OSY.SSI[2015][10]

Code injection : Counter-measures

Table of Contents

About counter-measures

About counter-measures

- **Fact:**

About counter-measures

- **Fact:** code injection happens every day

About counter-measures

- **Fact:** code injection happens every day
- **Corollary:**

About counter-measures

- **Fact:** code injection happens every day
- **Corollary:** if perfect solutions exist and are implemented, they are not correctly implemented

About counter-measures

- **Fact:** code injection happens every day
- **Corollary:** if perfect solutions exist and are implemented, they are not correctly implemented
- **Fact 2:**

About counter-measures

- **Fact:** code injection happens every day
- **Corollary:** if perfect solutions exist and are implemented, they are not correctly implemented
- **Fact 2:** buffer overflow vulnerabilities are disappearing

About counter-measures

- **Fact:** code injection happens every day
- **Corollary:** if perfect solutions exist and are implemented, they are not correctly implemented
- **Fact 2:** buffer overflow vulnerabilities are disappearing
- **Corollary:**

About counter-measures

- **Fact:** code injection happens every day
- **Corollary:** if perfect solutions exist and are implemented, they are not correctly implemented
- **Fact 2:** buffer overflow vulnerabilities are disappearing
- **Corollary:** the situation on this specific issue is improving

About counter-measures

- **Fact:** code injection happens every day
- **Corollary:** if perfect solutions exist and are implemented, they are not correctly implemented
- **Fact 2:** buffer overflow vulnerabilities are disappearing
- **Corollary:** the situation on this specific issue is improving

Let's have a look at the counter-measures we mentioned

0. Not making vulnerable code in the first place

Stultus in tenebris ambulat

0. Not making vulnerable code in the first place

Stultus in tenebris ambulat

- Testing?

0. Not making vulnerable code in the first place

Stultus in tenebris ambulat

- Testing? (e.g. Valve)

  Necessarily incomplete and unsound, often inaccurate, misses advanced approaches

0. Not making vulnerable code in the first place

Stultus in tenebris ambulat

- Testing? (e.g. Valve)
  Necessarily incomplete and unsound, often inaccurate, misses advanced approaches
- Static analysis?

0. Not making vulnerable code in the first place

Stultus in tenebris ambulat

- Testing? (e.g. Valve)

  Necessarily incomplete and unsound, often inaccurate, misses advanced approaches

- Static analysis? (e.g. Airbus)

  Provably complete but not sound, language specific, expensive, time-consuming, expert-driven, requires fine-tuning

0. Not making vulnerable code in the first place

Stultus in tenebris ambulat

- Testing? (e.g. Valve)
  Necessarily incomplete and unsound, often inaccurate, misses advanced approaches
- Static analysis? (e.g. Airbus)
  Provably complete but not sound, language specific, expensive, time-consuming, expert-driven, requires fine-tuning
- “Memory-safe language”?

0. Not making vulnerable code in the first place

Stultus in tenebris ambulat

- Testing? (e.g. Valve)
  Necessarily incomplete and unsound, often inaccurate, misses advanced approaches
- Static analysis? (e.g. Airbus)
  Provably complete but not sound, language specific, expensive, time-consuming, expert-driven, requires fine-tuning
- “Memory-safe language”? (e.g. RATP)
  Not always possible, not always safe from these attacks... see e.g. Python’s CVE-2008-5031

0. Not making vulnerable code in the first place

Stultus in tenebris ambulat

- Testing? (e.g. Valve)
  Necessarily incomplete and unsound, often inaccurate, misses advanced approaches
- Static analysis? (e.g. Airbus)
  Provably complete but not sound, language specific, expensive, time-consuming, expert-driven, requires fine-tuning
- “Memory-safe language”? (e.g. RATP)
  Not always possible, not always safe from these attacks... see e.g. Python’s CVE-2008-5031
- Coding guidelines?

0. Not making vulnerable code in the first place

Stultus in tenebris ambulat

- Testing? (e.g. Valve)
  Necessarily incomplete and unsound, often inaccurate, misses advanced approaches
- Static analysis? (e.g. Airbus)
  Provably complete but not sound, language specific, expensive, time-consuming, expert-driven, requires fine-tuning
- "Memory-safe language"? (e.g. RATP)
  Not always possible, not always safe from these attacks... see e.g. Python's CVE-2008-5031
- Coding guidelines? (e.g. Lockheed)
  Language specific, sensitive to change in needs and tools, may result in unsafe alternatives to forbidden primitives... e.g. rewriting `malloc`

0. Not making vulnerable code in the first place

Stultus in tenebris ambulat

- Testing? (e.g. Valve)
  Necessarily incomplete and unsound, often inaccurate, misses advanced approaches
- Static analysis? (e.g. Airbus)
  Provably complete but not sound, language specific, expensive, time-consuming, expert-driven, requires fine-tuning
- "Memory-safe language"? (e.g. RATP)
  Not always possible, not always safe from these attacks... see e.g. Python's CVE-2008-5031
- Coding guidelines? (e.g. Lockheed)
  Language specific, sensitive to change in needs and tools, may result in unsafe alternatives to forbidden primitives... e.g. rewriting `malloc`

Coders must be aware of, and alert to memory.

That being said: testing would prevent most bugs (USENIX'14), static analysis works (A380), etc. That's not enough, but that's improvement.

1. Detecting stack smashing and blocking it

No free lunch theorem

1. Detecting stack smashing and blocking it

No free lunch theorem

- Segmentation (e.g. x86)

  Efficient, but too coarse, doesn't work at kernel level, removed from x86-64.

1. Detecting stack smashing and blocking it

No free lunch theorem

- Segmentation (e.g. x86)

  Efficient, but too coarse, doesn't work at kernel level, removed from x86-64.
- Canaries (e.g. ProPolice)

  Efficient, but hard to implement (Vista), bypassable.

1. Detecting stack smashing and blocking it

No free lunch theorem

- Segmentation (e.g. x86)

  Efficient, but too coarse, doesn't work at kernel level, removed from x86-64.
- Canaries (e.g. ProPolice)

  Efficient, but hard to implement (Vista), bypassable.
- SEH, SEHOP (e.g. Win7, Win8)

  Large memory and CPU overhead, abuseable and abused.

1. Detecting stack smashing and blocking it

No free lunch theorem

- Segmentation (e.g. x86)

  Efficient, but too coarse, doesn't work at kernel level, removed from x86-64.
- Canaries (e.g. ProPolice)

  Efficient, but hard to implement (Vista), bypassable.
- SEH, SEHOP (e.g. Win7, Win8)

  Large memory and CPU overhead, abuseable and abused.

Also, when detection happens, it might be too late...

1. Detecting stack smashing and blocking it

No free lunch theorem

- Segmentation (e.g. x86)

  Efficient, but too coarse, doesn't work at kernel level, removed from x86-64.
- Canaries (e.g. ProPolice)

  Efficient, but hard to implement (Vista), bypassable.
- SEH, SEHOP (e.g. Win7, Win8)

  Large memory and CPU overhead, abuseable and abused.

Also, when detection happens, it might be too late...
Note: we didn't deactivate that during last session...

2. Analysing user actions

Malleus maleficarum, maleficas, & earum hæresim, ut phramea potentissima conterens.

2. Analysing user actions

Malleus maleficarum, maleficas, & earum hæresim, ut phramea potentissima conterens.

- Antivirus (e.g. ClamAV) ?

2. Analysing user actions

Malleus maleficarum, maleficas, & earum hæresim, ut phramea potentissima conterens.

- Antivirus (e.g. ClamAV) ?

  Signature-based A/V can't deal with modern threats, memory-inefficient, bypassable, abuseable (see e.g. Kaspersky).

2. Analysing user actions

Malleus maleficarum, maleficas, & earum hæresim, ut phramea potentissima conterens.

- Antivirus (e.g. ClamAV) ?

  Signature-based A/V can't deal with modern threats, memory-inefficient, bypassable, abuseable (see e.g. Kaspersky).
- IDS (e.g. Snort) ?

2. Analysing user actions

Malleus maleficarum, maleficas, & earum hæresim, ut phramea potentissima conterens.

- Antivirus (e.g. ClamAV) ?

  Signature-based A/V can't deal with modern threats, memory-inefficient, bypassable, abuseable (see e.g. Kaspersky).

- IDS (e.g. Snort) ?

  Neither sound nor complete, bound to the specificity/sensitivity trade-off, often filter-based, bypassable.

2. Analysing user actions

Malleus maleficarum, maleficas, & earum hæresim, ut phramea potentissima conterens.

- Antivirus (e.g. ClamAV) ?

  Signature-based A/V can't deal with modern threats, memory-inefficient, bypassable, abuseable (see e.g. Kaspersky).
- IDS (e.g. Snort) ?

  Neither sound nor complete, bound to the specificity/sensitivity trade-off, often filter-based, bypassable.
- Firewalls ?

2. Analysing user actions

Malleus maleficarum, maleficas, & earum hæresim, ut phramea potentissima conterens.

- Antivirus (e.g. ClamAV) ?

  Signature-based A/V can't deal with modern threats, memory-inefficient, bypassable, abuseable (see e.g. Kaspersky).
- IDS (e.g. Snort) ?

  Neither sound nor complete, bound to the specificity/sensitivity trade-off, often filter-based, bypassable.
- Firewalls ?

  not directly a defence against this type of attacks – sometimes a victim of it.

2. Analysing user actions

Malleus maleficarum, maleficas, & earum hæresim, ut phramea potentissima conterens.

- Antivirus (e.g. ClamAV) ?

  Signature-based A/V can't deal with modern threats, memory-inefficient, bypassable, abuseable (see e.g. Kaspersky).
- IDS (e.g. Snort) ?

  Neither sound nor complete, bound to the specificity/sensitivity trade-off, often filter-based, bypassable.
- Firewalls ?

  not directly a defence against this type of attacks – sometimes a victim of it.

How would you bypass an IDS? How would you bypass an A/V?

2. Analysing user actions

Bypassing an IDS

What is this?

2. Analysing user actions

Bypassing an IDS

What is this?

```
SUPREMELYCRUTCHESCATARACTINSTRUMENTATIONLOVABLY
PERILLABARBSPANISHIZESBEGANAMBIDEXTROUSLY
PHOSPHORSAVEDZEALOUSCONVINCEDFIXERS
```

3. Refusing unauthorised code and stack execution

Leeloo Dallas Multipass

- Signed code?

3. Refusing unauthorised code and stack execution

Leeloo Dallas Multipass

- Signed code? (e.g. TPM)

  Who checks the signatures? Also, ROP.

3. Refusing unauthorised code and stack execution

Leeloo Dallas Multipass

- Signed code? (e.g. TPM)

  Who checks the signatures? Also, ROP.
- Non-executable stack (e.g. NX, DEP) ?

3. Refusing unauthorised code and stack execution

Leeloo Dallas Multipass

- Signed code? (e.g. TPM)

  Who checks the signatures? Also, ROP.
- Non-executable stack (e.g. NX, DEP) ?

  Breaks things. Also, ROP and UAF.

3. Refusing unauthorised code and stack execution

Leeloo Dallas Multipass

- Signed code? (e.g. TPM)

  Who checks the signatures? Also, ROP.
- Non-executable stack (e.g. NX, DEP) ?

  Breaks things. Also, ROP and UAF.

Because of DEP, attack vectors have shifted to JITs and browser components: Java, Flash, Video.

4. Shuffling the memory model

"What is important is to spread confusion, not eliminate it."

- ASCII-Armoured Addresses?

4. Shuffling the memory model

"What is important is to spread confusion, not eliminate it."

- ASCII-Armoured Addresses?

  Makes it harder, but not impossible, to attack from strings. Not always possible.

  Heavily restrict memory.

4. Shuffling the memory model

"What is important is to spread confusion, not eliminate it."

- ASCII-Armoured Addresses?

  Makes it harder, but not impossible, to attack from strings. Not always possible.

  Heavily restrict memory.
- ASLR?

4. Shuffling the memory model

"What is important is to spread confusion, not eliminate it."

- ASCII-Armoured Addresses?

  Makes it harder, but not impossible, to attack from strings. Not always possible. Heavily restrict memory.

- ASLR?

  Hard to implement correctly (Vista, Win7, Win8, FreeBSD), inefficient under 64 bits, breaks things. Also, ROP and UAF.

4. Shuffling the memory model

"What is important is to spread confusion, not eliminate it."

- ASCII-Armoured Addresses?

  Makes it harder, but not impossible, to attack from strings. Not always possible. Heavily restrict memory.
- ASLR?

  Hard to implement correctly (Vista, Win7, Win8, FreeBSD), inefficient under 64 bits, breaks things. Also, ROP and UAF.
- PIE?

4. Shuffling the memory model

"What is important is to spread confusion, not eliminate it."

- ASCII-Armoured Addresses?

  Makes it harder, but not impossible, to attack from strings. Not always possible. Heavily restrict memory.

- ASLR?

  Hard to implement correctly (Vista, Win7, Win8, FreeBSD), inefficient under 64 bits, breaks things. Also, ROP and UAF.

- PIE?

  Breaks things, ROP unless kernel is also PIE. Not immune to UAF.

By the way, where do we find bofs?

By the way, where do we find bofs?

- Software code

By the way, where do we find bofs?

- Software code
- Libraries, shared objects

By the way, where do we find bofs?

- Software code
- Libraries, shared objects
- Firmware, drivers, BIOS

By the way, where do we find bofs?

- Software code
- Libraries, shared objects
- Firmware, drivers, BIOS
- OS Kernel

By the way, where do we find bofs?

- Software code
- Libraries, shared objects
- Firmware, drivers, BIOS
- OS Kernel

...i.e. everywhere.

By the way, where do we find bofs?

- Software code
- Libraries, shared objects
- Firmware, drivers, BIOS
- OS Kernel

...i.e. everywhere.

Remember: only one suffices.

So... what?

So... what?

- Efficiency/Overhead trade-off

So... what?

- Efficiency/Overhead trade-off
- Soundness/Completeness trade-off

So... what?

- Efficiency/Overhead trade-off
- Soundness/Completeness trade-off
- Sensibility/Sensitivity trade-off

So... what?

- Efficiency/Overhead trade-off
- Soundness/Completeness trade-off
- Sensibility/Sensitivity trade-off

To this day, no solution has proven 100% foolproof at the OS level.

So... what?

- Efficiency/Overhead trade-off
- Soundness/Completeness trade-off
- Sensibility/Sensitivity trade-off

To this day, no solution has proven 100% foolproof at the OS level.
And they all come at a cost.

But then...

But then...

But then people went stupid.

But then...

But then people went stupid.

- Re-implementation of unsafe “OS-like” environments in browsers (JavaScript...)

But then...

But then people went stupid.

- Re-implementation of unsafe “OS-like” environments in browsers (JavaScript...)
- Remote access to kernel-controled hardware (WebGL...)

But then...

But then people went stupid.

- Re-implementation of unsafe “OS-like” environments in browsers (JavaScript...)
- Remote access to kernel-controled hardware (WebGL...)
- De-sandboxing of user-provided content (HTML5 video...)

But then...

But then people went stupid.

- Re-implementation of unsafe “OS-like” environments in browsers (JavaScript...)
- Remote access to kernel-controled hardware (WebGL...)
- De-sandboxing of user-provided content (HTML5 video...)
- Wearable devices with critical function (pacemakers...)

But then...

But then people went stupid.

- Re-implementation of unsafe “OS-like” environments in browsers (JavaScript...)
- Remote access to kernel-controled hardware (WebGL...)
- De-sandboxing of user-provided content (HTML5 video...)
- Wearable devices with critical function (pacemakers...)

Why?

Why? Economics, hidden

Why? Economics, hidden

- Incentive to provide new products (novelty effect)

Why? Economics, hidden

- Incentive to provide new products (novelty effect)
- No significant breakthrough in $\sim$20 years

Why? Economics, hidden

- Incentive to provide new products (novelty effect)
- No significant breakthrough in $\sim$20 years
- Concerns are with design, and performance, not security (visible vs. invisible)

Why? Economics, hidden

- Incentive to provide new products (novelty effect)
- No significant breakthrough in $\sim$20 years
- Concerns are with design, and performance, not security (visible vs. invisible)
- Low awareness of low-level concerns

Why? Economics, hidden

- Incentive to provide new products (novelty effect)
- No significant breakthrough in $\sim$20 years
- Concerns are with design, and performance, not security (visible vs. invisible)
- Low awareness of low-level concerns

Remember: security in products is an economic externality.

Table of Contents

What do I mean “advanced injection”

and why do I talk about it

What do I mean “advanced injection”

and why do I talk about it

In recent years, a surge of *new, stealthier, more elaborate attacks* has taken over.

What do I mean “advanced injection”

and why do I talk about it

In recent years, a surge of *new, stealthier, more elaborate attacks* has taken over.

- Corresponds to a shift in the cybercriminal population

What do I mean “advanced injection”

and why do I talk about it

In recent years, a surge of *new, stealthier, more elaborate attacks* has taken over.

- Corresponds to a shift in the cybercriminal population
- Coincides with widespread superficial use of computers by untrained personnel

What do I mean “advanced injection”

and why do I talk about it

In recent years, a surge of *new, stealthier, more elaborate attacks* has taken over.

- Corresponds to a shift in the cybercriminal population
- Coincides with widespread superficial use of computers by untrained personnel
- Manages to bypass protection and evade detection

So there we go again: Vote

Yay again

Would you rather:

./0 Try some of these techniques yourself? (more coding)

./1 Learn about how modern attacks are planned? (less coding)

You'll have the slides and code either way to practice at home.

Also don't forget to vote for the exam

More training

`root-me.org`
`smashthestack.org`