Enabling Client-Side Crash-Resistance to Overcome Diversification and Information Hiding

Robert Gawlik, Benjamin Kollenda, Philipp Koppe, Behrad Garmany, Thorsten Holz

Ruhr University Bochum
Horst Görrtz Institute for IT-Security
Bochum, Germany
Crash-Resistance
char* addr = 0;

void crash(){
    addr++;
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main(){
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1){
        GetMessage(&msg, NULL, 0, 0);
        DispatchMessage(&msg);
    }
}
char* addr = 0;

void crash()
{
    addr++;
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main()
{
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1)
    {
        GetMessage(&msg, NULL, 0, 0);
        DispatchMessage(&msg);
    }
}
char* addr = 0;

void crash(){
    addr++;
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main(){
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1){
        GetMessage(&msg, NULL, 0, 0);
        DispatchMessage(&msg);
    }
}
Crash-Resistance

char* addr = 0;

void crash()
{
    addr++;
    printf("reading \%x", addr);
    char content = *(addr);
    printf("read done");
}

int main()
{
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1)
    {
        GetMessage(&msg, NULL, 0, 0);
        DispatchMessage(&msg);
    }
}

• Set timer callback crash()
• Dispatch crash() each ms
• crash() generates a fault on first execution
char* addr = 0;

void crash()
{
    addr++;
    printf("reading %x" addr);
    char content = *(addr);
    printf("read done");
}

int main()
{
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1){
        GetMessage(&msg, NULL, 0, 0);
        DispatchMessage(&msg);
    }
}
Internet Explorer has stopped working

Windows can check online for a solution to the problem.

- Check online for a solution and close the program
- Close the program
- Debug the program

View problem details
We're Sorry

Firefox had a problem and crashed. We'll try to restore your tabs and windows when it restarts.

To help us diagnose and fix the problem, you can send us a crash report.

☑ Tell Mozilla about this crash so they can fix it

Details...

Add a comment (comments are publicly visible)

☐ Include the address of the page I was on

☐ Allow Mozilla to contact me about this report

Enter your email address here

Your crash report will be submitted before you quit or restart.

Restart Firefox  Quit Firefox
Crash-Resistance

char* addr = 0;

void crash(){
    addr++;
    printf("reading %x" addr);
    char content = *(addr);
    printf("read done");
}

int main(){
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1){
        GetMessage(&msg, NULL, 0, 0);
        DispatchMessage(&msg);
    }
}

- Set timer callback `crash()`
- Dispatch `crash()` each ms
- `crash()` generates a fault on first execution

Program should terminate abnormally
Crash-Resistance

```c
char* addr = 0;

void crash()
{
    addr++;
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main()
{
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1)
    {
        GetMessage(&msg, NULL, 0, 0);
        DispatchMessage(&msg);
    }
}
```

- Set timer callback `crash()`
- Dispatch `crash()` each ms
- `crash()` generates a fault on first execution

Instead:
Program runs endlessly
Crash-Resistance

• Set timer callback `crash()`
• Dispatch `crash()` each ms
• `crash()` generates a fault on first execution
char* addr = 0;

void crash(){
    addr++;
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main(){
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1){
        GetMessage(&msg, NULL, 0, DispatchMessage(&msg));
    }
    return
}

DispatchMessage:
    __try
    {
        crash()
    }
    __except(expr)
    {
    
    }

Crash-Resistance
Behind the Scenes
Crash-Resistance

char* addr = 0;

void crash()
{
    addr++;
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main()
{
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1)
    {
        GetMessage(&msg, NULL, 0, NULL);
        DispatchMessage(&msg);
    }
    return
}

DispatchMessage:
__try
{
    crash()
}
__except(expr)
{
}

Access violation
**Crash-Resistance**

```
char* addr = 0;

void crash()
{
    addr++;
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main()
{
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1)
    {
        GetMessage(&msg, NULL, 0, NULL);
        DispatchMessage(&msg);
    }
    return
}
```

**Behind the Scenes**

```
DispatchMessage:
    __try
    {
        crash();
    }
    __except(expr)
    {
        // Access violation
        expr returns 1
    }
```

**Crash-Resistance Behind the Scenes**

Access violation

expr returns 1

**RUB**
Crash-Resistance

char* addr = 0;

void crash(){
    addr++;
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main(){
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1){
        GetMessage(&msg, NULL, 0, 0);
        DispatchMessage(&msg);
    }
}

DispatchMessage:
__try
{
    crash()
}
__except(expr)
{
    execute handler

    expr returns 1

    Access violation
}

return

Crash-Resistance

Behind the Scenes
char* addr = 0;

void crash(){
    addr++;
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main(){
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1){
        GetMessage(&msg, NULL, 0,
        DispatchMessage(&msg);
    }
}

DispatchMessage:
    __try
    {
        crash()
    }
    __except(expr)
    {
        execute handler
    }
    expr returns 1
    continue execution
    return

Access violation
char* addr = 0;

void crash(){
    addr++;  
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main(){
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1){
        GetMessage(&msg, NULL, 0, 0);
        DispatchMessage(&msg);
    }
    return
}

DispatchMessage:
    __try
    {
        crash()
    }
    __except(expr)
    {
    }

Crash-Resistance
Behind the Scenes
char* addr = 0;

void crash(){
    addr++;
    printf("reading %x", addr);
    char content = *(addr);
    printf("read done");
}

int main(){
    MSG msg;
    SetTimer(0, 0, 1, crash);
    while(1){
        GetMessage(&msg, NULL, 0, 0);
        DispatchMessage(&msg);
    }
}
Crash-Resistance

char* addr = 0;

void crash()
{
  addr++;
  printf("reading %x", addr);
  char content = *(addr);
  printf("read done");
}

int main()
{
  MSG msg;
  SetTimer(0, 0, 1, crash);
  while(1)
  {
    GetMessage(&msg, NULL, 0, 0);
    DispatchMessage(&msg);
  }
}

If a fault is generated, execution is transferred to the end of the loop.

Program continues running despite producing faults.
Crash-Resistance

Behind the Scenes

If a fault is generated, execution is transferred to the end of the loop.
Client-Side Crash-Resistance
Client-Side Crash-Resistance

• *Server* applications respawn upon abnormal termination
Client-Side Crash-Resistance

- **Server** applications respawn upon abnormal termination
  
  → Attacks: ASLR de-randomization [1]; Hacking Blind [2]; Missing the Point(er) [3]
Client-Side Crash-Resistance

- **Server** applications respawn upon abnormal termination
  
  → Attacks: ASLR de-randomization [1]; Hacking Blind [2]; Missing the Point(er) [3]

- **Client programs** do not restart upon a crash (e.g., web browsers)
Client-Side Crash-Resistance

• **Server** applications respawn upon abnormal termination

→ Attacks: ASLR de-randomization [1]; Hacking Blind [2]; Missing the Point(er) [3]

• **Client programs** do not restart upon a crash (e.g., web browsers)

• Crash-resistant code constructs are available in browsers
Client-Side Crash-Resistance

- **Server** applications respawn upon abnormal termination

  → Attacks: ASLR de-randomization [1]; Hacking Blind [2]; Missing the Point(er) [3]

- **Client programs** do not restart upon a crash (e.g., **web browsers**)

- Crash-resistant code constructs are **available** in browsers

- Crash-resistant code **prevents abnormal termination** of browsers
Client-Side Crash-Resistance

- **Server** applications respawn upon abnormal termination
  → Attacks: ASLR de-randomization [1]; Hacking Blind [2]; Missing the Point(er) [3]

- **Client programs** do not restart upon a crash (e.g., *web browsers*)
- Crash-resistant code constructs are *available* in browsers
- Crash-resistant code *prevents abnormal termination* of browsers
- It is possible to access memory *more than once* with wrong permissions
Client-Side Crash-Resistance

• *Server* applications respawn upon abnormal termination

  → Attacks: ASLR de-randomization [1]; Hacking Blind [2]; Missing the Point(er) [3]

• *Client programs* do not restart upon a crash (e.g., web browsers)

• Crash-resistant code constructs are *available* in browsers

• Crash-resistant code *prevents abnormal termination* of browsers

• It is possible to access memory *more than once* with wrong permissions

  → Client-Side Crash-Resistance is usable as an *attack primitive*
Attacks with Client-Side Crash-Resistance
Memory Oracles with JavaScript

• Vulnerability needed to read/write address space
Memory Oracles with JavaScript

• Vulnerability needed to read/write address space

(1) Use crash-resistance primitive to try reading attacker-set *address*
Memory Oracles with JavaScript

• Vulnerability needed to read/write address space

(1) Use crash-resistance primitive to try reading attacker-set address

(2) Recognize if read succeeds or fails
Memory Oracles with JavaScript

• Vulnerability needed to read/write address space

(1) Use crash-resistance primitive to try reading attacker-set address

(2) Recognize if read succeeds or fails

→ If address is readable, content is returned into JavaScript variable
Memory Oracles with JavaScript

- Vulnerability needed to read/write address space

1. Use crash-resistance primitive to try reading attacker-set *address*

2. Recognize if read succeeds or fails

→ If *address* is readable, *content* is returned into JavaScript variable

→ On a fault, reset *address* and try reading again
Memory Oracles with JavaScript

Memory Oracle in Internet Explorer (32-bit)

- `setInterval()` in web worker is crash-resistant
- `callback` function set with `setInterval()` queries memory
- $\approx 63$ probes/s
**Memory Oracle in Internet Explorer (32-bit)**

- `setInterval()` in web worker is crash-resistant
- `callback` function set with `setInterval()` queries memory
- $\approx 63$ probes/s

**Memory Oracle in Mozilla Firefox (64-bit)**

- `asm.js` uses exception handling for certain memory accesses
- Modification of `metadata` allows crash-resistant memory queries
- $\approx 700$ probes/s (Windows)
- $\approx 18,000$ probes/s (Linux)
Unveiling reference-less hidden memory regions

- Memory region is randomized by ASLR
- No references exist to memory region

First program run

Address space

| :readable memory | :nonreadable memory | :hidden memory |
Unveiling reference-less hidden memory regions

- memory region is randomized by ASLR
- no references exist to memory region

Second program run

Address space

- : readable memory
- : nonreadable memory
- : hidden memory
Crash-Resistant Memory Scanning

*Unveiling reference-less hidden memory regions*

- Use memory oracles to *probe* address space
Crash-Resistant Memory Scanning

*Unveiling reference-less hidden memory regions*

- Use memory oracles to *probe* address space
- Discover *readable* addresses
Crash-Resistant Memory Scanning

*Unveiling reference-less hidden memory regions*

- Use memory oracles to *probe* address space
- Discover *readable* addresses
- Read memory and verify that *discovered memory* is structured in the same way as *hidden region*
Unveiling reference-less hidden memory regions

- Use memory oracles to *probe* address space
- Discover *readable* addresses
- Read memory and verify that *discovered memory* is structured in the same way as *hidden region*

→ Discovery of *sensitive data* helpful for adversary to mount subsequent attacks
Unveiling reference-less hidden memory regions

- Use memory oracles to *probe* address space
- Discover *readable* addresses
- Read memory and verify that *discovered memory* is structured in the same way as *hidden region*

→ Discovery of *sensitive data* helpful for adversary to mount subsequent attacks
  - TEB: ≈ 1min (Windows 32-bit)
Crash-Resistant Memory Scanning

Unveiling reference-less hidden memory regions

• Use memory oracles to *probe* address space

• Discover *readable* addresses

• Read memory and verify that *discovered memory* is structured in the same way as *hidden region*

→ Discovery of *sensitive data* helpful for adversary to mount subsequent attacks
  • TEB: \(\approx 1\)min (Windows 32-bit)
  • Pointer protection metadata: < 1s (Linux 64-bit)
Crash-Resistant Memory Scanning

*Overcome hidden code and code re-randomization*

- Scan memory to discover *data regions* that contain function pointers
Crash-Resistant Memory Scanning

Overcome hidden code and code re-randomization

• Scan memory to discover *data regions* that contain function *pointers*

• Resolve available function *pointers* within discovered data
Crash-Resistant Memory Scanning

Overcome hidden code and code re-randomization

- Scan memory to discover *data regions* that contain function *pointers*
- Resolve available function *pointers* within discovered data

There was no Control Flow Hijacking involved yet!
Crash-Resistant Memory Scanning

Overcome hidden code and code re-randomization

• Scan memory to discover *data regions* that contain function *pointers*

• Resolve available function *pointers* within discovered data

There was no Control Flow Hijacking involved yet!

To mount a control flow hijacking attack, perform *whole function code reuse*
Crash-Resistant Oriented Programming (CROP)
• Crash-resistant primitive (Internet Explorer) catches *execution* violations
• Crash-resistant primitive (Internet Explorer) catches execution violations

1. Prepare attacker controlled memory with parameters and exported system call
• Crash-resistant primitive (Internet Explorer) catches *execution* violations

(1) Prepare attacker controlled memory with *parameters* and *exported system call*

(2) Set *return address* for system call to *NULL* in controlled memory
• Crash-resistant primitive (Internet Explorer) catches execution violations

(1) Prepare attacker controlled memory with parameters and exported system call

(2) Set return address for system call to NULL in controlled memory

(3) Use control flow hijacking to dispatch system call on indirect call site in crash-resistant mode
• Crash-resistant primitive (Internet Explorer) catches execution violations

(1) Prepare attacker controlled memory with parameters and exported system call

(2) Set return address for system call to NULL in controlled memory

(3) Use control flow hijacking to dispatch system call on indirect call site in crash-resistant mode

(4) Read return data of system call and proceed to step (1)
Conclusion
Conclusion

- Browsers can indeed operate in crash-resistant mode despite having a hard crash-policy
Conclusion

• Browsers can indeed operate in *crash-resistant* mode despite having a *hard crash-policy*

• Complete memory scanning is possible in *client programs*, previously it was only shown for server applications
Conclusion

• Browsers can indeed operate in crash-resistant mode despite having a hard crash-policy

• Complete memory scanning is possible in client programs, previously it was only shown for server applications

• Client-Side Crash-Resistance weakens defenses based on hiding and diversification
Conclusion

• Browsers can indeed operate in \textit{crash-resistant} mode despite having a \textit{hard crash-policy}

• Complete memory scanning is possible in \textit{client programs}, previously it was only shown for server applications

• Client-Side Crash-Resistance \textit{weakens} defenses based on \textit{hiding} and \textit{diversification}

• Correct exception handling can \textit{prevent} Crash-Resistance
  • Bug 1135903 (Mozilla Firefox) [5]
Conclusion

• Browsers can indeed operate in crash-resistant mode despite having a hard crash-policy.

• Complete memory scanning is possible in client programs, previously it was only shown for server applications.

• Client-Side Crash-Resistance weakens defenses based on hiding and diversification.

• Correct exception handling can prevent Crash-Resistance:
  • Bug 1135903 (Mozilla Firefox) [5]

• Defenses that prevent memory corruption vulnerabilities, can prevent current crash-resistance primitives.
References

[1] Shacham et al. **On the effectiveness of address-space randomization.** CCS 2004


