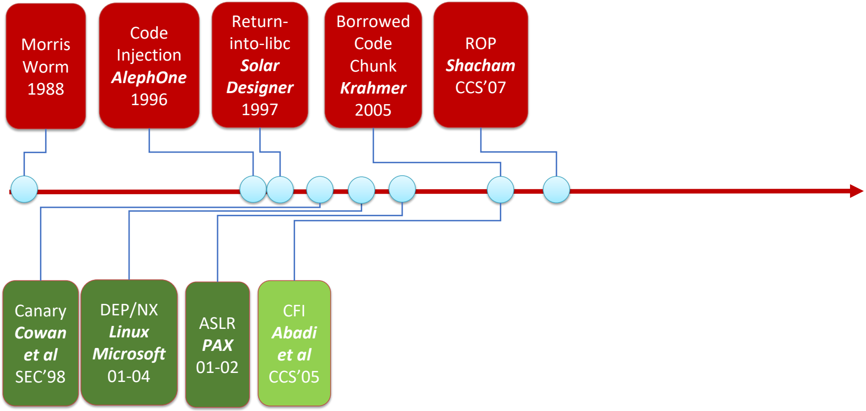
Canary ***Cowan et al*** SEC’98

DEP/NX ***Linux Microsoft*** 01-04

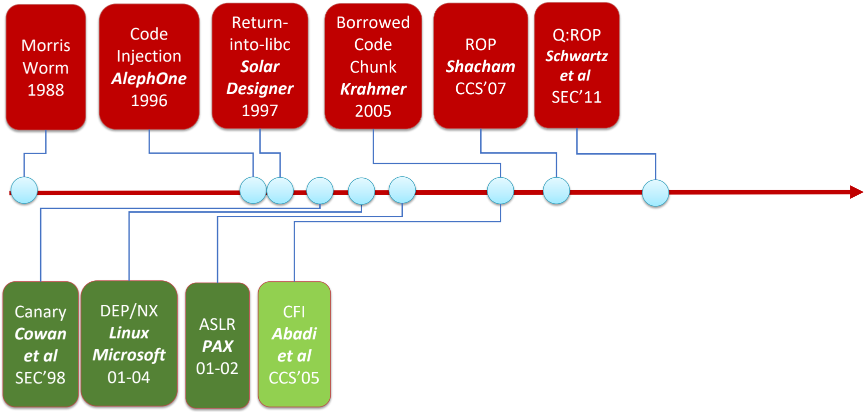
ASLR ***PAX*** 01-02

CFI ***Abadi et al*** CCS’05

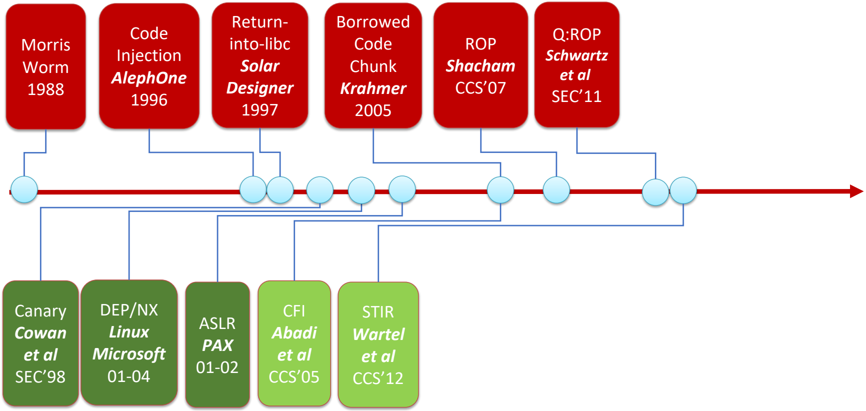
The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc *Solar Designer* 1997
Borrowed Code Chunk *Krahmer* 2005
ROP *Shacham* CCS'07
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04
ASLR *PAX* 01-02
CFI *Abadi et al* CCS'05

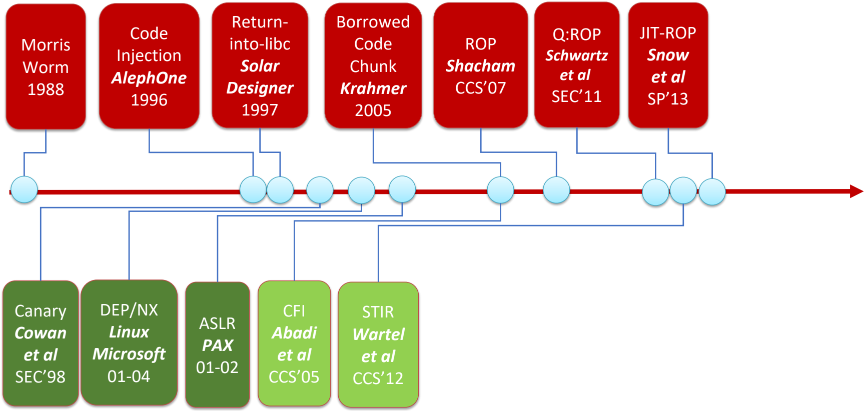
The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc *Solar Designer* 1997
Borrowed Code Chunk *Krahmer* 2005
ROP *Shacham* CCS'07
Q:ROP *Schwartz et al* SEC'11
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04
ASLR *PAX* 01-02
CFI *Abadi et al* CCS'05

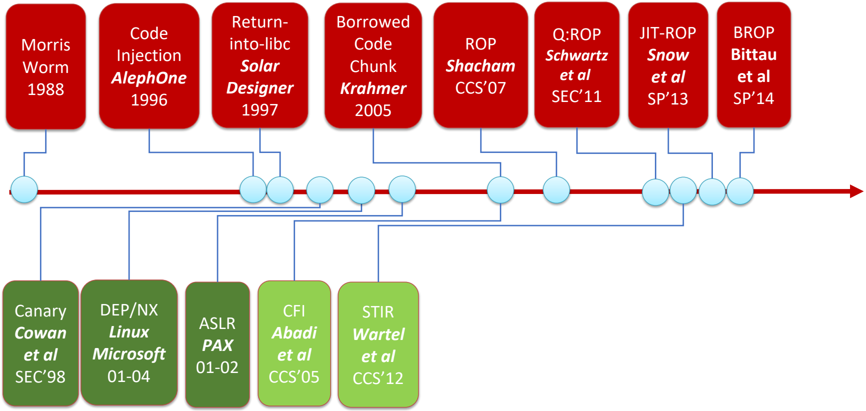
The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc *Solar Designer* 1997
Borrowed Code Chunk *Krahmer* 2005
ROP *Shacham* CCS'07
Q:ROP *Schwartz et al* SEC'11
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04
ASLR *PAX* 01-02
CFI *Abadi et al* CCS'05
STIR *Wartel et al* CCS'12

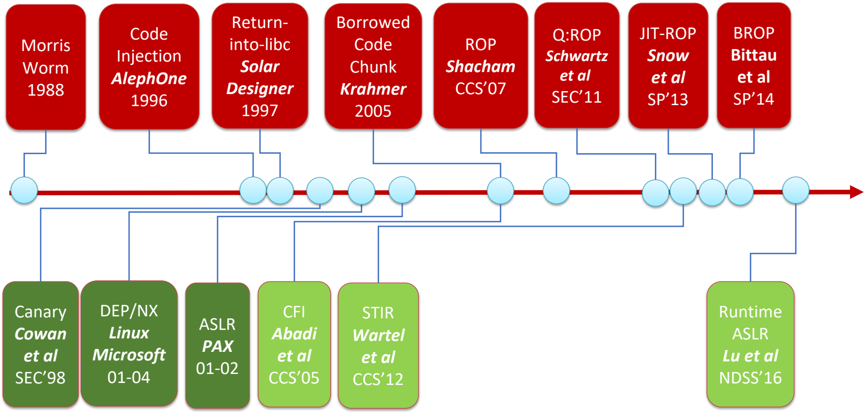
The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988
Code Injection ***AlephOne*** 1996
Return-into-libc ***Solar Designer*** 1997
Borrowed Code Chunk ***Krahmer*** 2005
ROP ***Shacham*** CCS'07
Q:ROP ***Schwartz et al*** SEC'11
JIT-ROP ***Snow et al*** SP'13
Canary ***Cowan et al*** SEC'98
DEP/NX ***Linux Microsoft*** 01-04
ASLR ***PAX*** 01-02
CFI ***Abadi et al*** CCS'05
STIR ***Wartel et al*** CCS'12

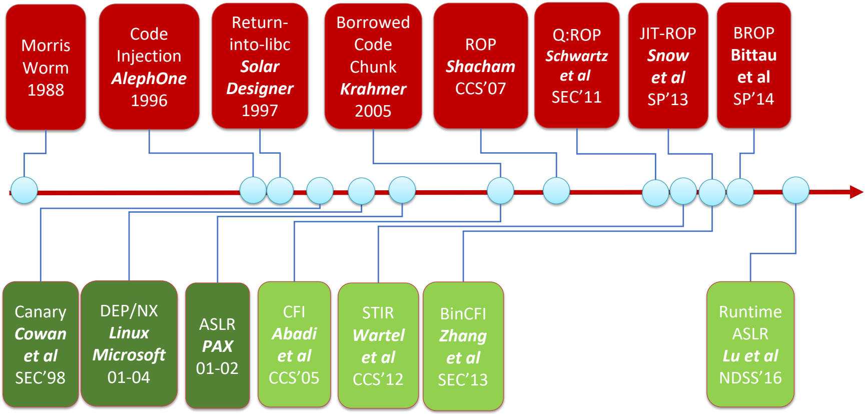
The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc *Solar Designer* 1997
Borrowed Code Chunk *Krahmer* 2005
ROP *Shacham* CCS'07
Q:ROP *Schwartz et al* SEC'11
JIT-ROP *Snow et al* SP'13
BROP **Bittau et al** SP'14
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04
ASLR *PAX* 01-02
CFI *Abadi et al* CCS'05
STIR *Wartel et al* CCS'12

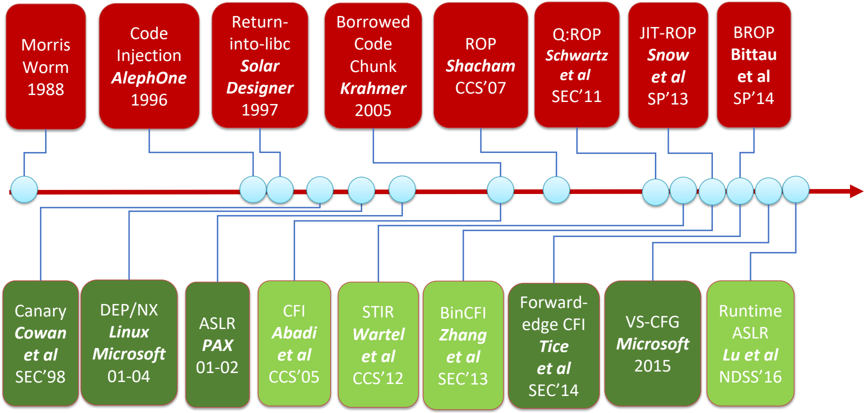
The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988
Code Injection ***AlephOne*** 1996
Return-into-libc ***Solar Designer*** 1997
Borrowed Code Chunk ***Krahmer*** 2005
ROP ***Shacham*** CCS'07
Q:ROP ***Schwartz et al*** SEC'11
JIT-ROP ***Snow et al*** SP'13
BROP **Bittau et al** SP'14
Canary ***Cowan et al*** SEC'98
DEP/NX ***Linux Microsoft*** 01-04
ASLR ***PAX*** 01-02
CFI ***Abadi et al*** CCS'05
STIR ***Wartel et al*** CCS'12
Runtime ASLR ***Lu et al*** NDSS'16

The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc *Solar Designer* 1997
Borrowed Code Chunk *Krahmer* 2005
ROP *Shacham* CCS'07
Q:ROP *Schwartz et al* SEC'11
JIT-ROP *Snow et al* SP'13
BROP **Bittau et al** SP'14
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04
ASLR *PAX* 01-02
CFI *Abadi et al* CCS'05
STIR *Wartel et al* CCS'12
BinCFI *Zhang et al* SEC'13
Runtime ASLR *Lu et al* NDSS'16

The Arm Race Between Offense and Defense w/ Memory Corruptions



Morris Worm 1988

Code Injection ***AlephOne*** 1996

Return-into-libc ***Solar Designer*** 1997

Borrowed Code Chunk ***Krahmer*** 2005

ROP ***Shacham*** CCS'07

Q:ROP ***Schwartz et al*** SEC'11

JIT-ROP ***Snow et al*** SP'13

BROP **Bittau et al** SP'14

Canary ***Cowan et al*** SEC'98

DEP/NX ***Linux Microsoft*** 01-04

ASLR ***PAX*** 01-02

CFI ***Abadi et al*** CCS'05

STIR ***Wartel et al*** CCS'12

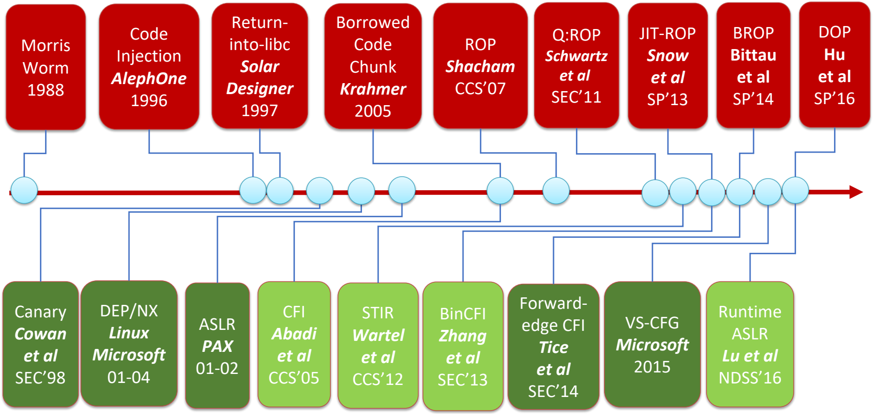
BinCFI ***Zhang et al*** SEC'13

Forward-edge CFI ***Tice et al*** SEC'14

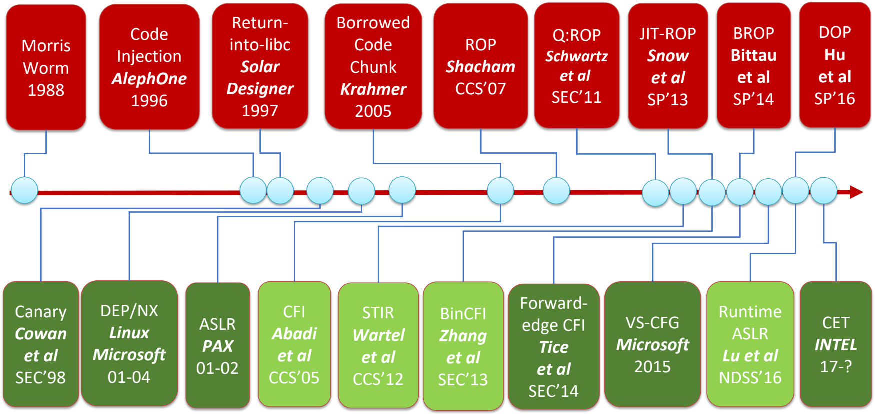
VS-CFG ***Microsoft*** 2015

Runtime ASLR ***Lu et al*** NDSS'16

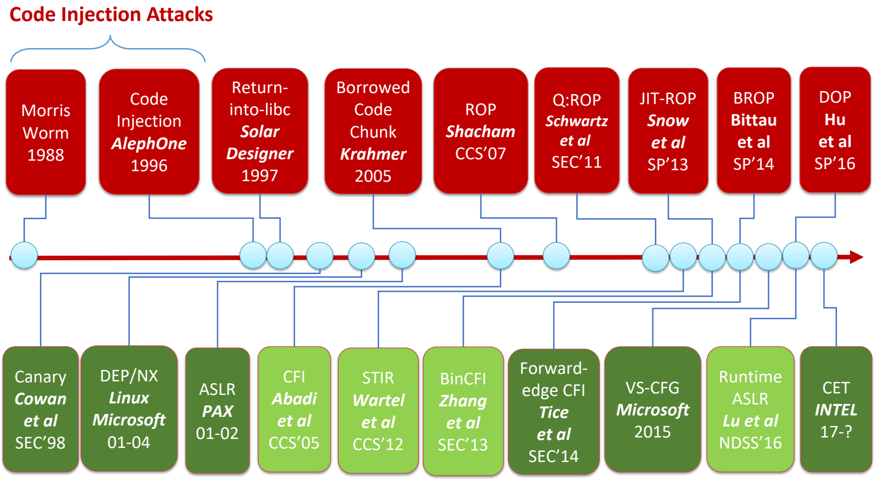
The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc ***Solar Designer*** 1997
Borrowed Code Chunk ***Krahmer*** 2005
ROP ***Shacham*** CCS'07
Q:ROP ***Schwartz et al*** SEC'11
JIT-ROP ***Snow et al*** SP'13
BROP **Bittau et al** SP'14
DOP **Hu et al** SP'16
Canary ***Cowan et al*** SEC'98
DEP/NX ***Linux Microsoft*** 01-04
ASLR ***PAX*** 01-02
CFI ***Abadi et al*** CCS'05
STIR ***Wartel et al*** CCS'12
BinCFI ***Zhang et al*** SEC'13
Forward-edge CFI ***Tice et al*** SEC'14
VS-CFG ***Microsoft*** 2015
Runtime ASLR ***Lu et al*** NDSS'16

The Arm Race Between Offense and Defense w/ Memory Corruptions


Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc *Solar Designer* 1997
Borrowed Code Chunk *Krahmer* 2005
ROP *Shacham* CCS'07
Q:ROP *Schwartz et al* SEC'11
JIT-ROP *Snow et al* SP'13
BROP **Bittau et al** SP'14
DOP **Hu et al** SP'16
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04
ASLR *PAX* 01-02
CFI *Abadi et al* CCS'05
STIR *Wartel et al* CCS'12
BinCFI *Zhang et al* SEC'13
Forward-edge CFI *Tice et al* SEC'14
VS-CFG *Microsoft* 2015
Runtime ASLR *Lu et al* NDSS'16
CET *INTEL* 17-?

The Arm Race Between Offense and Defense w/ Memory Corruptions



Code Injection Attacks

Morris Worm 1988

Code Injection ***AlephOne*** 1996

Return-into-libc ***Solar Designer*** 1997

Borrowed Code Chunk ***Krahmer*** 2005

ROP ***Shacham*** CCS'07

Q:ROP ***Schwartz et al*** SEC'11

JIT-ROP ***Snow et al*** SP'13

BROP **Bittau et al** SP'14

DOP **Hu et al** SP'16

Canary ***Cowan et al*** SEC'98

DEP/NX ***Linux Microsoft*** 01-04

ASLR ***PAX*** 01-02

CFI ***Abadi et al*** CCS'05

STIR ***Wartel et al*** CCS'12

BinCFI ***Zhang et al*** SEC'13

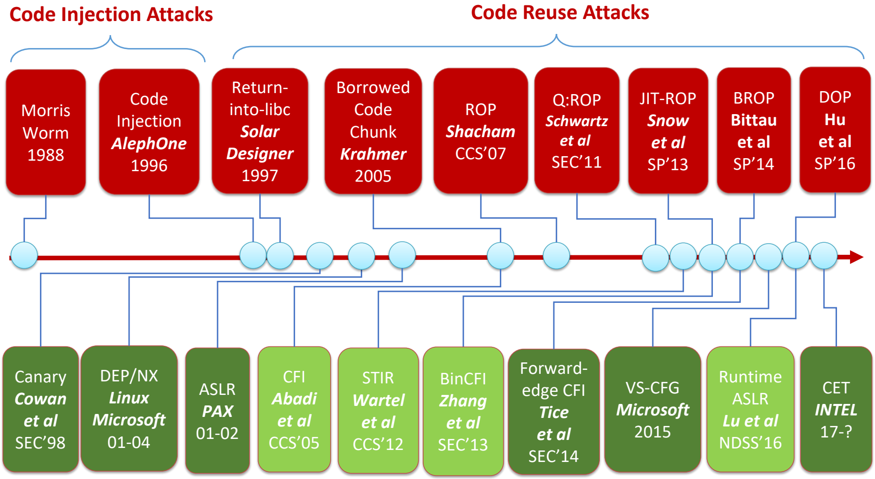
Forward-edge CFI ***Tice et al*** SEC'14

VS-CFG ***Microsoft*** 2015

Runtime ASLR ***Lu et al*** NDSS'16

CET ***INTEL*** 17-?

The Arm Race Between Offense and Defense w/ Memory Corruptions



Code Injection Attacks

Code Reuse Attacks

Morris Worm 1988

Code Injection ***AlephOne*** 1996

Return-into-libc ***Solar Designer*** 1997

Borrowed Code Chunk ***Krahmer*** 2005

ROP ***Shacham*** CCS'07

Q:ROP ***Schwartz et al*** SEC'11

JIT-ROP ***Snow et al*** SP'13

BROP **Bittau et al** SP'14

DOP **Hu et al** SP'16

Canary ***Cowan et al*** SEC'98

DEP/NX ***Linux Microsoft*** 01-04

ASLR ***PAX*** 01-02

CFI ***Abadi et al*** CCS'05

STIR ***Wartel et al*** CCS'12

BinCFI ***Zhang et al*** SEC'13

Forward-edge CFI ***Tice et al*** SEC'14

VS-CFG ***Microsoft*** 2015

Runtime ASLR ***Lu et al*** NDSS'16

CET ***INTEL*** 17-?

The Asymmetry Between Offense and Defense

Offense

1. Code injection
   - NULL-free shellcode
   - NOP sled
   - JMP %ESP
2. Code reuse
   - Return-into-libc
   - ROP
   - JIT-ROP
   - BROP

Defense

1. Stack canary
2. Non-executable memory
3. ASLR
   - Partial ASLR
   - Full ASLR
4. Control flow integrity
5. Toolchain hardening
6. Fuzzing

The Asymmetry Between Offense and Defense

Offense

1. Code injection
   - NULL-free shellcode
   - NOP sled
   - JMP %ESP
2. Code reuse
   - Return-into-libc
   - ROP
   - JIT-ROP
   - BROP

Defense

1. Stack canary
2. Non-executable memory
3. ASLR
   - Partial ASLR
   - Full ASLR
4. Control flow integrity
5. Toolchain hardening
6. Fuzzing

The Asymmetry Between Offense and Defense

Offense

1. Code injection
   - NULL-free shellcode
   - NOP sled
   - JMP %ESP
2. Code reuse
   - Return-into-libc
   - ROP
   - JIT-ROP
   - BROP

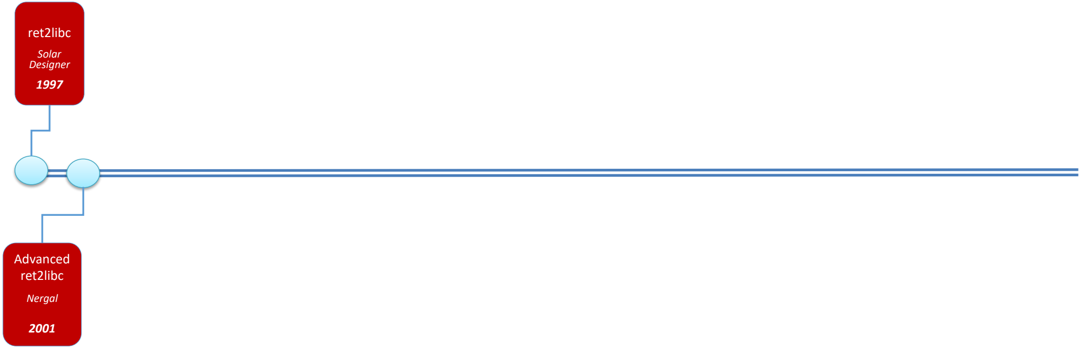
Defense

1. Stack canary
2. Non-executable memory
3. ASLR
   - Partial ASLR
   - Full ASLR
4. Control flow integrity
5. Toolchain hardening
6. Fuzzing

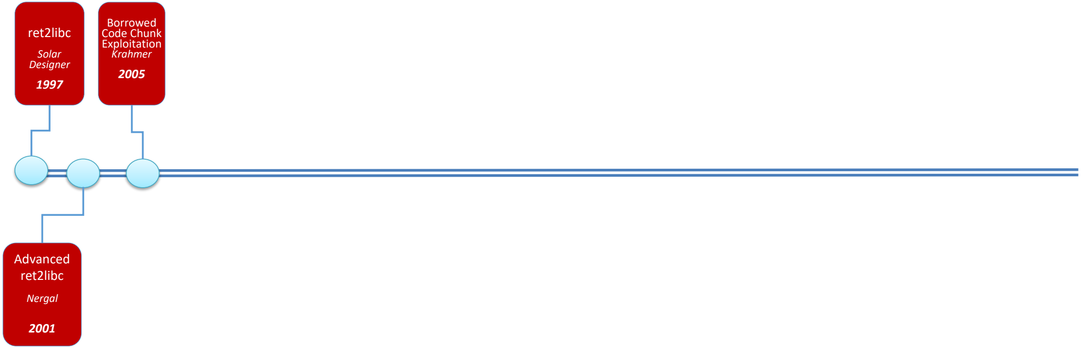
To Win: Succeed just once

To Win: Block every attack

Research on Code Reuse Attacks



Research on Code Reuse Attacks

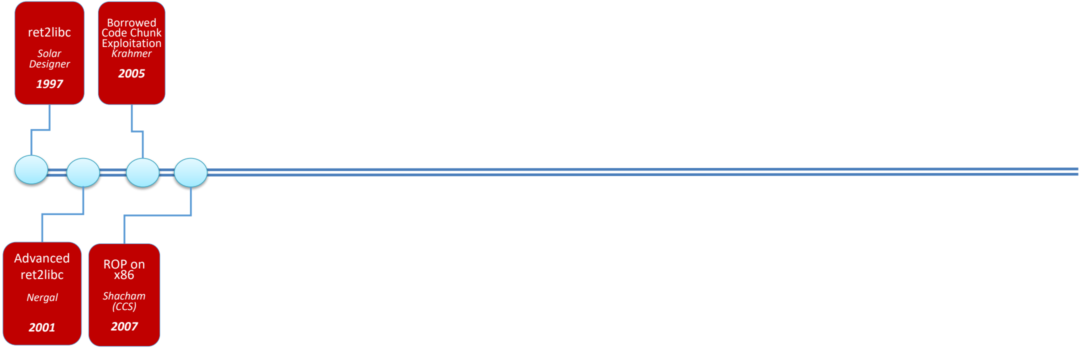


ret2libc
Solar Designer
1997

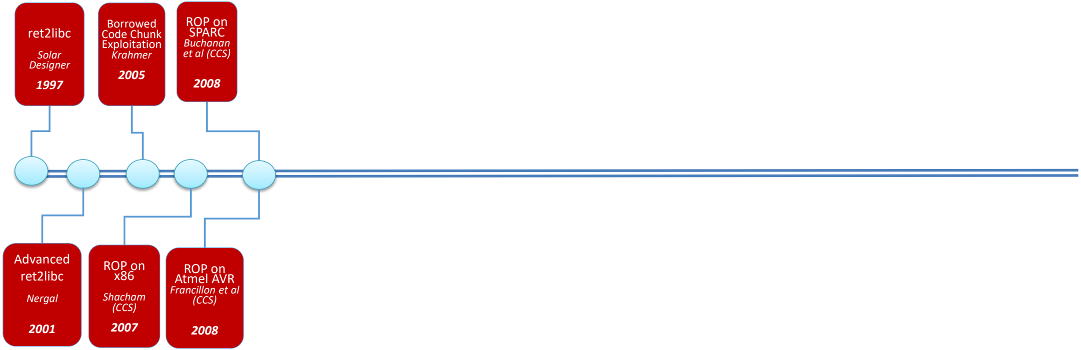
Advanced ret2libc
Nergal
2001

Borrowed Code Chunk Exploitation
Krahmer
2005

Research on Code Reuse Attacks



Research on Code Reuse Attacks


ret2libc
Solar Designer
1997

Advanced ret2libc
Nergal
2001

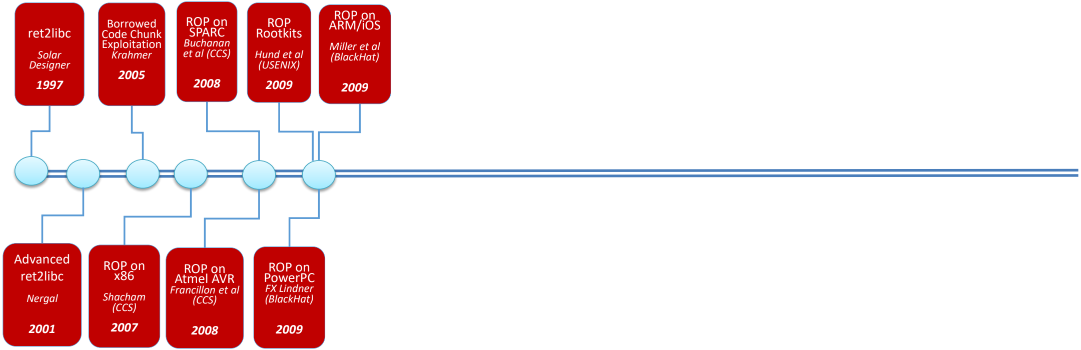
Borrowed Code Chunk Exploitation
Krahmer
2005

ROP on x86
Shacham (CCS)
2007

ROP on SPARC
Buchanan et al (CCS)
2008

ROP on Atmel AVR
Francillon et al (CCS)
2008

Research on Code Reuse Attacks



ret2libc
Solar Designer
1997

Advanced ret2libc
Nergal
2001

Borrowed Code Chunk Exploitation
Krahmer
2005

ROP on x86
Shacham (CCS)
2007

ROP on SPARC
Buchanan et al (CCS)
2008

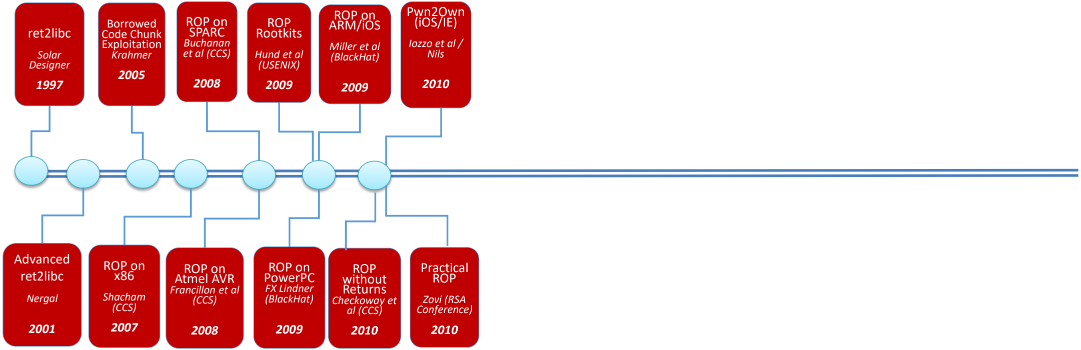
ROP on Atmel AVR
Francillon et al (CCS)
2008

ROP Rootkits
Hund et al (USENIX)
2009

ROP on PowerPC
FX Lindner (BlackHat)
2009

ROP on ARM/iOS
Miller et al (BlackHat)
2009

Research on Code Reuse Attacks



ret2libc
Solar Designer
1997

Advanced ret2libc
Nergal
2001

Borrowed Code Chunk Exploitation
Krahmer
2005

ROP on x86
Shacham (CCS)
2007

ROP on SPARC
Buchanan et al (CCS)
2008

ROP on Atmel AVR
Francillon et al (CCS)
2008

ROP Rootkits
Hund et al (USENIX)
2009

ROP on PowerPC
FX Lindner (BlackHat)
2009

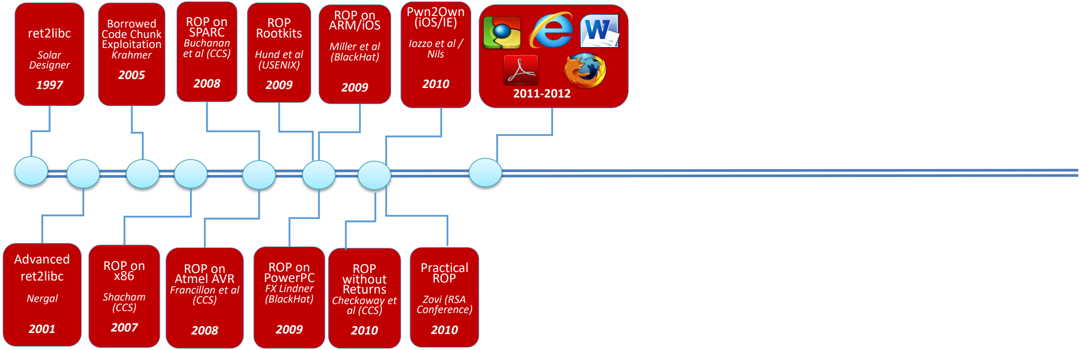
ROP on ARM/iOS
Miller et al (BlackHat)
2009

ROP without Returns
Checkoway et al (CCS)
2010

Pwn2Own (iOS/IE)
Iozzo et al / Nils
2010

Practical ROP
Zovi (RSA Conference)
2010

Research on Code Reuse Attacks



ret2libc
Solar Designer
1997

Borrowed Code Chunk Exploitation
Krahmer
2005

ROP on SPARC
Buchanan et al (CCS)
2008

ROP Rootkits
Hund et al (USENIX)
2009

ROP on ARM/iOS
Miller et al (BlackHat)
2009

Pwn2Own (iOS/IE)
Iozzo et al / Nils
2010

2011-2012

Advanced ret2libc
Nergal
2001

ROP on x86
Shacham (CCS)
2007

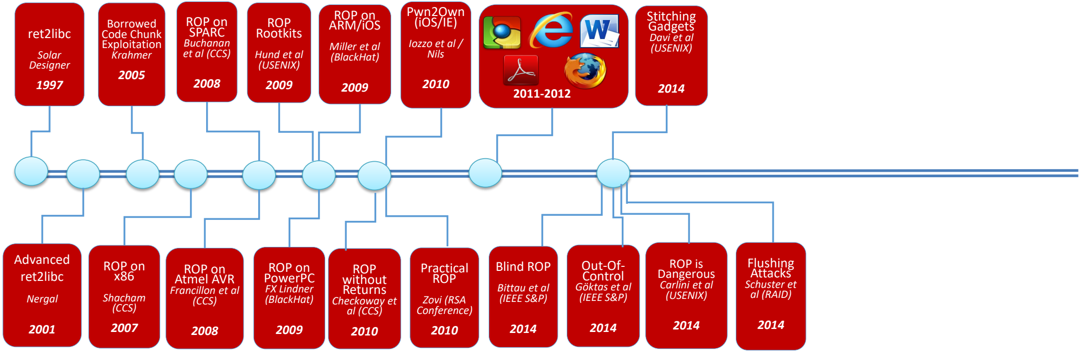
ROP on Atmel AVR
Francillon et al (CCS)
2008

ROP on PowerPC
FX Lindner (BlackHat)
2009

ROP without Returns
Checkoway et al (CCS)
2010

Practical ROP
Zovi (RSA Conference)
2010

Research on Code Reuse Attacks



ret2libc
Solar Designer
1997

Borrowed Code Chunk Exploitation
Krahmer
2005

ROP on SPARC
Buchanan et al (CCS)
2008

ROP Rootkits
Hund et al (USENIX)
2009

ROP on ARM/iOS
Miller et al (BlackHat)
2009

Pwn2Own (iOS/IE)
Iozzo et al / Nils
2010

2011-2012

Stitching Gadgets
Davi et al (USENIX)
2014

Advanced ret2libc
Nergal
2001

ROP on x86
Shacham (CCS)
2007

ROP on Atmel AVR
Francillon et al (CCS)
2008

ROP on PowerPC
FX Lindner (BlackHat)
2009

ROP without Returns
Checkoway et al (CCS)
2010

Practical ROP
Zovi (RSA Conference)
2010

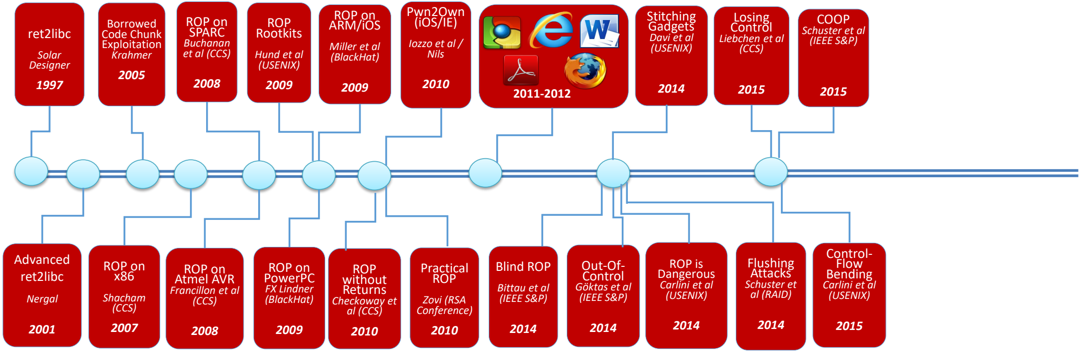
Blind ROP
Bittau et al (IEEE S&P)
2014

Out-Of-Control
Göktas et al (IEEE S&P)
2014

ROP is Dangerous
Carlini et al (USENIX)
2014

Flushing Attacks
Schuster et al (RAID)
2014

Research on Code Reuse Attacks



ret2libc *Solar Designer* ***1997***

Borrowed Code Chunk Exploitation *Krahmer* ***2005***

ROP on SPARC *Buchanan et al (CCS)* ***2008***

ROP Rootkits *Hund et al (USENIX)* ***2009***

ROP on ARM/iOS *Miller et al (BlackHat)* ***2009***

Pwn2Own (iOS/IE) *Iozzo et al / Nils* ***2010***

2011-2012

Stitching Gadgets *Davi et al (USENIX)* ***2014***

Losing Control *Liebchen et al (CCS)* ***2015***

COOP *Schuster et al (IEEE S&P)* ***2015***

Advanced ret2libc *Nergal* ***2001***

ROP on x86 *Shacham (CCS)* ***2007***

ROP on Atmel AVR *Francillon et al (CCS)* ***2008***

ROP on PowerPC *FX Lindner (BlackHat)* ***2009***

ROP without Returns *Checkoway et al (CCS)* ***2010***

Practical ROP *Zovi (RSA Conference)* ***2010***

Blind ROP *Bittau et al (IEEE S&P)* ***2014***

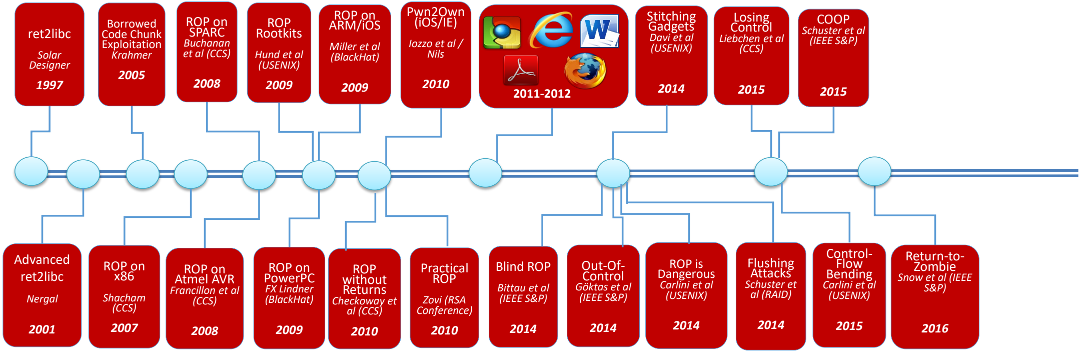
Out-Of-Control *Göktas et al (IEEE S&P)* ***2014***

ROP is Dangerous *Carlini et al (USENIX)* ***2014***

Flushing Attacks *Schuster et al (RAID)* ***2014***

Control-Flow Bending *Carlini et al (USENIX)* ***2015***

Research on Code Reuse Attacks



ret2libc *Solar Designer* ***1997***

Borrowed Code Chunk Exploitation *Krahmer* ***2005***

ROP on SPARC *Buchanan et al (CCS)* ***2008***

ROP Rootkits *Hund et al (USENIX)* ***2009***

ROP on ARM/iOS *Miller et al (BlackHat)* ***2009***

Pwn2Own (iOS/IE) *Iozzo et al / Nils* ***2010***

2011-2012

Stitching Gadgets *Davi et al (USENIX)* ***2014***

Losing Control *Liebchen et al (CCS)* ***2015***

COOP *Schuster et al (IEEE S&P)* ***2015***

Advanced ret2libc *Nergal* ***2001***

ROP on x86 *Shacham (CCS)* ***2007***

ROP on Atmel AVR *Francillon et al (CCS)* ***2008***

ROP on PowerPC *FX Lindner (BlackHat)* ***2009***

ROP without Returns *Checkoway et al (CCS)* ***2010***

Practical ROP *Zovi (RSA Conference)* ***2010***

Blind ROP *Bittau et al (IEEE S&P)* ***2014***

Out-Of-Control *Göktas et al (IEEE S&P)* ***2014***

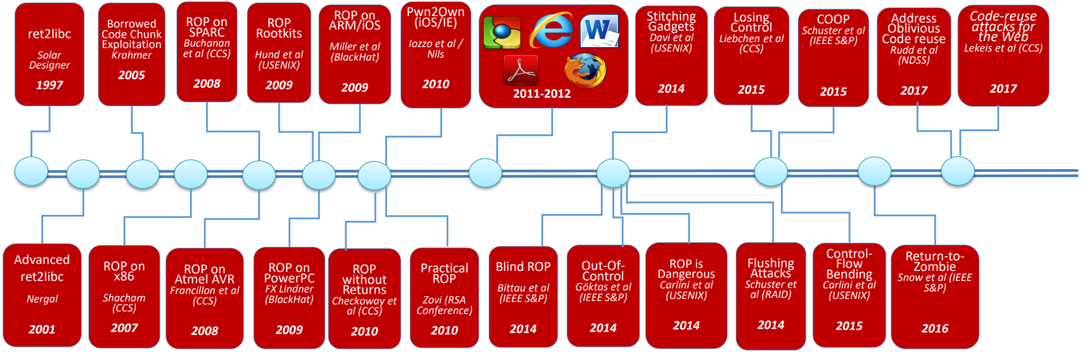
ROP is Dangerous *Carlini et al (USENIX)* ***2014***

Flushing Attacks *Schuster et al (RAID)* ***2014***

Control-Flow Bending *Carlini et al (USENIX)* ***2015***

Return-to-Zombie *Snow et al (IEEE S&P)* ***2016***

Research on Code Reuse Attacks



ret2libc
Solar Designer
1997

Borrowed Code Chunk Exploitation
Krahmer
2005

ROP on SPARC
Buchanan et al (CCS)
2008

ROP Rootkits
Hund et al (USENIX)
2009

ROP on ARM/iOS
Miller et al (BlackHat)
2009

Pwn2Own (iOS/IE)
Iozzo et al / Nils
2010

2011-2012

Stitching Gadgets
Davi et al (USENIX)
2014

Losing Control
Liebchen et al (CCS)
2015

COOP
Schuster et al (IEEE S&P)
2015

Address Oblivious Code reuse
Rudd et al (NDSS)
2017

Code-reuse attacks for the Web
Lekeis et al (CCS)
2017

Advanced ret2libc
Nergal
2001

ROP on x86
Shacham (CCS)
2007

ROP on Atmel AVR
Francillon et al (CCS)
2008

ROP on PowerPC
FX Lindner (BlackHat)
2009

ROP without Returns
Checkoway et al (CCS)
2010

Practical ROP
Zovi (RSA Conference)
2010

Blind ROP
Bittau et al (IEEE S&P)
2014

Out-Of-Control
Göktas et al (IEEE S&P)
2014

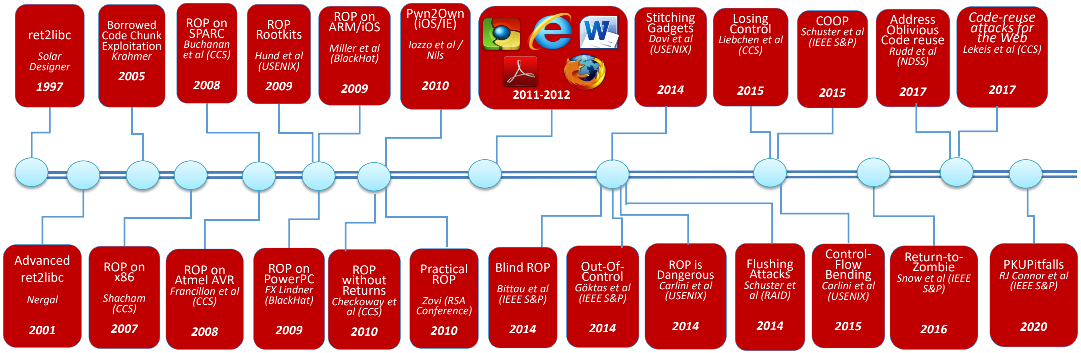
ROP is Dangerous
Carlini et al (USENIX)
2014

Flushing Attacks
Schuster et al (RAID)
2014

Control-Flow Bending
Carlini et al (USENIX)
2015

Return-to-Zombie
Snow et al (IEEE S&P)
2016

Research on Code Reuse Attacks



ret2libc
Solar Designer
1997

Borrowed Code Chunk Exploitation
Krahmer
2005

ROP on SPARC
Buchanan et al (CCS)
2008

ROP Rootkits
Hund et al (USENIX)
2009

ROP on ARM/iOS
Miller et al (BlackHat)
2009

Pwn2Own (iOS/IE)
Iozzo et al / Nils
2010

2011-2012

Stitching Gadgets
Davi et al (USENIX)
2014

Losing Control
Liebchen et al (CCS)
2015

COOP
Schuster et al (IEEE S&P)
2015

Address Oblivious Code reuse
Rudd et al (NDSS)
2017

Code-reuse attacks for the Web
Lekeis et al (CCS)
2017

Advanced ret2libc
Nergal
2001

ROP on x86
Shacham (CCS)
2007

ROP on Atmel AVR
Francillon et al (CCS)
2008

ROP on PowerPC
FX Lindner (BlackHat)
2009

ROP without Returns
Checkoway et al (CCS)
2010

Practical ROP
Zovi (RSA Conference)
2010

Blind ROP
Bittau et al (IEEE S&P)
2014

Out-Of-Control
Göktas et al (IEEE S&P)
2014

ROP is Dangerous
Carlini et al (USENIX)
2014

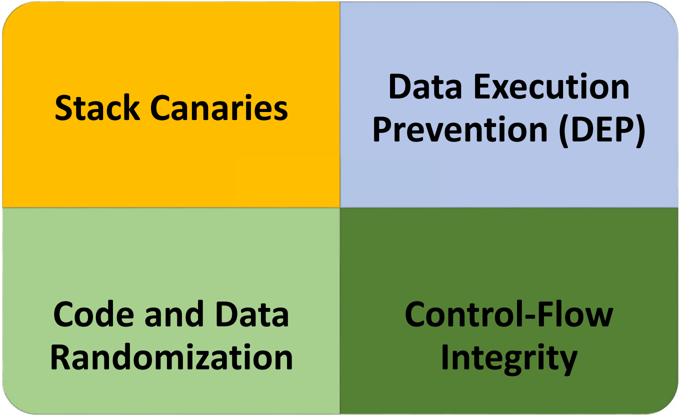
Flushing Attacks
Schuster et al (RAID)
2014

Control-Flow Bending
Carlini et al (USENIX)
2015

Return-to-Zombie
Snow et al (IEEE S&P)
2016

PKUPitfalls
RJ Connor et al (IEEE S&P)
2020

The Run-time Defenses Against Memory Corruptions



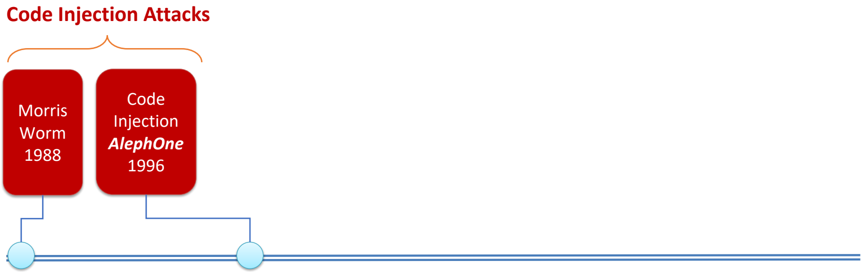
Stack Canaries

Data Execution Prevention (DEP)

Code and Data Randomization

Control-Flow Integrity

Defense Against Code Injection

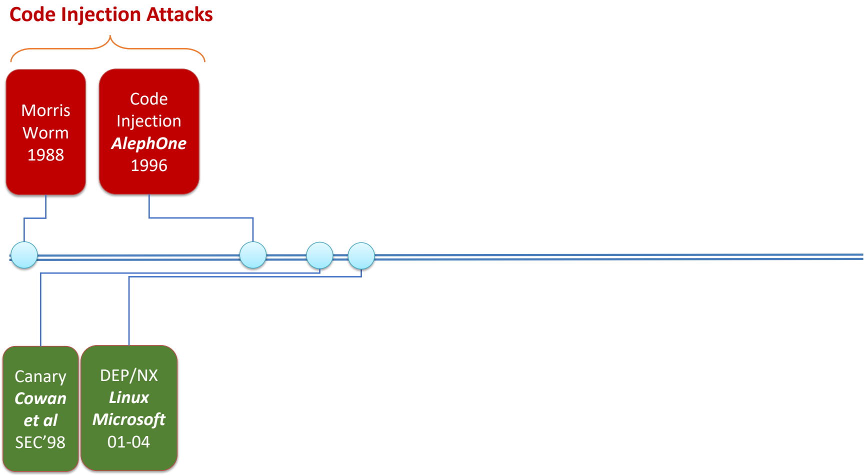


Code Injection Attacks

Morris Worm 1988

Code Injection ***AlephOne*** 1996

Defense Against Code Injection



Code Injection Attacks

Morris Worm 1988

Code Injection *AlephOne* 1996

Canary *Cowan et al* SEC'98

DEP/NX *Linux Microsoft* 01-04

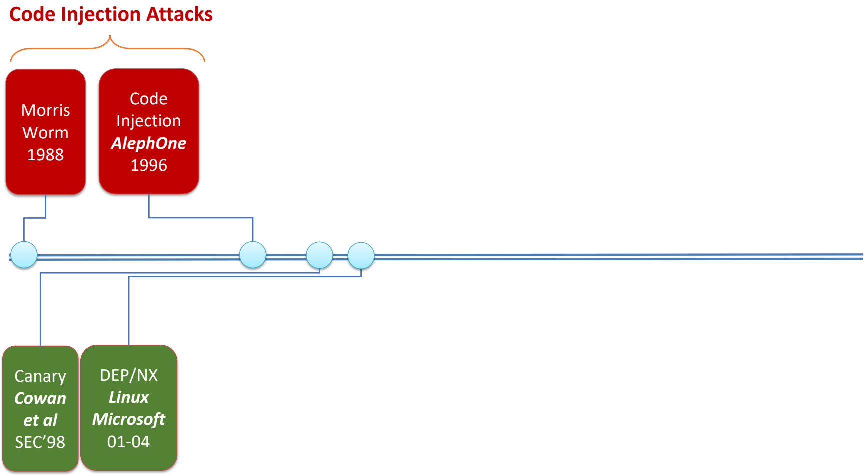
How Does Canary Work

```
    14b7:   55                      push   %ebp
    14b8:   89 e5                   mov    %esp,%ebp
    14ba:   53                      push   %ebx
    14bb:   81 ec 14 02 00 00       sub    $0x214,%esp
    14c1:   e8 6a fe ff ff          call   1330 <__x86.get_pc_thunk.bx>
    14c6:   81 c3 c2 2a 00 00       add    $0x2ac2,%ebx
    14cc:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
    14d2:   89 45 f4                mov    %eax,-0xc(%ebp)
    14d5:   31 c0                   xor    %eax,%eax
    14d7:   83 ec 08                sub    $0x8,%esp
...
    1590:   83 c4 10                add    $0x10,%esp
    1593:   90                      nop
    1594:   8b 45 f4                mov    -0xc(%ebp),%eax
    1597:   65 33 05 14 00 00 00    xor    %gs:0x14,%eax
    159e:   74 05                   je     15a5 <cli_hndl+0xf2>
    15a0:   e8 6b 04 00 00          call   1a10 <__stack_chk_fail_local>
    15a5:   8b 5d fc                mov    -0x4(%ebp),%ebx
    15a8:   c9                      leave
    15a9:   c3                      ret
...
00001a10 <__stack_chk_fail_local>:
    1a10:   f3 0f 1e fb             endbr32
    1a14:   53                      push   %ebx
    1a15:   e8 16 f9 ff ff          call   1330 <__x86.get_pc_thunk.bx>
    1a1a:   81 c3 6e 25 00 00       add    $0x256e,%ebx
    1a20:   83 ec 08                sub    $0x8,%esp
    1a23:   e8 b8 f7 ff ff          call   11e0 <__stack_chk_fail@plt>
```

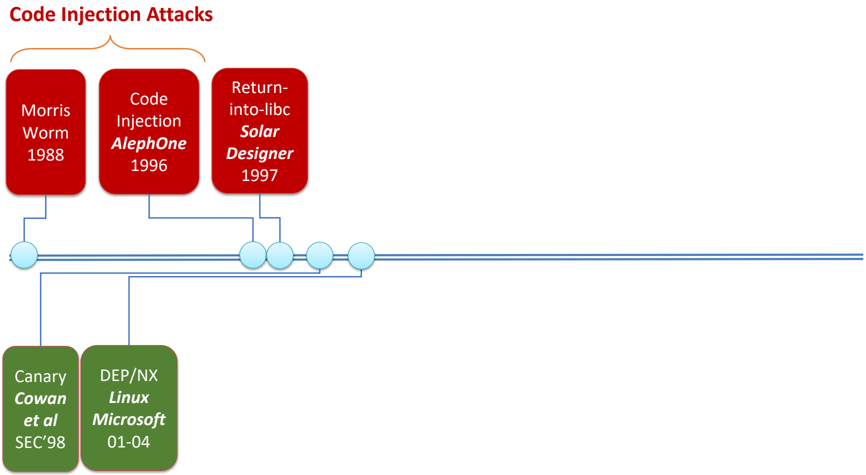
How Does Data Execution Presentation (DEP) or NX Work

```
zlin@ubuntu:~$ cat /proc/self/maps
561f3b5ac000-561f3b5ae000 r--p 00000000 08:05 527214                     /usr/bin/cat
561f3b5ae000-561f3b5b3000 r-xp 00002000 08:05 527214                     /usr/bin/cat
561f3b5b3000-561f3b5b6000 r--p 00007000 08:05 527214                     /usr/bin/cat
561f3b5b6000-561f3b5b7000 r--p 00009000 08:05 527214                     /usr/bin/cat
561f3b5b7000-561f3b5b8000 rw-p 0000a000 08:05 527214                     /usr/bin/cat
561f3ccdf000-561f3cd00000 rw-p 00000000 00:00 0                          [heap]
7fde12208000-7fde1222a000 rw-p 00000000 00:00 0
7fde1222a000-7fde1279a000 r--p 00000000 08:05 524363                     /usr/lib/locale/locale-archive
7fde1279a000-7fde127bf000 r--p 00000000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fde127bf000-7fde12937000 r-xp 00025000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fde12937000-7fde12981000 r--p 0019d000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fde12981000-7fde12982000 ---p 001e7000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fde12982000-7fde12985000 r--p 001e7000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fde12985000-7fde12988000 rw-p 001ea000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fde12988000-7fde1298e000 rw-p 00000000 00:00 0
7fde129a2000-7fde129a3000 r--p 00000000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fde129a3000-7fde129c6000 r-xp 00001000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fde129c6000-7fde129ce000 r--p 00024000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fde129cf000-7fde129d0000 r--p 0002c000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fde129d0000-7fde129d1000 rw-p 0002d000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fde129d1000-7fde129d2000 rw-p 00000000 00:00 0
7fffb5b70000-7fffb5b91000 rw-p 00000000 00:00 0                          [stack]
7fffb5b97000-7fffb5b9a000 r--p 00000000 00:00 0                          [vvar]
7fffb5b9a000-7fffb5b9b000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

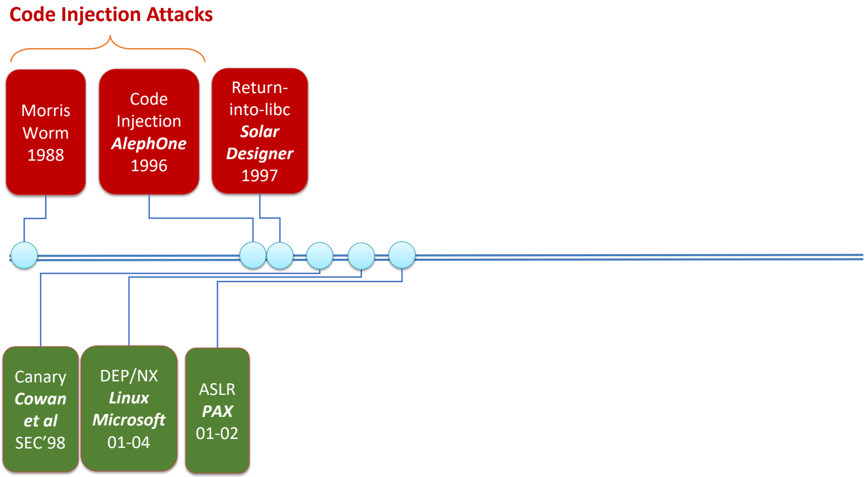
Defense Using Randomization


Code Injection Attacks
Morris Worm 1988
Code Injection *AlephOne* 1996
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04

Defense Using Randomization


Code Injection Attacks
Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc *Solar Designer* 1997
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04

Defense Using Randomization



Code Injection Attacks

Morris Worm 1988

Code Injection ***AlephOne*** 1996

Return-into-libc ***Solar Designer*** 1997

Canary ***Cowan et al*** SEC'98

DEP/NX ***Linux Microsoft*** 01-04

ASLR ***PAX*** 01-02

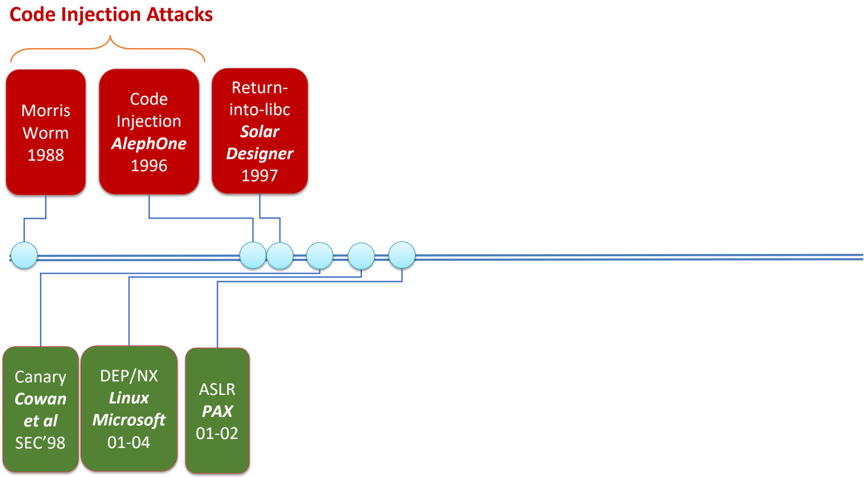
How Does ASLR Work

```
zlin@ubuntu:~$ cat /proc/self/maps
5640ca7b8000-5640ca7ba000 r--p 00000000 08:05 527214                     /usr/bin/cat
5640ca7ba000-5640ca7bf000 r-xp 00002000 08:05 527214                     /usr/bin/cat
5640ca7bf000-5640ca7c2000 r--p 00007000 08:05 527214                     /usr/bin/cat
5640ca7c2000-5640ca7c3000 r--p 00009000 08:05 527214                     /usr/bin/cat
5640ca7c3000-5640ca7c4000 rw-p 0000a000 08:05 527214                     /usr/bin/cat
5640cbc28000-5640cbc49000 rw-p 00000000 00:00 0                          [heap]
7f0767f0d000-7f0767f2f000 rw-p 00000000 00:00 0
7f0767f2f000-7f076849f000 r--p 00000000 08:05 524363                     /usr/lib/locale/locale-archive
7f076849f000-7f07684c4000 r--p 00000000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f07684c4000-7f076863c000 r-xp 00025000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f076863c000-7f0768686000 r--p 0019d000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f0768686000-7f0768687000 ---p 001e7000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f0768687000-7f076868a000 r--p 001e7000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f076868a000-7f076868d000 rw-p 001ea000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f076868d000-7f0768693000 rw-p 00000000 00:00 0
7f07686a7000-7f07686a8000 r--p 00000000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f07686a8000-7f07686cb000 r-xp 00001000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f07686cb000-7f07686d3000 r--p 00024000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f07686d4000-7f07686d5000 r--p 0002c000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f07686d5000-7f07686d6000 rw-p 0002d000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f07686d6000-7f07686d7000 rw-p 00000000 00:00 0
7ffc4b646000-7ffc4b667000 rw-p 00000000 00:00 0                          [stack]
7ffc4b73b000-7ffc4b73e000 r--p 00000000 00:00 0                          [vvar]
7ffc4b73e000-7ffc4b73f000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

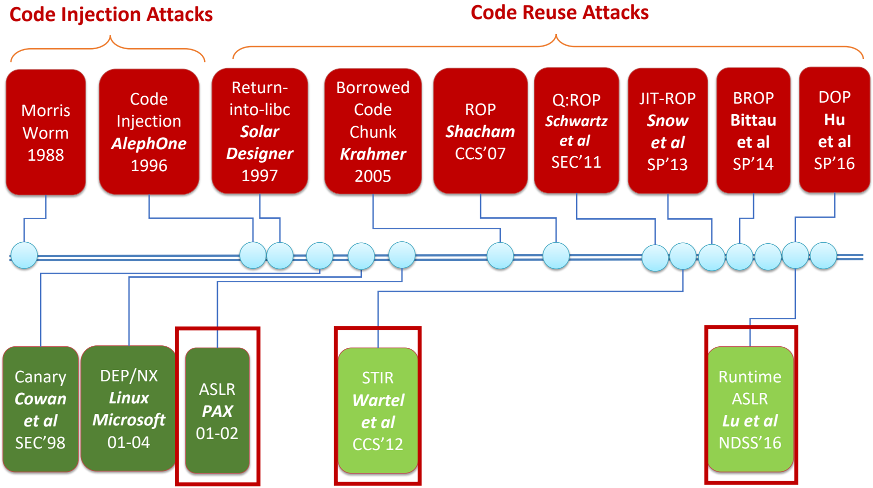
How Does ASLR Work

```
zlin@ubuntu:~$ cat /proc/self/maps
5591966c3000-5591966c5000 r--p 00000000 08:05 527214                     /usr/bin/cat
5591966c5000-5591966ca000 r-xp 00002000 08:05 527214                     /usr/bin/cat
5591966ca000-5591966cd000 r--p 00007000 08:05 527214                     /usr/bin/cat
5591966cd000-5591966ce000 r--p 00009000 08:05 527214                     /usr/bin/cat
5591966ce000-5591966cf000 rw-p 0000a000 08:05 527214                     /usr/bin/cat
5591977ca000-5591977eb000 rw-p 00000000 00:00 0                          [heap]
7f7def86f000-7f7def891000 rw-p 00000000 00:00 0
7f7def891000-7f7defe01000 r--p 00000000 08:05 524363                     /usr/lib/locale/locale-archive
7f7defe01000-7f7defe26000 r--p 00000000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7defe26000-7f7deff9e000 r-xp 00025000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7deff9e000-7f7deffe8000 r--p 0019d000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7deffe8000-7f7deffe9000 ---p 001e7000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7deffe9000-7f7deffec000 r--p 001e7000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7deffec000-7f7deffef000 rw-p 001ea000 08:05 529248                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7deffef000-7f7defff5000 rw-p 00000000 00:00 0
7f7df0009000-7f7df000a000 r--p 00000000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7df000a000-7f7df002d000 r-xp 00001000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7df002d000-7f7df0035000 r--p 00024000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7df0036000-7f7df0037000 r--p 0002c000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7df0037000-7f7df0038000 rw-p 0002d000 08:05 529244                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7df0038000-7f7df0039000 rw-p 00000000 00:00 0
7ffdcdd4c000-7ffdcdd6d000 rw-p 00000000 00:00 0                          [stack]
7ffdcddf0000-7ffdcddf3000 r--p 00000000 00:00 0                          [vvar]
7ffdcddf3000-7ffdcddf4000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

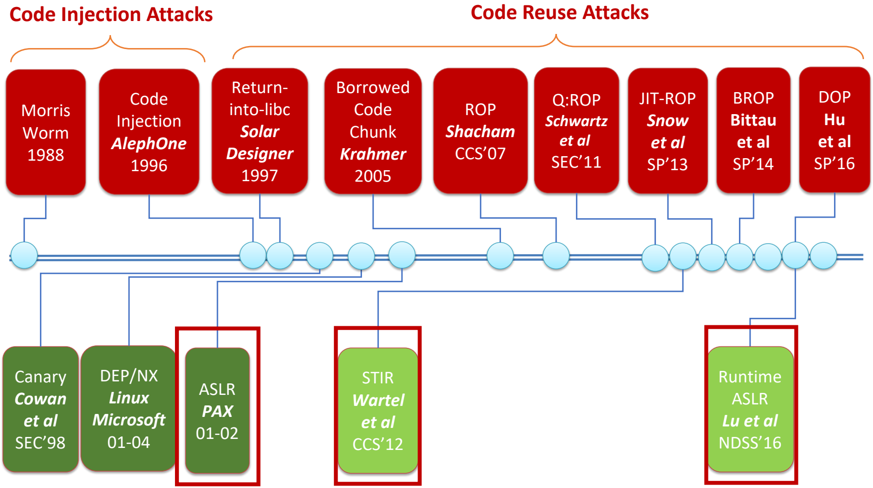
More Defenses Using Randomization


Code Injection Attacks
Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc *Solar Designer* 1997
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04
ASLR *PAX* 01-02

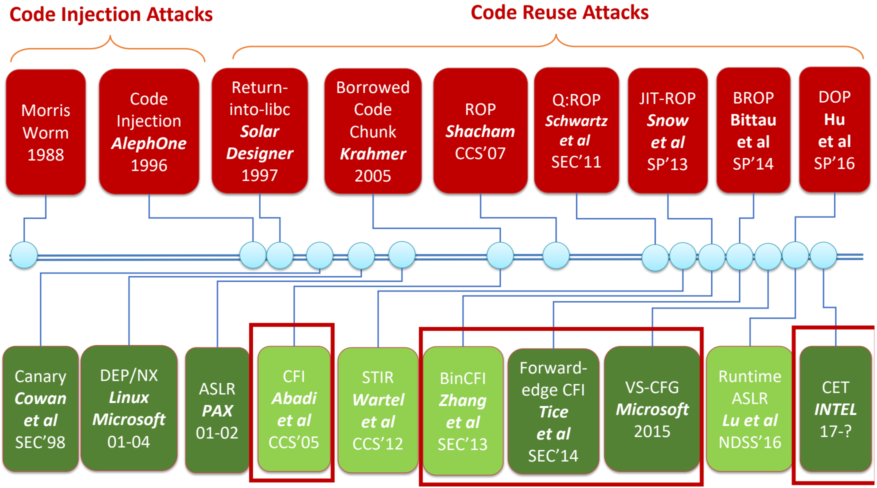
More Defenses Using Randomization


Code Injection Attacks
Code Reuse Attacks
Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc *Solar Designer* 1997
Borrowed Code Chunk *Krahmer* 2005
ROP *Shacham* CCS'07
Q:ROP *Schwartz et al* SEC'11
JIT-ROP *Snow et al* SP'13
BROP **Bittau et al** SP'14
DOP **Hu et al** SP'16
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04
ASLR *PAX* 01-02
STIR *Wartel et al* CCS'12
Runtime ASLR *Lu et al* NDSS'16

Defense Using Control Flow Integrity


Code Injection Attacks
Code Reuse Attacks
Morris Worm 1988
Code Injection *AlephOne* 1996
Return-into-libc *Solar Designer* 1997
Borrowed Code Chunk *Krahmer* 2005
ROP *Shacham* CCS'07
Q:ROP *Schwartz et al* SEC'11
JIT-ROP *Snow et al* SP'13
BROP **Bittau et al** SP'14
DOP **Hu et al** SP'16
Canary *Cowan et al* SEC'98
DEP/NX *Linux Microsoft* 01-04
ASLR *PAX* 01-02
STIR *Wartel et al* CCS'12
Runtime ASLR *Lu et al* NDSS'16

Defense Using Control Flow Integrity



Code Injection Attacks

Code Reuse Attacks

Morris Worm 1988

Code Injection ***AlephOne*** 1996

Return-into-libc ***Solar Designer*** 1997

Borrowed Code Chunk ***Krahmer*** 2005

ROP ***Shacham*** CCS'07

Q:ROP ***Schwartz et al*** SEC'11

JIT-ROP ***Snow et al*** SP'13

BROP **Bittau et al** SP'14

DOP **Hu et al** SP'16

Canary ***Cowan et al*** SEC'98

DEP/NX ***Linux Microsoft*** 01-04

ASLR ***PAX*** 01-02

CFI ***Abadi et al*** CCS'05

STIR ***Wartel et al*** CCS'12

BinCFI ***Zhang et al*** SEC'13

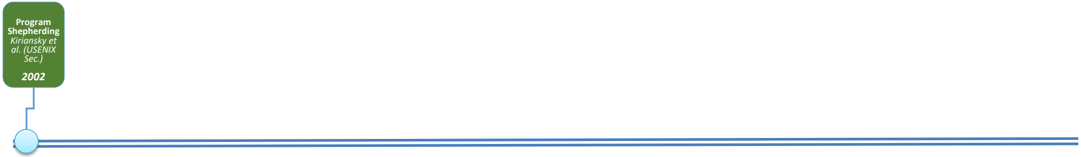
Forward-edge CFI ***Tice et al*** SEC'14

VS-CFG ***Microsoft*** 2015

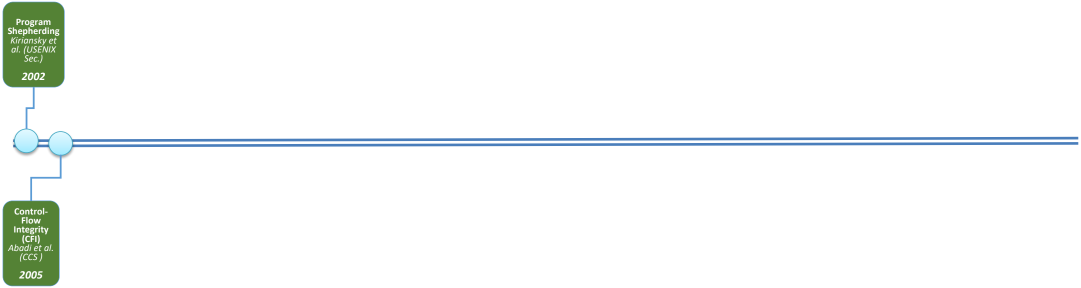
Runtime ASLR ***Lu et al*** NDSS'16

CET ***INTEL*** 17-?

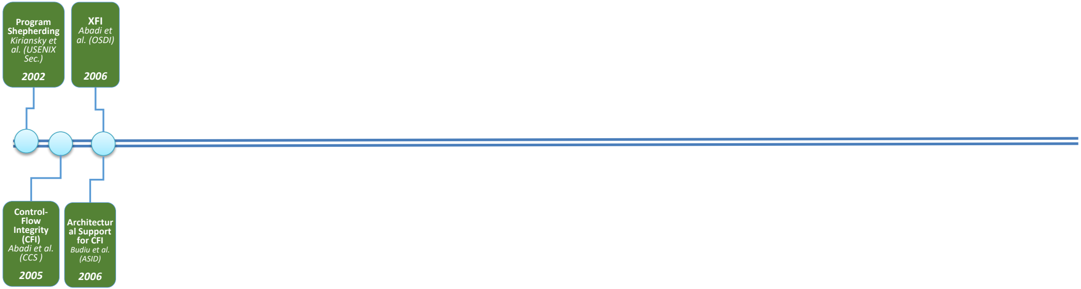
Research on Control Flow Integrity



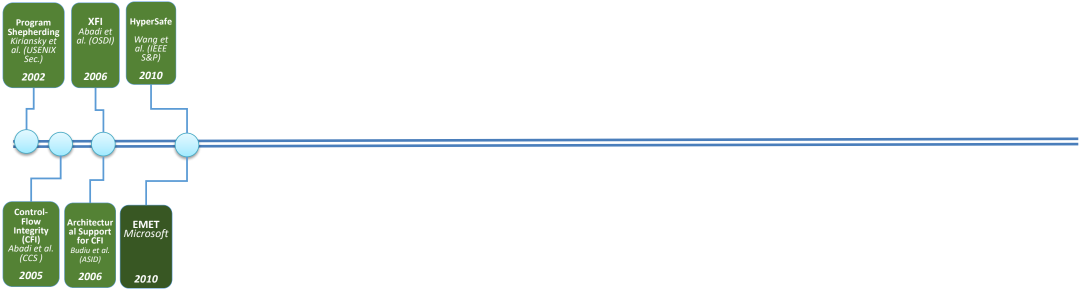
Research on Control Flow Integrity


Program Shepherding *Kiriansky et al. (USENIX Sec.)* ***2002***
Control-Flow Integrity (CFI) *Abadi et al. (CCS)* ***2005***

Research on Control Flow Integrity



Research on Control Flow Integrity



Program Shepherding
Kiriansky et al. (USENIX Sec.)
2002

XFI
Abadi et al. (OSDI)
2006

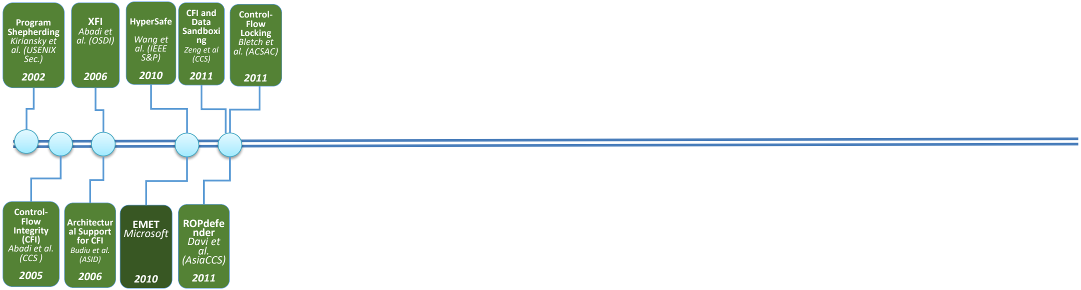
HyperSafe
Wang et al. (IEEE S&P)
2010

Control-Flow Integrity (CFI)
Abadi et al. (CCS)
2005

Architectural Support for CFI
Budiu et al. (ASID)
2006

EMET
Microsoft
2010

Research on Control Flow Integrity



Program Shepherding *Kiriansky et al. (USENIX Sec.)* *2002*

XFI *Abadi et al. (OSDI)* *2006*

HyperSafe *Wang et al. (IEEE S&P)* *2010*

CFI and Data Sandboxing *Zeng et al (CCS)* *2011*

Control-Flow Locking *Bletch et al. (ACSAC)* *2011*

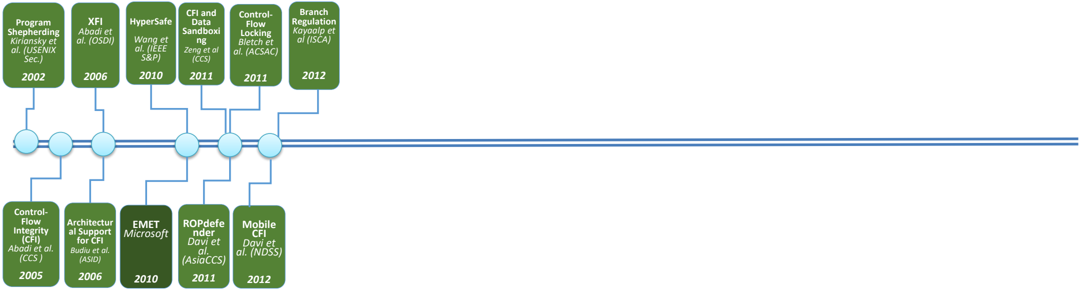
Control-Flow Integrity (CFI) *Abadi et al. (CCS)* *2005*

Architectural Support for CFI *Budiu et al. (ASID)* *2006*

EMET *Microsoft* *2010*

ROPdefender *Davi et al. (AsiaCCS)* *2011*

Research on Control Flow Integrity



Program Shepherding
Kiriansky et al. (USENIX Sec.)
2002

XFI
Abadi et al. (OSDI)
2006

HyperSafe
Wang et al. (IEEE S&P)
2010

CFI and Data Sandboxing
Zeng et al (CCS)
2011

Control-Flow Locking
Bletch et al. (ACSAC)
2011

Branch Regulation
Kayaalp et al (ISCA)
2012

Control-Flow Integrity (CFI)
Abadi et al. (CCS)
2005

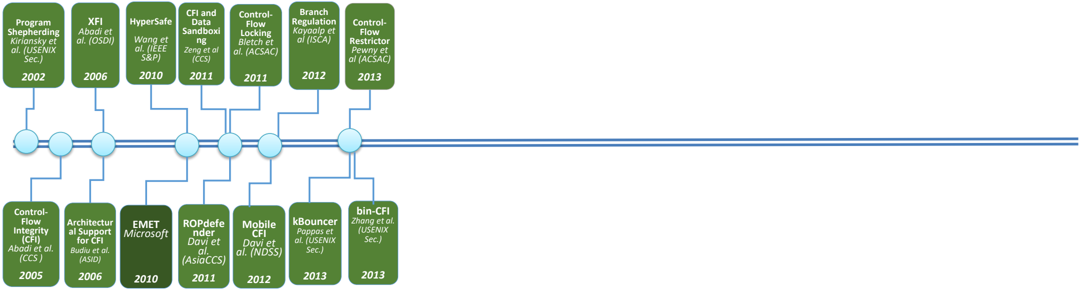
Architectural Support for CFI
Budiu et al. (ASID)
2006

EMET
Microsoft
2010

ROPdefender
Davi et al. (AsiaCCS)
2011

Mobile CFI
Davi et al. (NDSS)
2012

Research on Control Flow Integrity



Program Shepherding *Kiriansky et al. (USENIX Sec.)* ***2002***

XFI *Abadi et al. (OSDI)* ***2006***

HyperSafe *Wang et al. (IEEE S&P)* ***2010***

CFI and Data Sandboxing *Zeng et al (CCS)* ***2011***

Control-Flow Locking *Bletch et al. (ACSAC)* ***2011***

Branch Regulation *Kayaalp et al (ISCA)* ***2012***

Control-Flow Restrictor *Pewny et al (ACSAC)* ***2013***

Control-Flow Integrity (CFI) *Abadi et al. (CCS)* ***2005***

Architectural Support for CFI *Budiu et al. (ASID)* ***2006***

EMET *Microsoft* ***2010***

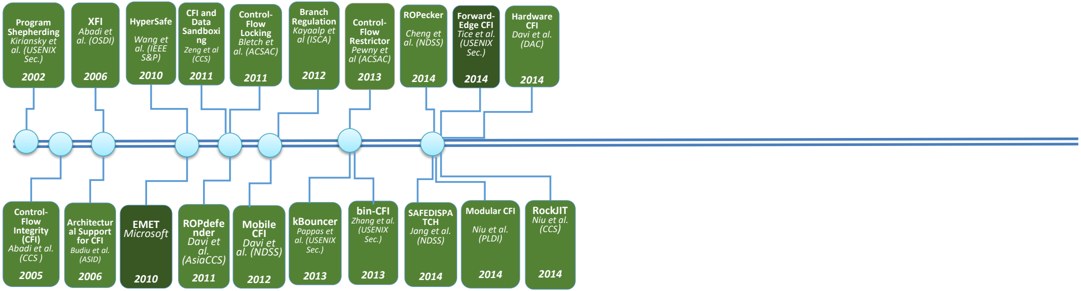
ROPdefender *Davi et al. (AsiaCCS)* ***2011***

Mobile CFI *Davi et al. (NDSS)* ***2012***

kBouncer *Pappas et al. (USENIX Sec.)* ***2013***

bin-CFI *Zhang et al. (USENIX Sec.)* ***2013***

Research on Control Flow Integrity



Program Shepherding *Kiriansky et al. (USENIX Sec.)* *2002*

XFI *Abadi et al. (OSDI)* *2006*

HyperSafe *Wang et al. (IEEE S&P)* *2010*

CFI and Data Sandboxing *Zeng et al (CCS)* *2011*

Control-Flow Locking *Bletch et al. (ACSAC)* *2011*

Branch Regulation *Kayaalp et al (ISCA)* *2012*

Control-Flow Restrictor *Pewny et al (ACSAC)* *2013*

ROPecker *Cheng et al. (NDSS)* *2014*

Forward-Edge CFI *Tice et al. (USENIX Sec.)* *2014*

Hardware CFI *Davi et al. (DAC)* *2014*

Control-Flow Integrity (CFI) *Abadi et al. (CCS)* *2005*

Architectural Support for CFI *Budiu et al. (ASID)* *2006*

EMET *Microsoft* *2010*

ROPdefender *Davi et al. (AsiaCCS)* *2011*

Mobile CFI *Davi et al. (NDSS)* *2012*

kBouncer *Pappas et al. (USENIX Sec.)* *2013*

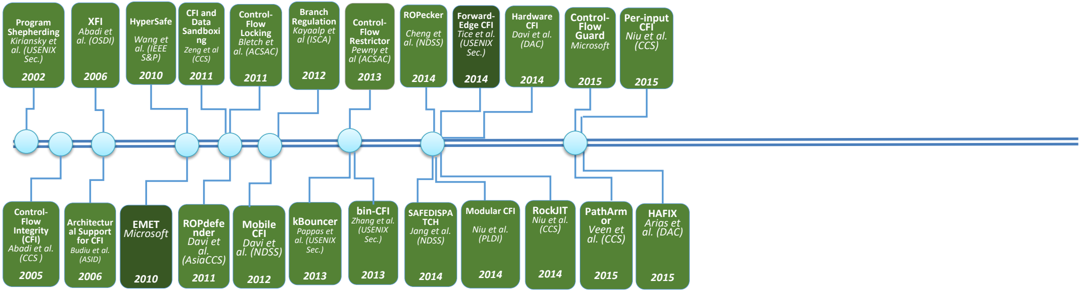
bin-CFI *Zhang et al. (USENIX Sec.)* *2013*

SAFEDISPATCH *Jang et al. (NDSS)* *2014*

Modular CFI *Niu et al. (PLDI)* *2014*

RockJIT *Niu et al. (CCS)* *2014*

Research on Control Flow Integrity



Program Shepherding *Kiriansky et al. (USENIX Sec.)* **2002**

XFI *Abadi et al. (OSDI)* **2006**

HyperSafe *Wang et al. (IEEE S&P)* **2010**

CFI and Data Sandboxing *Zeng et al (CCS)* **2011**

Control-Flow Locking *Bletch et al. (ACSAC)* **2011**

Branch Regulation *Kayaalp et al (ISCA)* **2012**

Control-Flow Restrictor *Pewny et al (ACSAC)* **2013**

ROPecker *Cheng et al. (NDSS)* **2014**

Forward-Edge CFI *Tice et al. (USENIX Sec.)* **2014**

Hardware CFI *Davi et al. (DAC)* **2014**

Control-Flow Guard *Microsoft* **2015**

Per-input CFI *Niu et al. (CCS)* **2015**

Control-Flow Integrity (CFI) *Abadi et al. (CCS)* **2005**

Architectural Support for CFI *Budiu et al. (ASID)* **2006**

EMET *Microsoft* **2010**

ROPdefender *Davi et al. (AsiaCCS)* **2011**

Mobile CFI *Davi et al. (NDSS)* **2012**

kBouncer *Pappas et al. (USENIX Sec.)* **2013**

bin-CFI *Zhang et al. (USENIX Sec.)* **2013**

SAFEDISPATCH *Jang et al. (NDSS)* **2014**

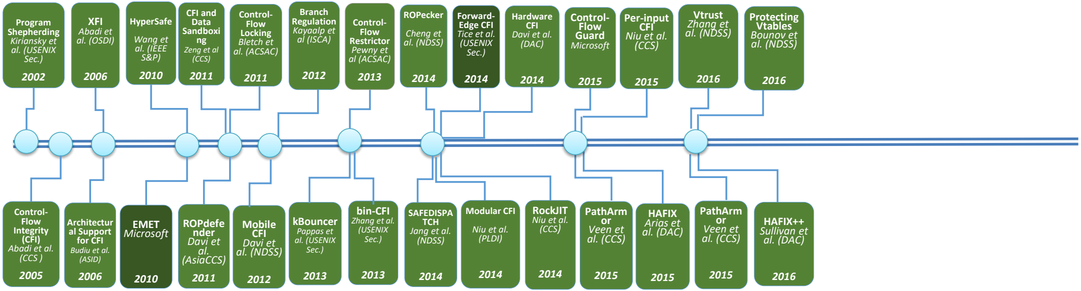
Modular CFI *Niu et al. (PLDI)* **2014**

RockJIT *Niu et al. (CCS)* **2014**

PathArmor *Veen et al. (CCS)* **2015**

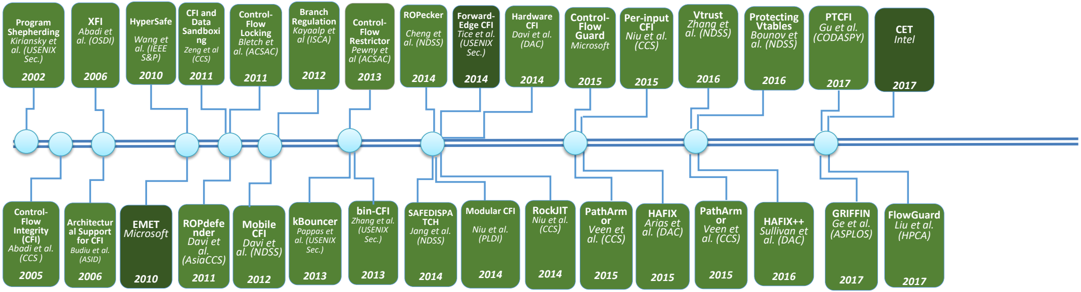
HAFIX *Arias et al. (DAC)* **2015**

Research on Control Flow Integrity



- Program Shepherding *Kiriansky et al. (USENIX Sec.)* 2002
- XFI *Abadi et al. (OSDI)* 2006
- HyperSafe *Wang et al. (IEEE S&P)* 2010
- CFI and Data Sandboxing *Zeng et al (CCS)* 2011
- Control-Flow Locking *Bletch et al. (ACSAC)* 2011
- Branch Regulation *Kayaalp et al (ISCA)* 2012
- Control-Flow Restrictor *Pewny et al (ACSAC)* 2013
- ROPecker *Cheng et al. (NDSS)* 2014
- Forward-Edge CFI *Tice et al. (USENIX Sec.)* 2014
- Hardware CFI *Davi et al. (DAC)* 2014
- Control-Flow Guard *Microsoft* 2015
- Per-input CFI *Niu et al. (CCS)* 2015
- Vtrust *Zhang et al. (NDSS)* 2016
- Protecting Vtables *Bounov et al. (NDSS)* 2016
- Control-Flow Integrity (CFI) *Abadi et al. (CCS)* 2005
- Architectural Support for CFI *Budiu et al. (ASID)* 2006
- EMET *Microsoft* 2010
- ROPdefender *Davi et al. (AsiaCCS)* 2011
- Mobile CFI *Davi et al. (NDSS)* 2012
- kBouncer *Pappas et al. (USENIX Sec.)* 2013
- bin-CFI *Zhang et al. (USENIX Sec.)* 2013
- SAFEDISPATCH *Jang et al. (NDSS)* 2014
- Modular CFI *Niu et al. (PLDI)* 2014
- RockJIT *Niu et al. (CCS)* 2014
- PathArmor *Veen et al. (CCS)* 2015
- HAFIX *Arias et al. (DAC)* 2015
- PathArmor *Veen et al. (CCS)* 2015
- HAFIX++ *Sullivan et al. (DAC)* 2016

Research on Control Flow Integrity



Program Shepherding *Kiriansky et al. (USENIX Sec.)* **2002**

XFI *Abadi et al. (OSDI)* **2006**

HyperSafe *Wang et al. (IEEE S&P)* **2010**

CFI and Data Sandboxing *Zeng et al (CCS)* **2011**

Control-Flow Locking *Bletch et al. (ACSAC)* **2011**

Branch Regulation *Kayaalp et al (ISCA)* **2012**

Control-Flow Restrictor *Pewny et al (ACSAC)* **2013**

ROPecker *Cheng et al. (NDSS)* **2014**

Forward-Edge CFI *Tice et al. (USENIX Sec.)* **2014**

Hardware CFI *Davi et al. (DAC)* **2014**

Control-Flow Guard *Microsoft* **2015**

Per-input CFI *Niu et al. (CCS)* **2015**

Vtrust *Zhang et al. (NDSS)* **2016**

Protecting Vtables *Bounov et al. (NDSS)* **2016**

PTCFI *Gu et al. (CODASPY)* **2017**

CET *Intel* **2017**

Control-Flow Integrity (CFI) *Abadi et al. (CCS)* **2005**

Architectural Support for CFI *Budiu et al. (ASID)* **2006**

EMET *Microsoft* **2010**

ROPdefender *Davi et al. (AsiaCCS)* **2011**

Mobile CFI *Davi et al. (NDSS)* **2012**

kBouncer *Pappas et al. (USENIX Sec.)* **2013**

bin-CFI *Zhang et al. (USENIX Sec.)* **2013**

SAFEDISPATCH *Jang et al. (NDSS)* **2014**

Modular CFI *Niu et al. (PLDI)* **2014**

RockJIT *Niu et al. (CCS)* **2014**

PathArmor *Veen et al. (CCS)* **2015**

HAFIX *Arias et al. (DAC)* **2015**

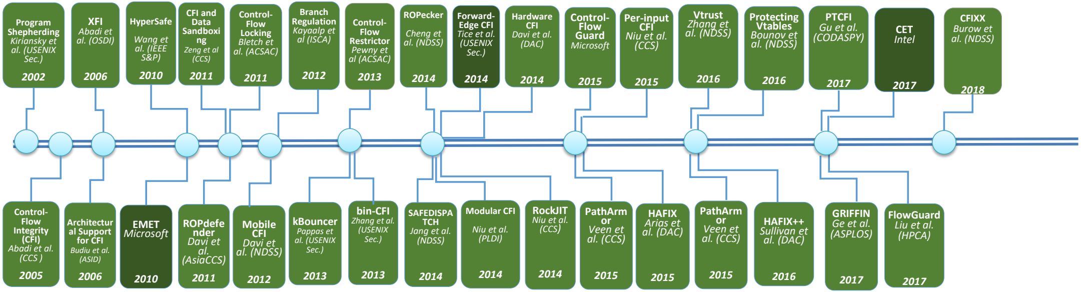
PathArmor *Veen et al. (CCS)* **2015**

HAFIX++ *Sullivan et al. (DAC)* **2016**

GRIFFIN *Ge et al. (ASPLOS)* **2017**

FlowGuard *Liu et al. (HPCA)* **2017**

Research on Control Flow Integrity



Program Shepherding *Kiriansky et al. (USENIX Sec.)* **2002**

XFI *Abadi et al. (OSDI)* **2006**

HyperSafe *Wang et al. (IEEE S&P)* **2010**

CFI and Data Sandboxing *Zeng et al (CCS)* **2011**

Control-Flow Locking *Bletch et al. (ACSAC)* **2011**

Branch Regulation *Kayaalp et al (ISCA)* **2012**

Control-Flow Restrictor *Pewny et al (ACSAC)* **2013**

ROPecker *Cheng et al. (NDSS)* **2014**

Forward-Edge CFI *Tice et al. (USENIX Sec.)* **2014**

Hardware CFI *Davi et al. (DAC)* **2014**

Control-Flow Guard *Microsoft* **2015**

Per-input CFI *Niu et al. (CCS)* **2015**

Vtrust *Zhang et al. (NDSS)* **2016**

Protecting Vtables *Bounov et al. (NDSS)* **2016**

PTCFI *Gu et al. (CODASPY)* **2017**

CET *Intel* **2017**

CFIXX *Burow et al. (NDSS)* **2018**

Control-Flow Integrity (CFI) *Abadi et al. (CCS)* **2005**

Architectural Support for CFI *Budiu et al. (ASID)* **2006**

EMET *Microsoft* **2010**

ROPdefender *Davi et al. (AsiaCCS)* **2011**

Mobile CFI *Davi et al. (NDSS)* **2012**

kBouncer *Pappas et al. (USENIX Sec.)* **2013**

bin-CFI *Zhang et al. (USENIX Sec.)* **2013**

SAFEDISPATCH *Jang et al. (NDSS)* **2014**

Modular CFI *Niu et al. (PLDI)* **2014**

RockJIT *Niu et al. (CCS)* **2014**

PathArmor *Veen et al. (CCS)* **2015**

HAFIX *Arias et al. (DAC)* **2015**

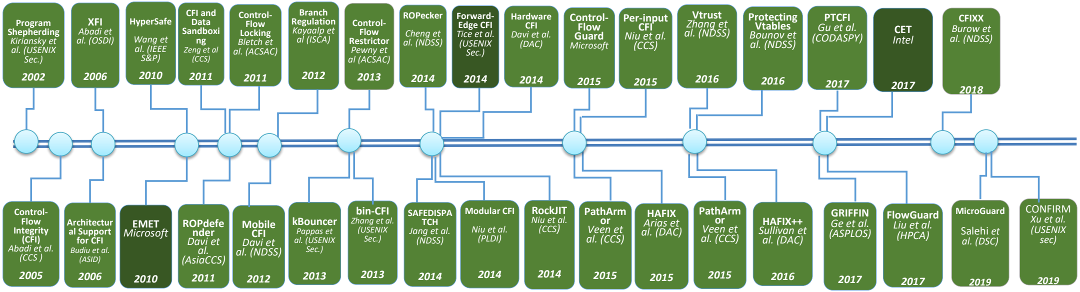
PathArmor *Veen et al. (CCS)* **2015**

HAFIX++ *Sullivan et al. (DAC)* **2016**

GRIFFIN *Ge et al. (ASPLOS)* **2017**

FlowGuard *Liu et al. (HPCA)* **2017**

Research on Control Flow Integrity



Program Shepherding *Kiriansky et al. (USENIX Sec.)* **2002**

XFI *Abadi et al. (OSDI)* **2006**

HyperSafe *Wang et al. (IEEE S&P)* **2010**

CFI and Data Sandboxing *Zeng et al (CCS)* **2011**

Control-Flow Locking *Bletch et al. (ACSAC)* **2011**

Branch Regulation *Kayaalp et al (ISCA)* **2012**

Control-Flow Restrictor *Pewny et al (ACSAC)* **2013**

ROPecker *Cheng et al. (NDSS)* **2014**

Forward-Edge CFI *Tice et al. (USENIX Sec.)* **2014**

Hardware CFI *Davi et al. (DAC)* **2014**

Control-Flow Guard *Microsoft* **2015**

Per-input CFI *Niu et al. (CCS)* **2015**

Vtrust *Zhang et al. (NDSS)* **2016**

Protecting Vtables *Bounov et al. (NDSS)* **2016**

PTCFI *Gu et al. (CODASPY)* **2017**

CET *Intel* **2017**

CFIXX *Burow et al. (NDSS)* **2018**

Control-Flow Integrity (CFI) *Abadi et al. (CCS)* **2005**

Architectural Support for CFI *Budiu et al. (ASID)* **2006**

EMET *Microsoft* **2010**

ROPdefender *Davi et al. (AsiaCCS)* **2011**

Mobile CFI *Davi et al. (NDSS)* **2012**

kBouncer *Pappas et al. (USENIX Sec.)* **2013**

bin-CFI *Zhang et al. (USENIX Sec.)* **2013**

SAFEDISPATCH *Jang et al. (NDSS)* **2014**

Modular CFI *Niu et al. (PLDI)* **2014**

RockJIT *Niu et al. (CCS)* **2014**

PathArmor *Veen et al. (CCS)* **2015**

HAFIX *Arias et al. (DAC)* **2015**

PathArmor *Veen et al. (CCS)* **2015**

HAFIX++ *Sullivan et al. (DAC)* **2016**

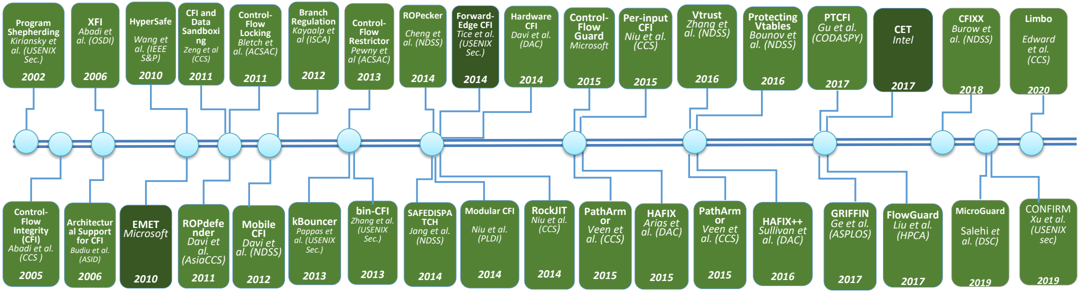
GRIFFIN *Ge et al. (ASPLOS)* **2017**

FlowGuard *Liu et al. (HPCA)* **2017**

MicroGuard Salehi et al. (DSC) **2019**

CONFIRM *Xu et al. (USENIX sec)* **2019**

Research on Control Flow Integrity



Program Shepherding *Kiriansky et al. (USENIX Sec.)* **2002**

XFI *Abadi et al. (OSDI)* **2006**

HyperSafe *Wang et al. (IEEE S&P)* **2010**

CFI and Data Sandboxing *Zeng et al (CCS)* **2011**

Control-Flow Locking *Bletch et al. (ACSAC)* **2011**

Branch Regulation *Kayaalp et al (ISCA)* **2012**

Control-Flow Restrictor *Pewny et al (ACSAC)* **2013**

ROPecker *Cheng et al. (NDSS)* **2014**

Forward-Edge CFI *Tice et al. (USENIX Sec.)* **2014**

Hardware CFI *Davi et al. (DAC)* **2014**

Control-Flow Guard *Microsoft* **2015**

Per-input CFI *Niu et al. (CCS)* **2015**

Vtrust *Zhang et al. (NDSS)* **2016**

Protecting Vtables *Bounov et al. (NDSS)* **2016**

PTCFI *Gu et al. (CODASPY)* **2017**

CET *Intel* **2017**

CFIXX *Burow et al. (NDSS)* **2018**

Limbo *Edward et al. (CCS)* **2020**

Control-Flow Integrity (CFI) *Abadi et al. (CCS)* **2005**

Architectural Support for CFI *Budiu et al. (ASID)* **2006**

EMET *Microsoft* **2010**

ROPdefender *Davi et al. (AsiaCCS)* **2011**

Mobile CFI *Davi et al. (NDSS)* **2012**

kBouncer *Pappas et al. (USENIX Sec.)* **2013**

bin-CFI *Zhang et al. (USENIX Sec.)* **2013**

SAFEDISPATCH *Jang et al. (NDSS)* **2014**

Modular CFI *Niu et al. (PLDI)* **2014**

RockJIT *Niu et al. (CCS)* **2014**

PathArmor *Veen et al. (CCS)* **2015**

HAFIX *Arias et al. (DAC)* **2015**

PathArmor *Veen et al. (CCS)* **2015**

HAFIX++ *Sullivan et al. (DAC)* **2016**

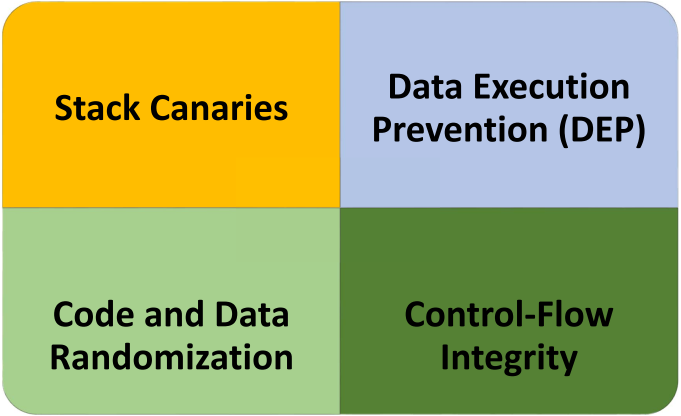
GRIFFIN *Ge et al. (ASPLOS)* **2017**

FlowGuard *Liu et al. (HPCA)* **2017**

MicroGuard Salehi *et al. (DSC)* **2019**

CONFIRM *Xu et al. (USENIX sec)* **2019**

The Practical Run-time Defenses Against Memory Corruptions



Stack Canaries

Data Execution Prevention (DEP)

Code and Data Randomization

Control-Flow Integrity

Other Approaches: Proactive Vulnerability Identification

Fuzzing

1. Identifying vulnerabilities before attackers
   - Blackbox (dumb) fuzzing
   - Generational aka grammar-based fuzzing
   - Whitebox fuzzing with SAGE
     - Looking at symbolic execution of the code
   - Evolutionary fuzzing with afl
     - Grey-box, observing execution of the (instrumented) code

Other Approaches: Proactive Vulnerability Identification

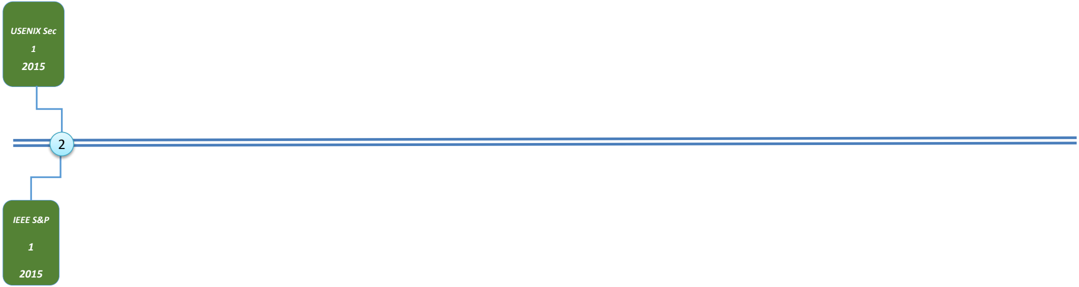
Fuzzing

1. Identifying vulnerabilities before attackers
   - Blackbox (dumb) fuzzing
   - Generational aka grammar-based fuzzing
   - Whitebox fuzzing with SAGE
     - Looking at symbolic execution of the code
   - Evolutionary fuzzing with afl
     - Grey-box, observing execution of the (instrumented) code

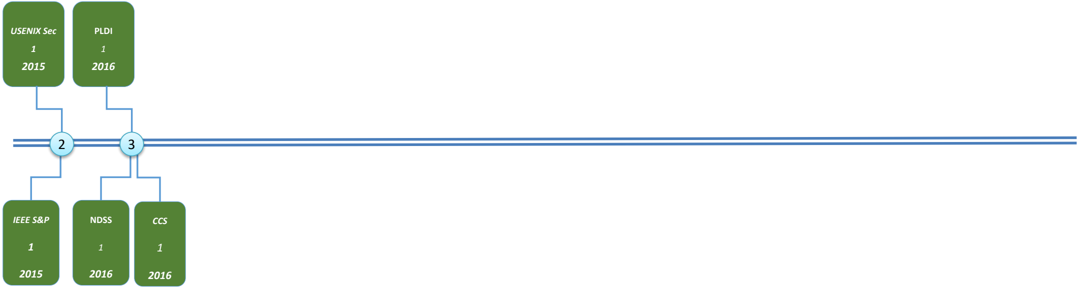
AFL (American Fuzzy Lop)

1. Support software w/
   - Source code (using a compiler flag)
     - C/C++/Object-C
     - Hand-written assembly
   - Binary (executed in an emulator)
2. Identified many 0-day vulnerabilities
3. Highly practical

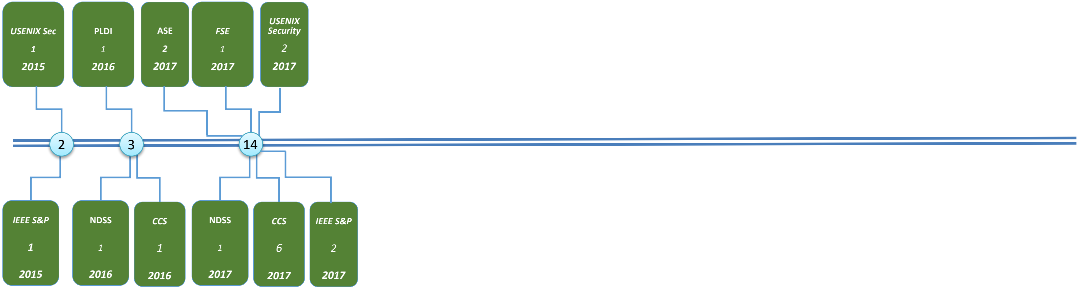
Recent Research on Fuzzing



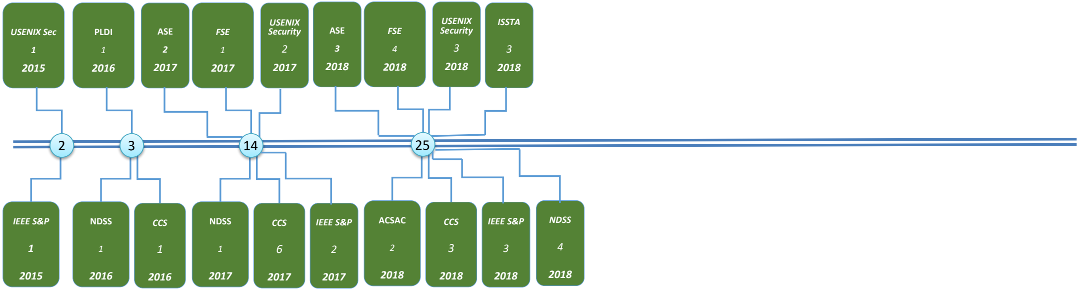
Recent Research on Fuzzing



Recent Research on Fuzzing



Recent Research on Fuzzing



USENIX Sec 1 2015

PLDI 1 2016

ASE 2 2017

FSE 1 2017

USENIX Security 2 2017

ASE 3 2018

FSE 4 2018

USENIX Security 3 2018

ISSTA 3 2018

2

3

14

25

IEEE S&P 1 2015

NDSS 1 2016

CCS 1 2016

NDSS 1 2017

CCS 6 2017

IEEE S&P 2 2017

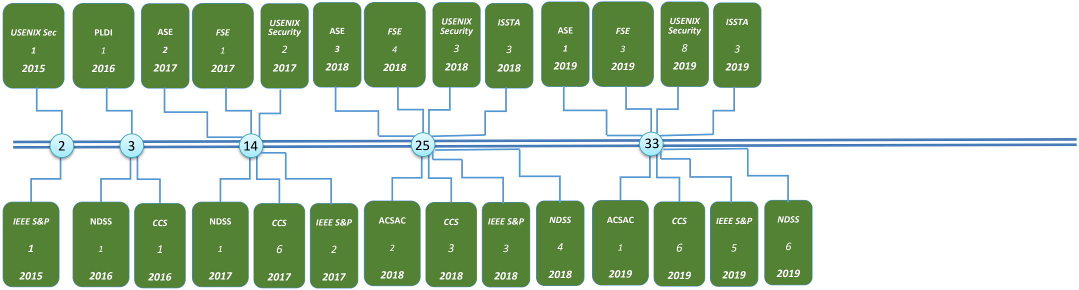
ACSAC 2 2018

CCS 3 2018

IEEE S&P 3 2018

NDSS 4 2018

Recent Research on Fuzzing



USENIX Sec 1 2015

PLDI 1 2016

ASE 2 2017

FSE 1 2017

USENIX Security 2 2017

ASE 3 2018

FSE 4 2018

USENIX Security 3 2018

ISSTA 3 2018

ASE 1 2019

FSE 3 2019

USENIX Security 8 2019

ISSTA 3 2019

2

3

14

25

33

IEEE S&P 1 2015

NDSS 1 2016

CCS 1 2016

NDSS 1 2017

CCS 6 2017

IEEE S&P 2 2017

ACSAC 2 2018

CCS 3 2018

IEEE S&P 3 2018

NDSS 4 2018

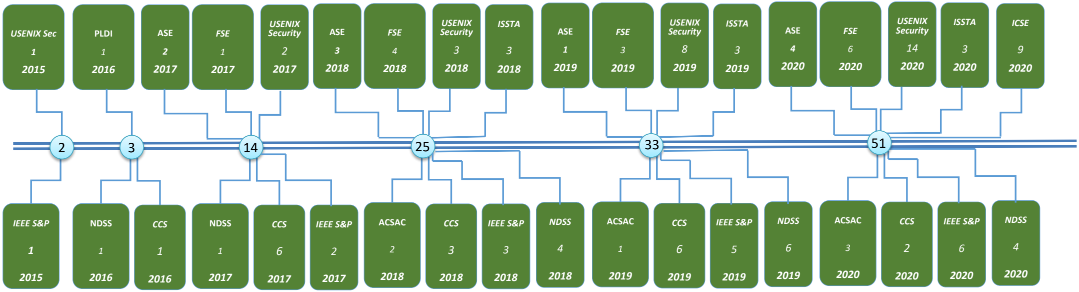
ACSAC 1 2019

CCS 6 2019

IEEE S&P 5 2019

NDSS 6 2019

Recent Research on Fuzzing



USENIX Sec 1 2015
PLDI 1 2016
ASE 2 2017
FSE 1 2017
USENIX Security 2 2017
ASE 3 2018
FSE 4 2018
USENIX Security 3 2018
ISSTA 3 2018
ASE 1 2019
FSE 3 2019
USENIX Security 8 2019
ISSTA 3 2019
ASE 4 2020
FSE 6 2020
USENIX Security 14 2020
ISSTA 3 2020
ICSE 9 2020

2
3
14
25
33
51

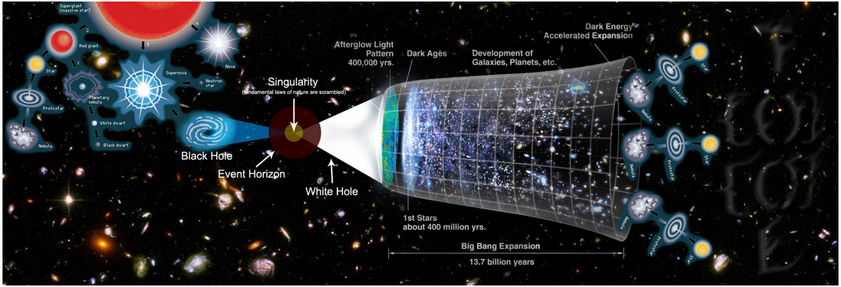
IEEE S&P 1 2015
NDSS 1 2016
CCS 1 2016
NDSS 1 2017
CCS 6 2017
IEEE S&P 2 2017
ACSAC 2 2018
CCS 3 2018
IEEE S&P 3 2018
NDSS 4 2018
ACSAC 1 2019
CCS 6 2019
IEEE S&P 5 2019
NDSS 6 2019
ACSAC 3 2020
CCS 2 2020
IEEE S&P 6 2020
NDSS 4 2020

Fuzzing Papers

Our Recent Efforts: Binary Code Rewriting and Hardening

1. Superset Disassembly: Statically Rewriting x86 Binaries Without Heuristics. In NDSS 2018
2. SelectiveTaint: Efficient Data Flow Tracking With Static Binary Rewriting. In USENIX Security 2021

Multiverse: the first heuristic-free static binary rewriter


Supergiant (massive star)
Red giant
Star
Protostar
Nebula
Planetary nebula
White dwarf
Black dwarf
Supernova
Neutron star
Nova
1
2
3
4
Black Hole
Event Horizon
Singularity
(fundamental laws of nature are scrambled)
White Hole
Afterglow Light Pattern 400,000 yrs.
Dark Ages
Development of Galaxies, Planets, etc.
Dark Energy Accelerated Expansion
1st Stars about 400 million yrs.
Big Bang Expansion
13.7 billion years
Nebula
Protostar
Star

“The Quantum Universe: Everything That Can Happen Does Happen”, a 2011 book by the theoretical physicists Brian Cox and Jeff Forshaw.

Static binary rewriting is important

Applications

1. Software fault isolation (SFI) [WLAG93]
2. Control Flow Integrity (CFI) [ABEL09]
3. Binary code hardening (e.g.,STIR [WMHL12])
4. Binary code reuse (e.g.,BCR [CJMS10])
5. Platform-specific optimizations [ASE+13]

Challenges in disassembling

1. Recognizing and relocating static memory addresses
2. Handling dynamically computed memory addresses
3. Differentiating code from data
4. Handling function pointer arguments (e.g., callbacks)
5. Handing PIC (Position Independent Code)

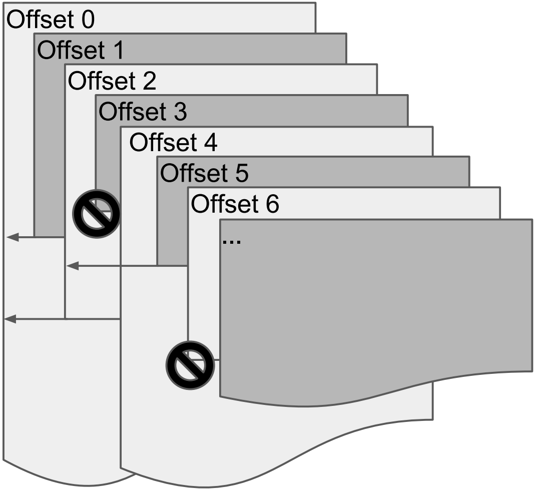
Existing static rewriters: w/ heuristics

1. Assume certain compiler generated binaries
2. Assume having debug symbols
3. Assume knowledge of APIs (call backs)
4. Assume no code and data interleaving
5. Rely on relocation metadata
6. Use heuristics to recognize static memory addresses
7. ...

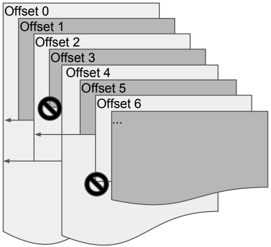
Brute Force Disassembler

"When in doubt, use brute force." – Ken Thompson

Brute Force Disassembler

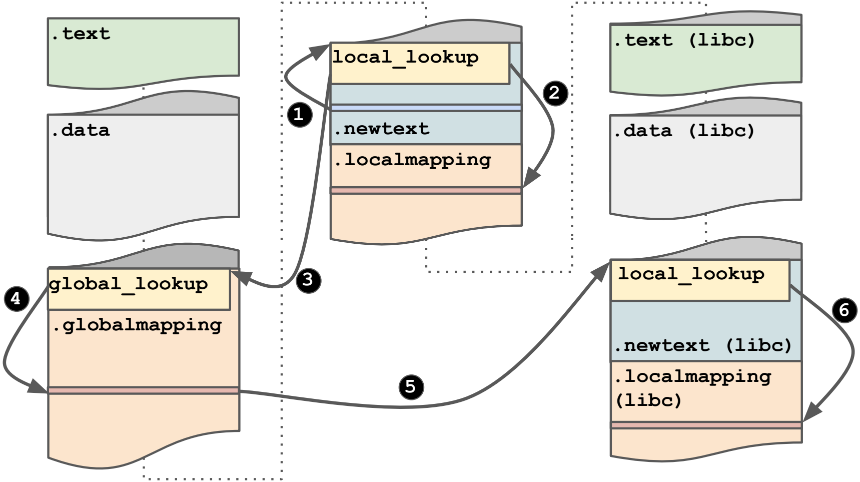


Brute Force Disassembler

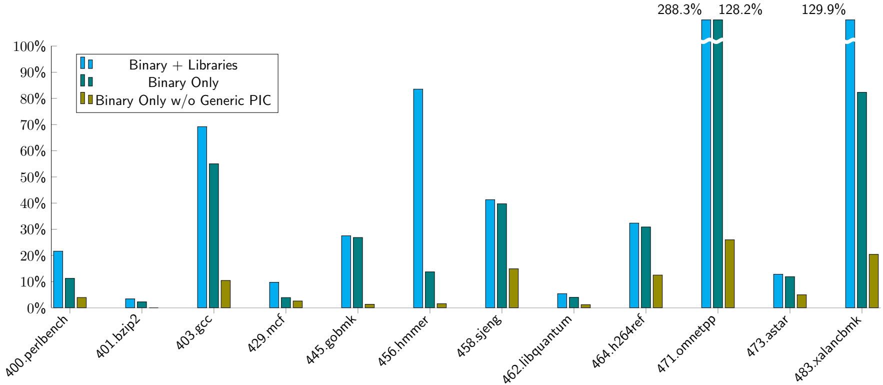


1. Statically Disassembly of Obfuscated Binaries [KRVV04]
2. Shingled Graph Disassembly [WZHK14]
3. GPU-Disasm: GPU-based x86 Disassembly [LVP+15]

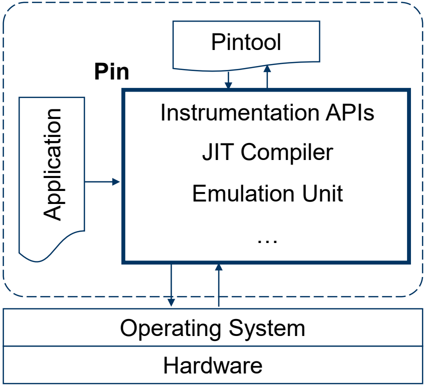
Instruction Address Mapping



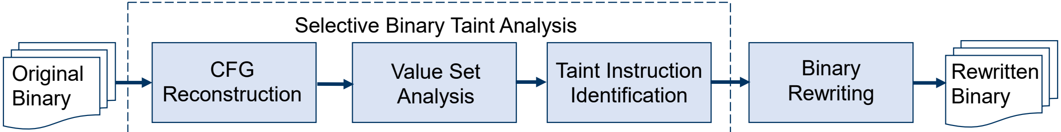
MultiVerse Overhead



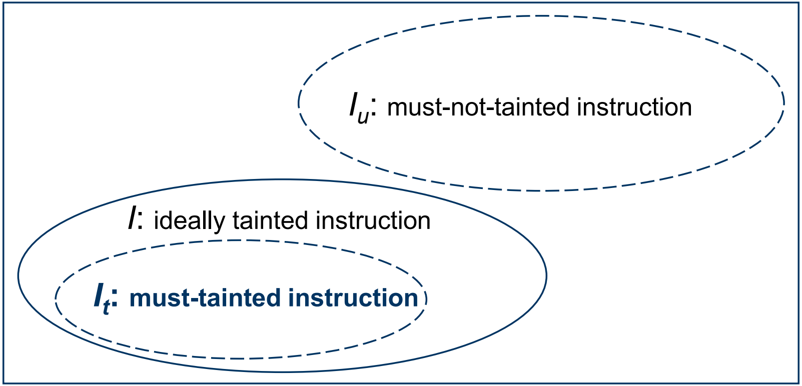
Limitations of Existing Dynamic Taint Analysis (e.g., libdft)



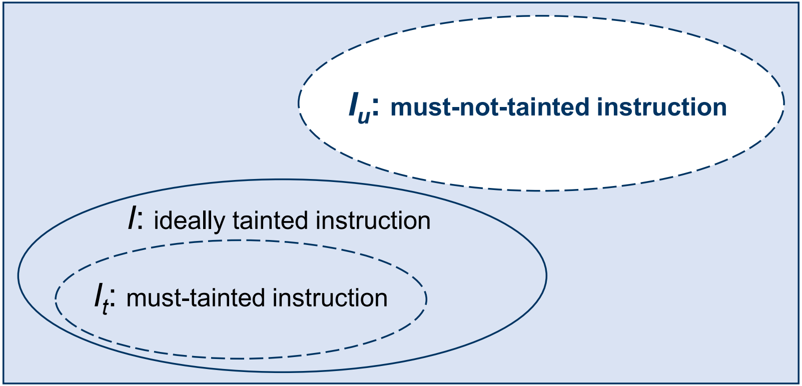
Design of SelectiveTaint



Essence of SelectiveTaint


I_u: must-not-tainted instruction
I: ideally tainted instruction
I_t: must-tainted instruction

Essence of SelectiveTaint


I_u: must-not-tainted instruction
I: ideally tainted instruction
I_t: must-tainted instruction

Taint Policy in SelectiveTaint

Unreachable instructions

Removed from must-not-tainted set

```
<version_etc_arn>:
804b7a0: push ebp
```

Potentially tainted instructions

Removed from must-not-tainted set

```
8055c3c: call 8048f30 <__IO_getc@plt>
8055c41: mov eax, edx
```

Untainted operand instructions

Added to must-not-tainted set

```
8096a07: inc ebp
```

None taint-propagation instructions

Added to must-not-tainted set

```
8062456: jmp 806238b <mbslen+0x8b>
```

Taint Policy in SelectiveTaint

Unreachable instructions

Removed from must-not-tainted set

```
<version_etc_arn>:
804b7a0: push ebp
```

Potentially tainted instructions

Removed from must-not-tainted set

```
8055c3c: call 8048f30 <__IO_getc@plt>
8055c41: mov eax, edx
```

Untainted operand instructions

Added to must-not-tainted set

```
8096a07: inc ebp
```

None taint-propagation instructions

Added to must-not-tainted set

```
8062456: jmp 806238b <mbslen+0x8b>
```

Taint Policy in SelectiveTaint

Unreachable instructions

Removed from must-not-tainted set

```
<version_etc_arn>:
804b7a0: push ebp
```

Potentially tainted instructions

Removed from must-not-tainted set

```
8055c3c: call 8048f30 <__IO_getc@plt>
8055c41: mov eax, edx
```

Untainted operand instructions

Added to must-not-tainted set

```
8096a07: inc ebp
```

None taint-propagation instructions

Added to must-not-tainted set

```
8062456: jmp 806238b <mbslen+0x8b>
```

Taint Policy in SelectiveTaint

Unreachable instructions

Removed from must-not-tainted set

```
<version_etc_arn>:
804b7a0: push ebp
```

Potentially tainted instructions

Removed from must-not-tainted set

```
8055c3c: call 8048f30 <__IO_getc@plt>
8055c41: mov eax, edx
```

Untainted operand instructions

Added to must-not-tainted set

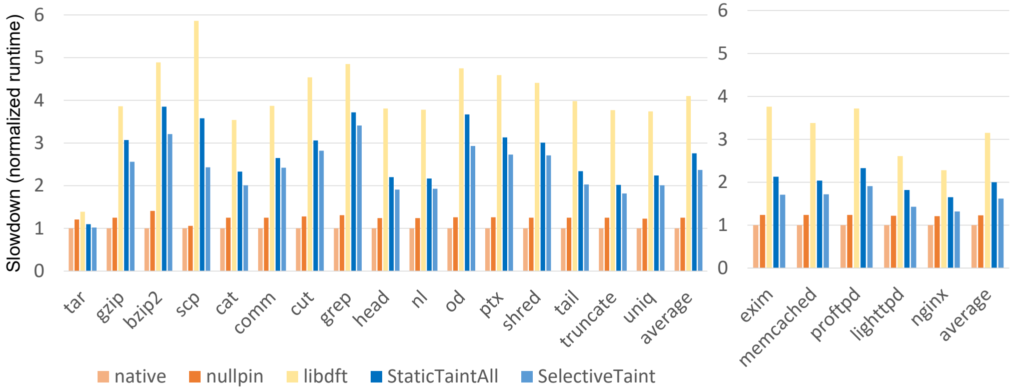
```
8096a07: inc ebp
```

None taint-propagation instructions

Added to must-not-tainted set

```
8062456: jmp 806238b <mbslen+0x8b>
```

Performance Overhead of SelectiveTaint



Exploit Detection w/ SelectiveTaint

Program	Category	Vulnerability	CVE ID	StaticTaintAll	SelectiveTaint
SoX 14.4.2	Sound Processing Utilities	Buffer Overflow	CVE-2019-8356	✓	✓
TinTin++ 2.01.6	Multiplayer Online Game Client	Buffer Overflow	CVE-2019-7629	✓	✓
dcraw 9.28	Raw Image Decoder	Buffer Overflow	CVE-2018-19655	✓	✓
ngiflib 0.4	GIF Format Decoding Library	Buffer Overflow	CVE-2018-11575	✓	✓
Gravity 0.3.5	Programming Language Interpreter	Buffer Overflow	CVE-2017-1000437	✓	✓
MP3Gain 1.5.2	Audio Normalization Software	Buffer Overflow	CVE-2017-14411	✓	✓
NASM 2.14.02	Assembler and Disassembler	Double Free	CVE-2019-8343	✓	✓
Jhead 3.00	Exif Jpeg Header Manipulation Tool	Integer Underflow	CVE-2018-6612	✓	✓
nginx 1.4.0	Web Server	Buffer Overflow	CVE-2013-2028	✓	✓

Ongoing Work: Enabling Dynamic Analysis of IoT Firmware

Goal: Firmware used in embedded devices (e.g., Bluetooth devices) with microcontroller (MCU) is everywhere, but they are weak in security (or no security feature supported). How to analyze and detect vulnerabilities among them.

Firmware Fuzzing

1. Peripheral registers are directly mapped to memory
2. The operations are hidden to the firmware code
3. How to analyze the mapping and enable the dynamic analysis
4. How to fuzz the firmware to identify the vulnerabilities.

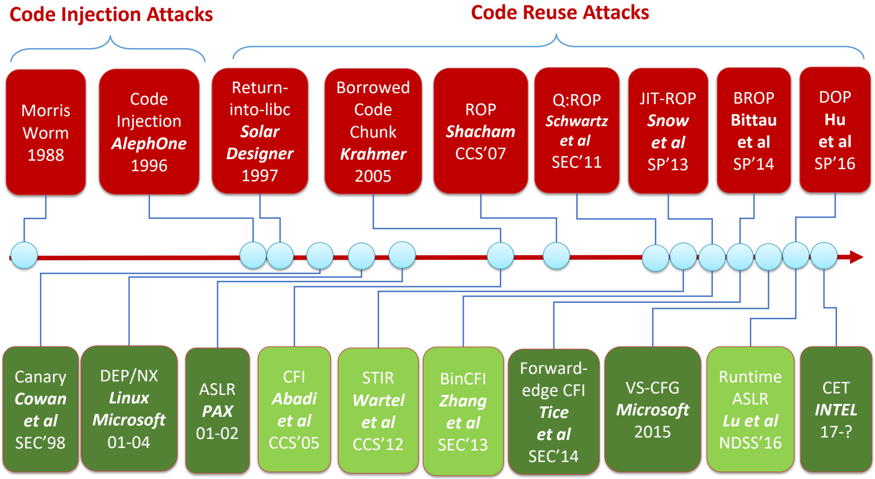
Future Work

Vision: Decompiled code should also contain the meaning of the variables, and functions, as what they were named by the developers.

Next Generation Decompilation

1. Recognizing the semantics of variables, functions, and naming them properly in the decompiled code
2. Recognizing macro and template code
3. Enabling program analysis with decompiled code, e.g., symbolic execution (KLEE)
4. Detecting logic vulnerabilities

The Arm Race Between Offense and Defense



Code Injection Attacks

Code Reuse Attacks

Morris Worm 1988

Code Injection ***AlephOne*** 1996

Return-into-libc ***Solar Designer*** 1997

Borrowed Code Chunk ***Krahmer*** 2005

ROP ***Shacham*** CCS'07

Q:ROP ***Schwartz et al*** SEC'11

JIT-ROP ***Snow et al*** SP'13

BROP **Bittau et al** SP'14

DOP **Hu et al** SP'16

Canary ***Cowan et al*** SEC'98

DEP/NX ***Linux Microsoft*** 01-04

ASLR ***PAX*** 01-02

CFI ***Abadi et al*** CCS'05

STIR ***Wartel et al*** CCS'12

BinCFI ***Zhang et al*** SEC'13

Forward-edge CFI ***Tice et al*** SEC'14

VS-CFG ***Microsoft*** 2015

Runtime ASLR ***Lu et al*** NDSS'16

CET ***INTEL*** 17-?

Other Practical Defenses

When Developing New Software

1. Turnning on compiler flag
   - Canary (/gs, -fstack-protector)
   - CFI (/cfg, -fsanitize=cfi-icall)
   - CET (gcc 8.0), PAC (clang/gcc)
   - ASLR (-pie -fPIE)
2. Using type safe language
   - Rust

Other Practical Defenses

When Developing New Software

1. Turnning on compiler flag
   - Canary (/gs, -fstack-protector)
   - CFI (/cfg, -fsanitize=cfi-icall)
   - CET (gcc 8.0), PAC (clang/gcc)
   - ASLR (-pie -fPIE)
2. Using type safe language
   - Rust

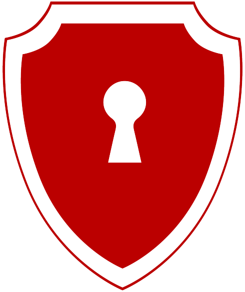
When Deploying Old Software

1. Turnning on kernel DEP
2. Turnning on ASLR
   - Load-time, and Microsoft EMET
3. Keeping software patched
4. Fuzzing to identify vulnerabilities
   - Rewriting (Taint, ASLR, CFI)

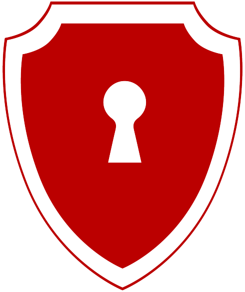
Our Recent Efforts

1. Superset Disassembly: Statically Rewriting x86 Binaries Without Heuristics. In NDSS 2018
2. SelectiveTaint: Efficient Data Flow Tracking With Static Binary Rewriting. In USENIX Security 2021

Our Key Research Thrusts and Interests



Our Key Research Thrusts and Interests



Key Research Thrusts

1. (**Why**) Understanding and discovering of **known** or new-emerging (**unknown**) vulnerabilities/attacks/-malware
2. (**How**) Developing automated algorithms, systems, and tools for analysis and defenses

Our Key Research Thrusts and Interests

Key Research Thrusts

1. (**Why**) Understanding and discovering of **known** or new-emerging (**unknown**) vulnerabilities/attacks/-malware
2. (**How**) Developing automated algorithms, systems, and tools for analysis and defenses

Current Interests

1. Systems security (e.g., **trusted computing**, virtualization, kernel)
2. Software security (e.g.,**binary** analysis, and **vulnerability** discovery)
3. Security in emerging platforms (e.g., AI, **IoT**, **automobile**).

Thank You

Software Security: Past, Present, and Future

Zhiqiang Lin

zlin@cse.ohio-state.edu

08/19/2021

References I

Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti, *Control-flow integrity principles, implementations, and applications*, ACM Trans. Information and System Security **13** (2009), no. 1.

Kapil Anand, Matthew Smithson, Khaled Elwazeer, Aparna Kotha, Jim Gruen, Nathan Giles, and Rajeev Barua, *A compiler-level intermediate representation based binary analysis and rewriting system*, Proceedings of the 8th ACM European Conference on Computer Systems, ACM, 2013, pp. 295–308.

Juan Caballero, Noah M. Johnson, Stephen McCamant, and Dawn Song, *Binary code extraction and interface identification for security applications*, NDSS, Feb. 2010.

Christopher Kruegel, William Robertson, Fredrik Valeur, and Giovanni Vigna, *Static disassembly of obfuscated binaries*, USENIX security Symposium, vol. 13, 2004, pp. 18–18.

Evangelos Ladakis, Giorgos Vasiliadis, Michalis Polychronakis, Sotiris Ioannidis, and Georgios Portokalidis, *Gpu-disasm: A gpu-based x86 disassembler*, International Information Security Conference, Springer, 2015, pp. 472–489.

Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham, *Efficient software-based fault isolation*, Proc. ACM Sym. Operating Systems Principles, 1993, pp. 203–216.

Richard Wartell, Vishwath Mohan, Kevin Hamlen, and Zhiqiang Lin, *Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code*, Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS'12) (Raleigh, NC), October 2012.

Richard Wartell, Yan Zhou, Kevin W Hamlen, and Murat Kantarcioglu, *Shingled graph disassembly: Finding the undecideable path*, Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2014, pp. 273–285.