SELF-DEFENSELESS

EUSKALHACK IV

silent signal

BÁLINT VARGA-PERKE 2019.06.22

WHOAMI

- Silent Signal co-founder
  - Penetration testing
  - Custom training
  - Consulting
- @buherator
  - Top Hungarian IT-sec resource for some time…
  - Moved to polluting the tubes via Twitter

BACKGROUND

- Some hits
  - Aruba wIPS
  - Panda cloud infrastructure
  - Bitdefender
  - Symantec Critical System Protection
  - Trend Micro Office Scan
  - McAfee crapware
- All logic bugs
- Tried fuzzing too
  - Not really my game…

PREVIOUS RESEARCH

silent signal

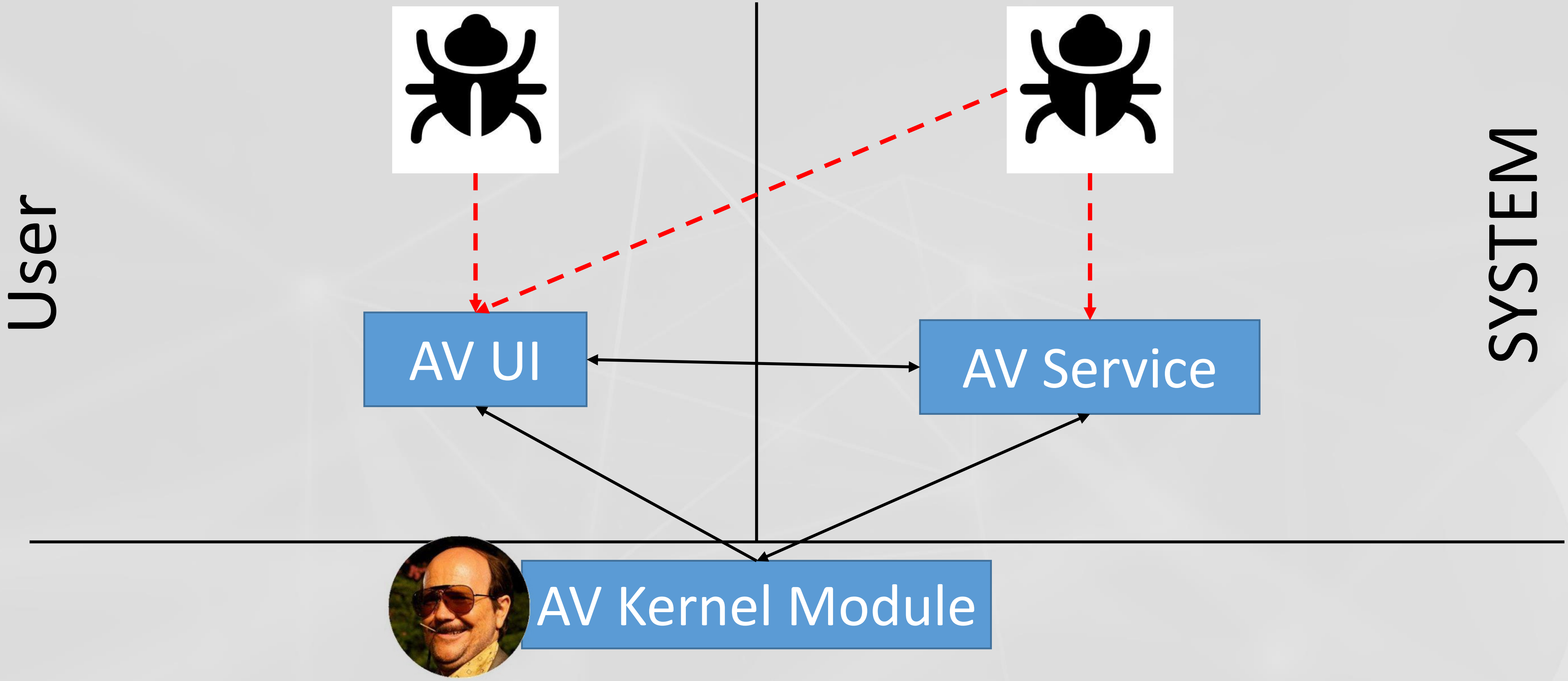
ABUSING PRIVILEGED FILE ACCESS IN ANTIVIRUS SOFTWARE

- Parallel research with Florian Bogner and Clement Lavoillotte
  - AVGater
  - Abusing Privileged File Manipulation
- LPE in multiple endpoint security products
  - Bitdefender, Kaspersky, Symantec, …
- My approach: Self-defense bypass
  - Bare-Knuckled Anti-Virus Breaking
  - Primary idea: COM hijacking

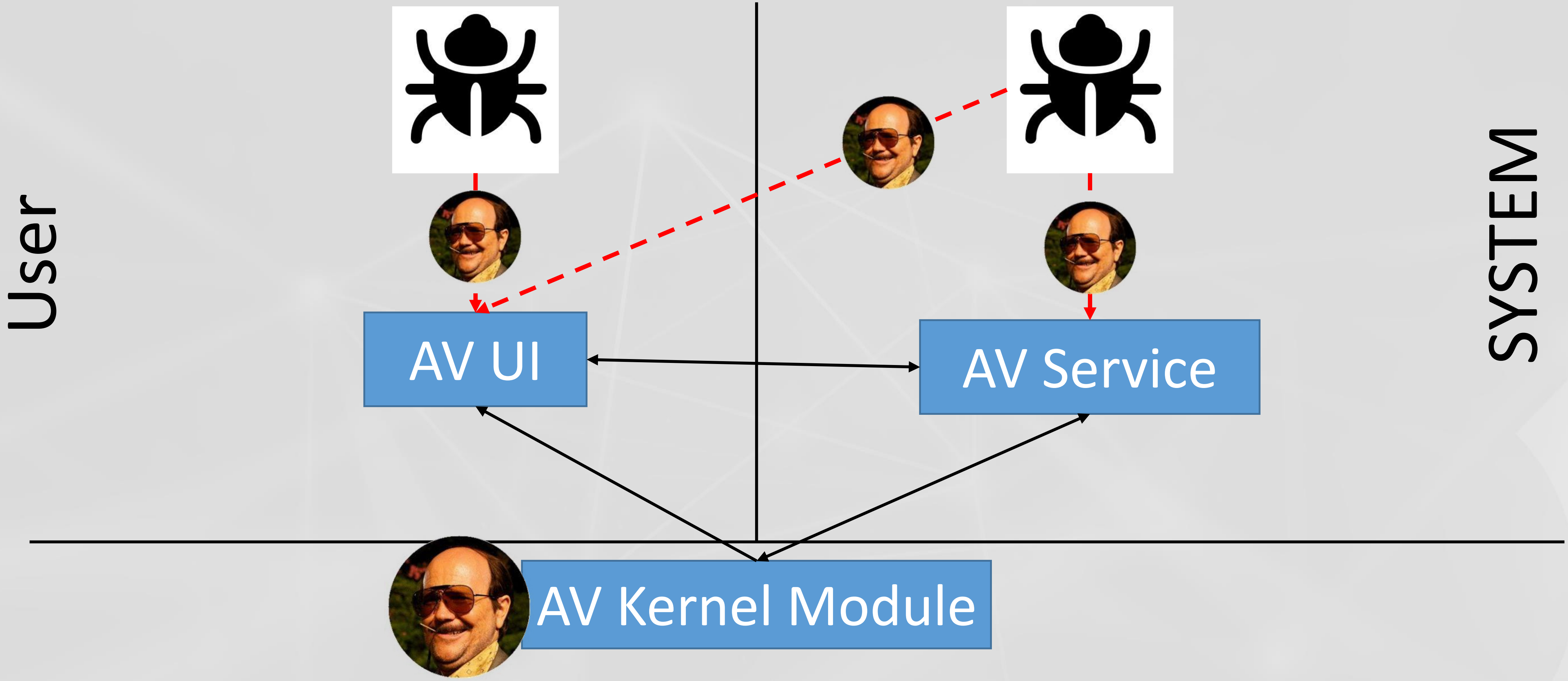
HYPOTHESIS

Self-defense hides exploitable attack surface.

ARCHITECTURE



ARCHITECTURE



SELF-DEFENSE

IS SELF-DEFENSE A SECURITY BOUNDARY?

- Symantec
  - CVE-2017-6331
- Avast
  - CVE-2017-8307
  - CVE-2017-8308
- Kaspersky
  - Bypass from 2007:
    „Kaspersky Lab does not consider this to be a vulnerability: it is not an error in our code, but an obscure method for manipulating standard Windows routines to circumvent our self-defense mechanisms."

KASPERSKY

- No political agenda here…
- Self-defense bypass != vulnerability
  - My original bypass still works
- Some experience from previous research
  - Well-known components
  - Configurability
- Only AV that caught my previous exploits while they were 0-day :P
  - I found bypasses ofc. ;)
- Research target: KFA
  - Was released around the time my research began
  - Reusable components (KIS, KES, Secure Connection…)

PRIOR WORK

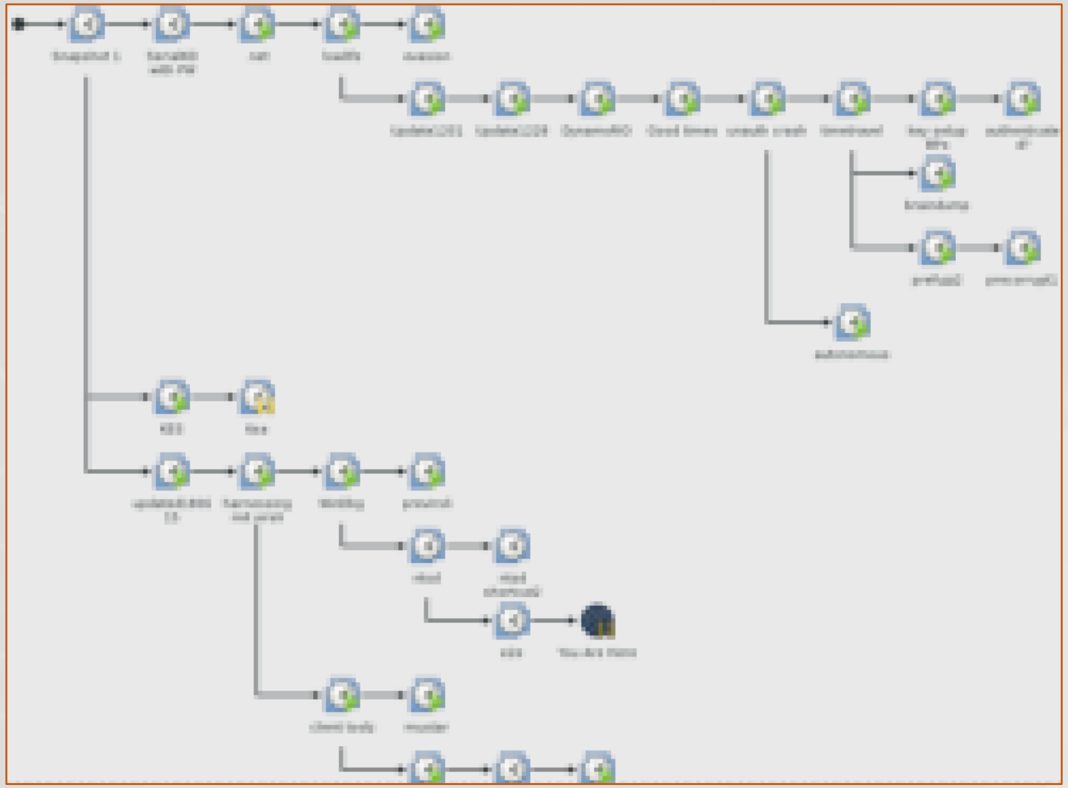
2008 SOURCE LEAK

- Kaspersky source code appeared on the Internet in 2011
  - Leaked by former employee
  - KASPERSKY.AV.2008.SRCS.ELCRABE.RAR
- Source code was from 2008
- I did not use it of course
  - That would be **illegal**…
  - *"It also contains fragments of an obsolete version of the Kaspersky anti-virus engine, which has been radically redesigned and updated since the source code was stolen"*

ANTIVIRUS DEBUGGING

silent signal

- Use VM's
  - Preferably with a good API for snapshot-revert
- Airgap
  - Unwanted updates
  - Unwanted leaks
  - More deterministic
- **Script everything**
  - Everything is slow, speed up where we can
  - pykd rocks!



ANTIVIRUS DEBUGGING

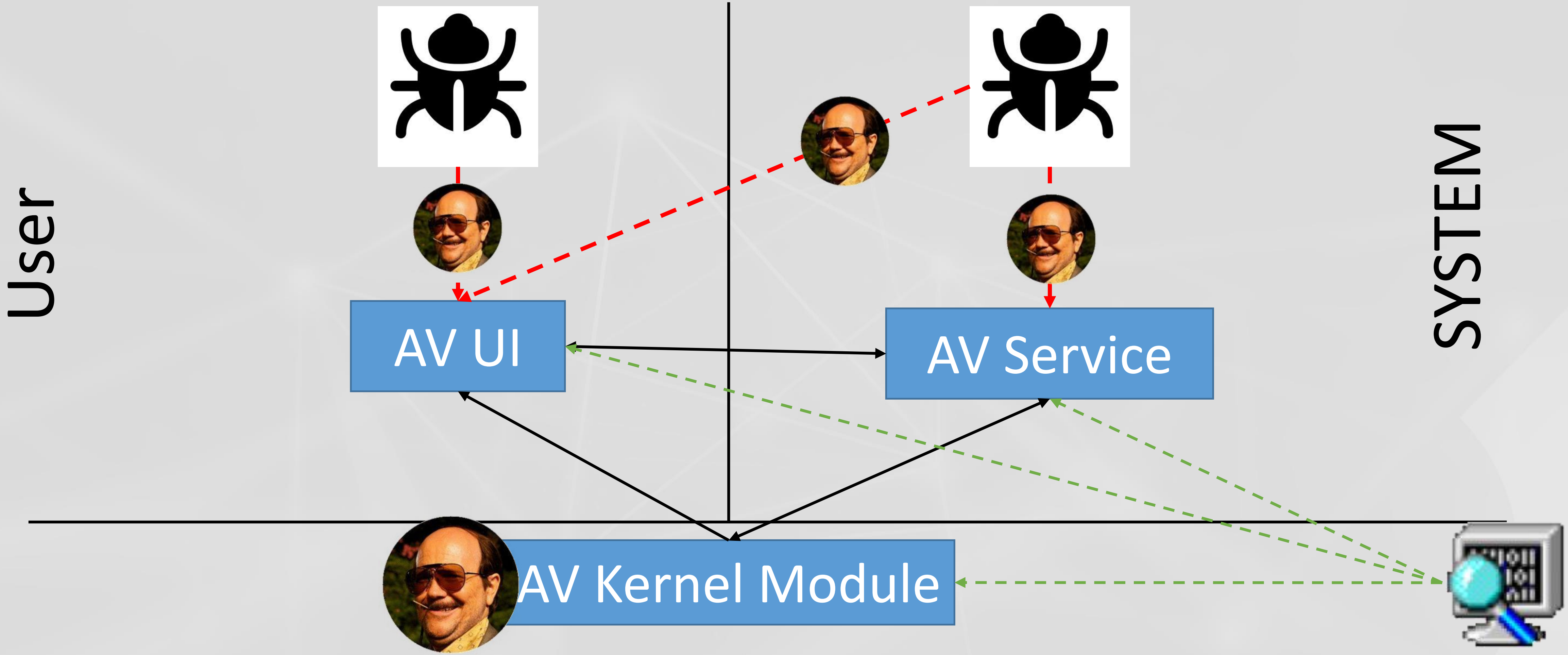
silent signal

- You *may* be allowed to disable self-defense
  - Kaspersky has an option for this
- User-mode *sometimes* works
  - Snapshot!
- Use a Kernel Debugger like proper adults!
  - Need to switch to user process context - slow!
  - Control the user debugger from KD (thx guys!)
    - Much faster (over COM port!)

```
avp_info=pykd.dbgCommand("!process 0 0 avp.exe")
avp_eprocess=avp_info.split(" ")[1]
pykd.dbgCommand(".process -r -i -p %s; " % avp_eprocess)
```

```
ntsd -d -p <PID>
```

ANTIVIRUS DEBUGGING



REVERSE ENGINEERING

KASPERSKY

- 32-bit application
  - WOW64 is hard, use a 32-bit OS for testing
- __fastcall calling convention
  - First two params in ECX and EDX, rest on stack
  - Many RE tools can't handle this…
- “Real-life” complexity
  - Module sizes in order of MBs
  - Structures/exports imitating OO design
  - Wide set of x86 instructions (killing RE tools)

KASPERSKY

silent signal

TARGET: IPC COMPONENT

- PRRemote.DLL
  - + PRCore.DLL
  - "Prague"
- Common IPC interface among multiple products
  - KFA, KES, Secure Connection, etc.
- Today's agenda:
  High level message processing (~ OSI Layer 5)
  - Needed for upper layer analysis
  - Tip of the iceberg

COMMUNICATION

Endpoints

Pid	Protocol	Name
816	ncalrpc	PRRNameService:816
816	ncalrpc	PRRemote:816
816	ncalrpc	OLE6650FA6437E59F5D10194B10DB16

Procedures

Index	Name	Address	Format

Interfaces

Pid	Uuid	Ver	Type	Procs	Stub	Callback	Name	Base	Location
816	18a27bed-c75c-28ad-4b50-52524f424a53	28387.816	RPC	0				0x000000006ee30000	C:\Program Files\Kaspersky Lab\Kaspersky Free 18.0.0\prremote.dll
816	18a27bed-d801-7233-4b50-525250524f50	28387.816	RPC	0				0x000000006ee30000	C:\Program Files\Kaspersky Lab\Kaspersky Free 18.0.0\prremote.dll
816	18a27bed-e474-f035-4b50-525250524f50	28387.816	RPC	0				0x000000006ee30000	C:\Program Files\Kaspersky Lab\Kaspersky Free 18.0.0\prremote.dll
816	18f70770-8e64-11cf-9af1-0020af6e72f4	0.0	RPC	5	Interpreted			0x0000000075ab0000	C:\Windows\System32\combase.dll
816	806411e0-2ed2-194f-bb8c-e27194948ac1	1.0	RPC	5		0x000000007fffffff		0x000000006ee30000	C:\Program Files\Kaspersky Lab\Kaspersky Free 18.0.0\prremote.dll

PRREMOTE.DLL

- Implements RPC functionality
- Functionality for both client and server
- Debug strings
  - … the reverser's best friends
- Non-trivial debug print mechanism ->

```
"Hijacking debug output:
    1) allocate new memory buffer ($dump)
    2) [$dump] <- pointer referencing the beginning of
       data inside the buffer
    3) [$dump]+0x10 Size of data DWORD, data starts at 0x18
    4) err_logger expects dst buffer in ECX, so put $dump
       there when the function starts
    5) Log information put inside $dummy when err_logger
       exits. Size of data is at $dump+8
    6) Enable err_logger by placing $dummy to the stack of
       is_Debug every time it's called

    Still crashes sometimes (on DB update attempts?)..."
```

- My notes, verbatim

(I definitely should write better notes)

PRREMOTE.DLL

```
$ strings prremote.dll | fgrep rpc_
rmt     rpc_send_receive_server exception
rmt     rpc_send_receive_server failed,
rmt     rpc_send_receive_server2 called, connection
rmt     rpc_send_receive_server2 exception during method call
rmt     rpc_send_receive_server3: failed to parse packet (size=
rmt     rpc_send_receive_server3 unknown call type:
rmt     rpc_invoke3 unknown call type:
rmt     rpc_invoke3 not enough memory to store returned data:
rmt     rpc_init_context_handle failed, RpcStatus is
rmt     rpc_send_receive2 failed, RpcStatus is
rmt     rpc_send_receive2: not enough memory to store received data:
rmt     rpc_send_receive2 call failed, RpcStatus is
rmt     rpc_send_receive3 failed, RpcStatus is
rmt     rpc_send_receive3: not enough memory to store received data:
rmt     rpc_send_receive3 call failed, RpcStatus is
rmt     rpc_disconnect_from_server exit
```

PRREMOTE.DLL

- 3 versions of `rpc_send_receive_server*()`
  - Older versions still present
- Regular breaks on `rpc_send_receive_server3()`
- Call stack shows one previous call in the module
  - I called it `my_rpc_message_handler()`
  - Deeper frames are from RPCRT4: built-in Windows RPC

PRREMOTE.DLL

`my_rpc_message_handler()`

- Called from RPCRT4
- Single argument, correctly identified as `RPC_MESSAGE*` by IDA
  - Windows RPC is merely a transport layer
  - Internal structure: *"The RPC_MESSAGE structure contains information shared between NDR and the rest of the RPC or OLE runtime."*
- Basic sanity check
- `rpc_message->Buffer` passed as argument to `rpc_send_receive_server3()`

SENDING MESSAGES

silent signal

- PythonForWindows
- Endpoint: `PRRemote:<AVP PID>`
- Interface: `806411e0-2ed2-194f-bb8c-e27194948ac1`
- Method: 4
  - What are the others for?

```
client = windows.rpc.RPCClient(r"\RPC Control\PRRemote:%d" % int(avp_pid) )
iid = client.bind("806411e0-2ed2-194f-bb8c-e27194948ac1")
ndr_params = ndr.make_parameters([ndr.NdrLong]*len(pkt))
resp = client.call(iid, 4, ndr_params.pack(pkt))
```

PRREMOTE.DLL

silent signal

MESSAGE BUFFER

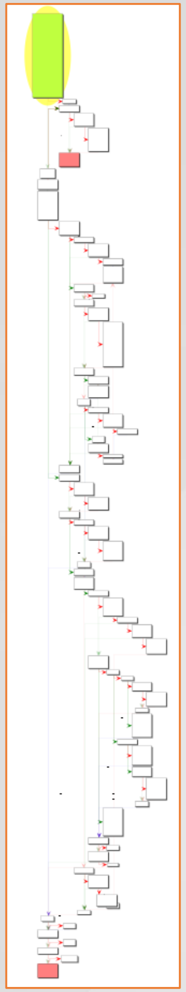
- Recognizable header
- Readable strings
  - UTF-16

```
00000000 00000000 00000000 00000000
01013200 SMALLINT
```

rpc_send_receive_server3() → **my_rpc_header_size_check()**

- Top-level message dispatcher
- Interesting strings:
  - "rmt\tReceived message has wrong integrity code"
  - "rmt\tNo session found for ID"

- `len_in`: WORD @ 0x12
- `len_out`: DWORD @ 0x14
- `len_in + len_out < rpc_msg->Size`
- LangSec ppl love this ;)



MESSAGE STRUCTURE

```
struct KASPY_IPC_REVERSED {
DWORD    zero0
DWORD    zero4
DWORD    zero8
DWORD    zeroC
WORD     doubleOne
WORD     len_out
DWORD    len_in
};
```

MESSAGE STRUCTURE

- Trace with x64dbg and Lighthouse
- Debug: *"No session found for ID"*
- Need a correct 64-bit value for parsing to happen
  - QWORD @ 0x18
  - You don't brute-force 64-bits, even locally
- Except on first connect
  - SID = 0
  - Authorization2() runs
- In practice:
  - sess0 = 0xFFFA783B (slowly grows on service respawn)
  - sess1 < 0x10000 (random DWORD on respawn)
  - Brute-force is totally practical!
  - Lack of boot-time entropy?

```
struct KASPY_IPC_REVERSED {
DWORD   zero0
DWORD   zero4
DWORD   zero8
DWORD   zeroC
WORD    doubleOne
WORD    len_out
DWORD   len_in
DWORD   session0
DWORD   session1
};
```

MESSAGE STRUCTURE

silent signal

- Debug: *"Received message has wrong integrity code"*
- Based on Flower-Noll-Vo (FNV) hash
  - Widely used algorithm, e.g. in spam filters
  - Not a cryptographic hash
  - FNV offset basis constant is present
  - Modified version, but primitives can be identified
- Created standalone implementation with ripr
  - Static code from Binary Ninja + Unicorn Engine
- 64-bits random looking prefix makes this a MAC ☹
  - Set by the client in payload upon first connect (SID=0, key=0)

```
struct KASPY_IPC_REVERSED {
DWORD    zero0
DWORD    zero4
DWORD    zero8
DWORD    zeroC
WORD     doubleOne
WORD     len_out
DWORD    len_in
DWORD    session0
DWORD    session1
WORD     unk
DWORD    hash0
DWORD    hash1
};
```

MESSAGE STRUCTURE

silent signal

- 0x101 -> protocol version
  - Header parser behavior depends on this value
  - 0x100 - 0x101
- Timestamp
- Length == 0x32

```
Set version+header length in one instr.
        MOV      dword ptr [EBP + msg.version],0x320101

        MOV      dword ptr [EBP + msg.lenIn],EAX
ff      CALL     set_kaspy_session
        LEA      EAX=>systemtime,[0xfffffed0 + EBP]

        PUSH     EAX
        CALL     dword ptr [GetSystemTimeAsFileTime]

        PUSH     dword ptr [systemtime.dwHighDateTime + EBP]

        LEA      ECX=>msg,[EBP + -0x60]
        PUSH     dword ptr [systemtime.dwLowDateTime + EBP]

ff      CALL     set_msgtime
```

```
void __thiscall set_msgtime(kaspy_msg_obj *this,dword time_low,dword time_high)

{
  if (0x100 < this->version) {
    this->systime_low = time_low;
    this->systime_high = time_high;
  }
  return;
}
```

```
struct KASPY_IPC_REVERSED {
DWORD    zero0
DWORD    zero4
DWORD    zero8
DWORD    zeroC
WORD     version
WORD     len_out
DWORD    len_in
DWORD    session0
DWORD    session1
WORD     unk
DWORD    hash0
DWORD    hash1
DWORD    time0
DWORD    time1
};
```

MESSAGE CHECKS

- Four DWORD's are needed to accept the message for further parsing
  - 2 DWORD's as "session"
  - 2 DWORD's as "integrity key"
- Current IDs/keys are stored in global structures in both the high priv. (avp.exe) and low priv. (avpui.exe) processes
  - With self-defense bypass the secrets can be obtained
  - Other options:
    - ~~Brute-force~~
    - Pre-auth messages
    - ???

BUGS

CODE REVIEW

silent signal

- Remember that length check?
- It goes like this:

my_rpc_header_size_check()

- len_in: WORD @ 0x12
- len_out: DWORD @ 0x14
- len_in + len_out < rpc_msg->Size

```
MOVZX EDX,word ptr [ECX + 0x16] ; len_out
...
MOV EAX,dword ptr [ECX + 0x18] ; len_in
ADD EAX,EDX
CMP dword ptr [EBP + size],EAX
```

- Pre-auth integer overflow
- I don't think it's exploitable (nor I am a pro exploit dev)
- Still quite telling…

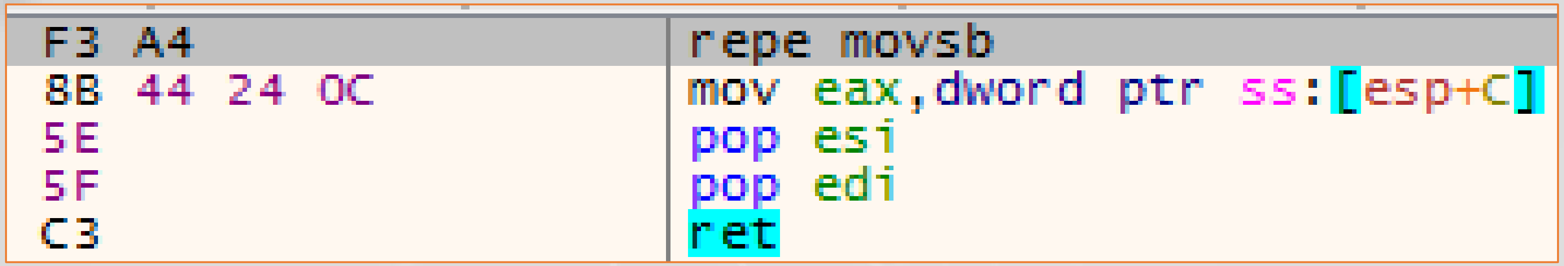
FUZZING

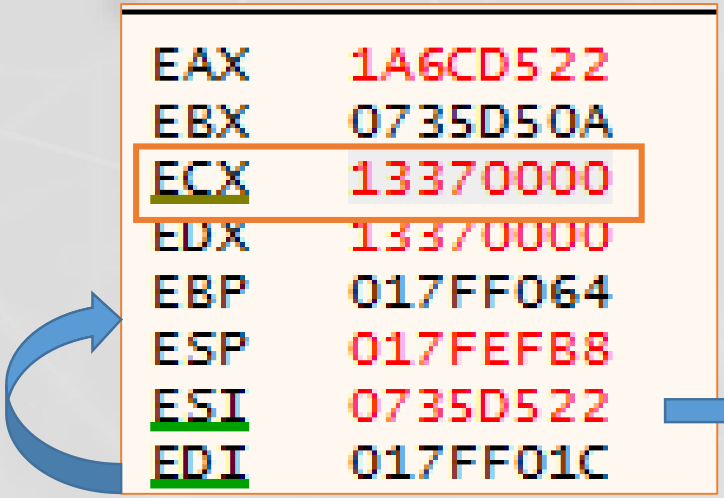
silent signal

„Any fuzzer at all, no matter how primitive, has a better chance of finding a bug than an idle CPU core." – Ben Nagy

- <20 LoC fuzzer in Python
- Replay mutated packets captured at `rpc_send_receive_server3()`
- Patched out session/integrity checks with debugger
- Pre+post auth crashes in minutes

FUZZING





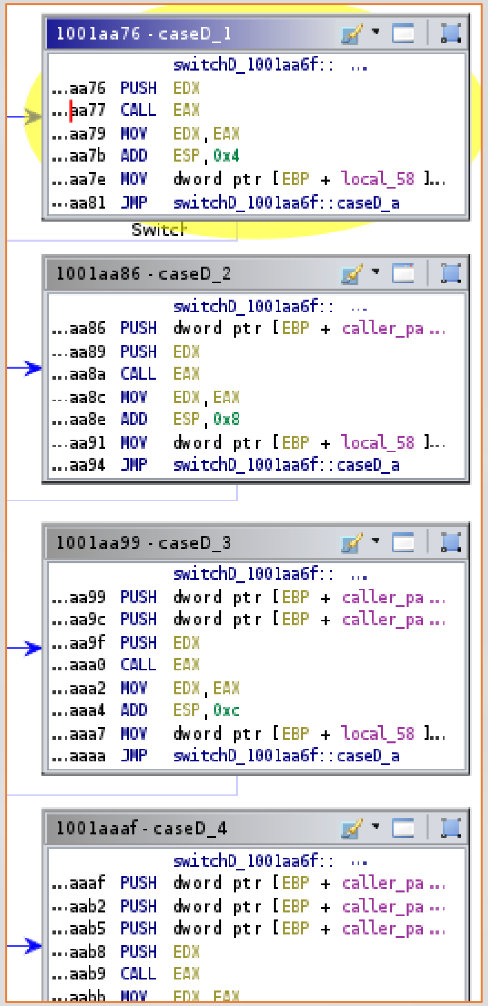
Attacker controlled

FUZZING

CONTROLLED MEMCPY

- The memcpy() in use wasn't identified as a library function
- memcpy() doesn't open a stack frame
- Caller has stack canary
  - Leak through arbitrary sized FNV preimage?
- Destination is a stack array right before the canary
- Can we do anything interesting with full control over the array?

PROCEDURE CALLS




```
call_my_buffer:
        switch((int)((int)stack - (int)&caller_vtable) >> 2) {
        case 1:
          local_58 = (*(code *)func_addr)(caller_vtable);
          break;
        case 2:
          local_58 = (*(code *)func_addr)(caller_vtable,caller_params[0]);
          break;
        case 3:
          local_58 = (*(code *)func_addr)(caller_vtable,caller_params[0],caller_params[1]);
          break;
        case 4:
          local_58 = (*(code *)func_addr)
                               (caller_vtable,caller_params[0],caller_params[1],caller_params[2]);
          break;
        case 5:
          local_58 = (*(code *)func_addr)
```

PROCEDURE CALLS

- We are in the old `rpc_send_receive_server()` now!
  - Called from `rpc_send_receive_server3()`
  - So much for "*radical redesign*"…
- `func_addr` is chosen from different function pointer tables
- User chooses the table
- User chooses the offset
- Offset is bounds checked

FUNCTION TABLES

Typical function in the table:

```
void f(int param_1,int param_2,int param_3,int param_4,int param_5,int param_6,int param_7)
{
  if (param_2 == -0xf000) {
    (**(code **)(*(int *)DWORD_100739a0 + 0x14))(0xffff1000,param_3,param_5);
    return;
  }
  (**(code **)(*(int *)(param_1 + 4) + 0x124))
            (param_1,param_2,param_3,param_4,param_5,param_6,param_7,0xffffffff);
  return;
}
```

- Can we control param1?
- Unlikely: Not present in the input stream
  - First parameter is stored early in EDX in rpc_send_receive_server()
  - Our memcpy() doesn't affect is
  - Neither does any subsequent memory corruption

FUNCTION TABLES

silent signal

```
undefined4 __cdecl call_param2(undefined4 param_1,int param_2)
{
  int iVar1;
  iVar1 = (**(code **)(*(int *)(DWORD_10077ad4 + 4) + 0x58))(DWORD_10077ad4,param_2);
  if (-1 < iVar1) {
    (**(code **)(*(int *)(param_2 + 4) + 0x5c))(param_2);
  }
  return 0;
}
```

Are we happy, Vincent?

EXPLOITATION

EXPLOITATION

silent signal

THE GOOD

- We are local…
  - ASLR ineffective
  - Arbitrary computation (dynamic shellcode, ROP, etc.)
- AVP respawns
- Pokemon exception handling

THE BAD

- Stack canaries
  - Thanks Tavis…
- DEP
- Losing session+keys at respawn
- Heap entropy still exists
  - Randomizing things before it was cool…

EIP CONTROL

- 4th WORD after header holds flags
  - Needs proper setting to reach the table based call
- Next DWORD is the table offset
- What on Earth is this?

```
undefined4 __cdecl call_param2(undefined4 param_1,int param_2)
{
  int iVar1;
  iVar1 = (**(code **)(*(int *)(DWORD_10077ad4 + 4) + 0x58))(DWORD_10077ad4,param_2);
  if (-1 < iVar1) {
    (**(code **)(*(int *)(param_2 + 4) + 0x5c))(param_2);
  }
  return 0;
}
```

EIP CONTROL

silent signal

- Looks like a method call on a global object
- Implementation in PRCORE.DLL
  - The real deal is reached after multiple calls
  - my_struct_checker()

```
undefined4 __cdecl call_param2(undefined4 param_1,int param_2)
{
  int iVar1;
  iVar1 = (**(code **)(*(int *)(DWORD_10077ad4 + 4) + 0x58))(DWORD_10077ad4,param_2);
  if (-1 < iVar1) {
    (**(code **)(*(int *)(param_2 + 4) + 0x5c))(param_2);
  }
  return 0;
}
```

STRUCT CHECKER

```
uint my_struct_checker(int ptr,dword char_out)
{
  uint ptr1;

  ptr1 = -(uint)(ptr != 0) & ptr - 0x4cU;
  if ((ptr1 != 0) && ((char)char_out != 0)) {
    char_out = 0;
    (*__ptr_check_param1)(ptr1 + 0x54, &char_out,4,0);
    if ((char_out == 0) || (char_out != ptr1 + 0x58)) {
      ptr1 = 0;
    }
  }
  return ptr1;
}
```

STRUCT CHECKER

silent signal

```
int my_check_param1(byte *ptr, byte *char_out, int ctr4)
{
  int iVar1;
  int *in_FS_OFFSET;
  undefined local_14 [16];

  iVar1 = *in_FS_OFFSET;
  *(undefined **)in_FS_OFFSET = local_14;
  while (ctr4 != 0) {
    *char_out = *ptr;
    ctr4 = ctr4 + -1;
    char_out = char_out + 1;
    ptr = ptr + 1;
  }
  *in_FS_OFFSET = iVar1;
  return 0;
}
```

STRUCT CHECKER

- I used dynamic analysis + VM snapshots to keep heap addresses constant
  - If it works, it's not stupid!
- These functions get hit all the time
  - Must single-step from rpc_send_receive_server()
- Struct checker performs basic sanity checks
- Param2 has to survive multiple dereferences
  - Provide self-referencing pointers

STRUCT CHECKER

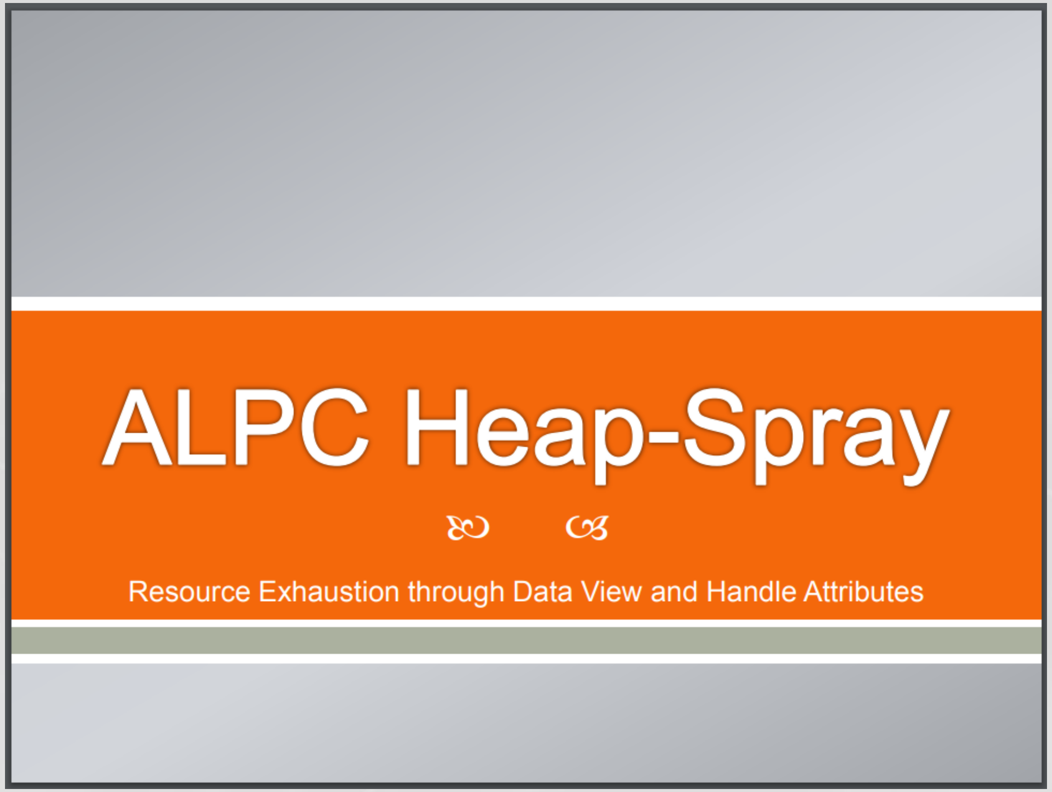
- Sent 20K packages with self-referencing pointers, then the trigger packet
  - Still based on predictable heap addresses + VM snapshots
- Checks passed -> EIP overwritten \o/
- EIP value read from an address after the checked struct values -> Possible to control!
- How?

WE NEED TO SPRAY THE HEAP!

HEAP SPRAY

- Tests showed that packet sizes are limited (~2K)
- Parsed buffers are freed by `my_rpc_msg_handler()`
- Hooked HeapAlloc in IAT via KD
  - Terribly slow…
  - Physical page offsets?
- Patched PythonForWindows so it won't check sizes or wait for replies
  - Managed to spray my packets over a 78K, non-continuous space :P
  - Let's read up again on this ALPC thingy…

HEAP SPRAY



Alex Ionescu already did it!

(duh!)

HEAP SPRAY

silent signal

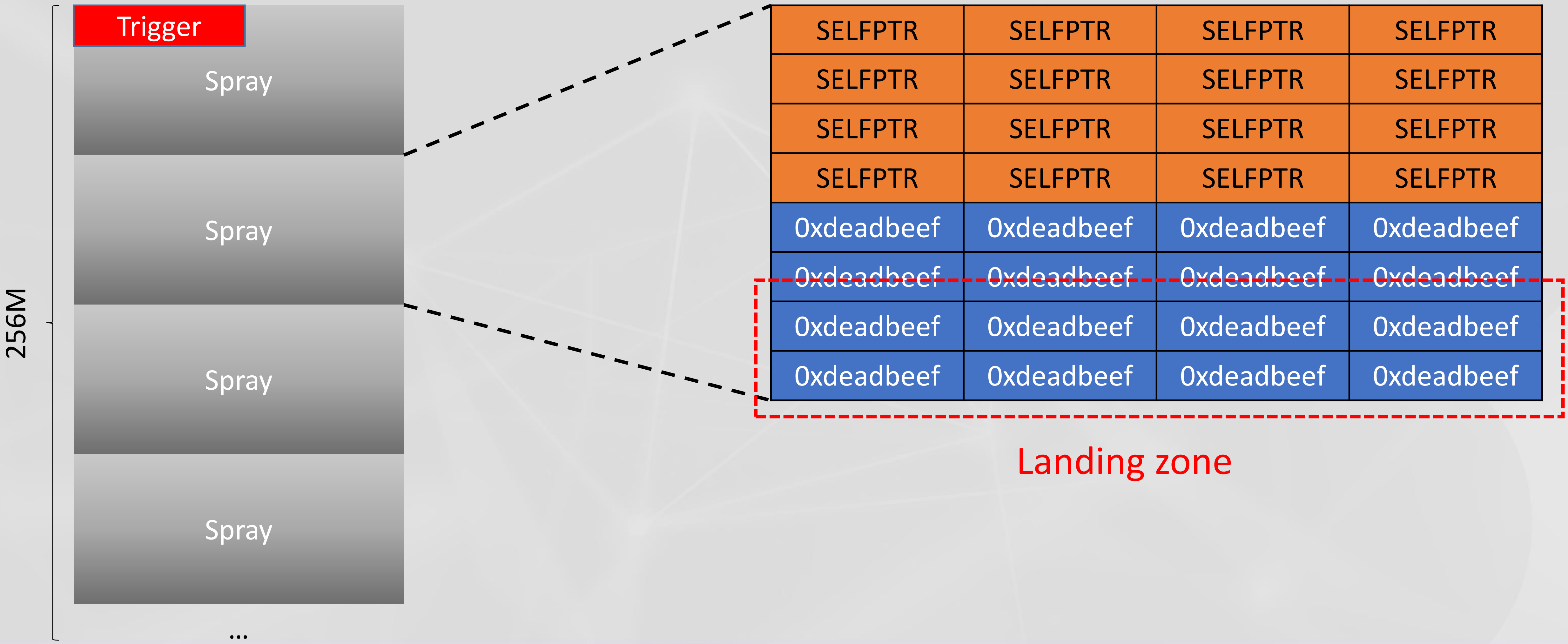
ALPC HEAP SPRAY

- ALPC allows passing large messages via shared memory
  - DataView's
- Unmapped after use (RPCRT4), but can be arbitrary large!
- Virtual base addresses will differ between client and server
- Offset inside allocation is known

STRATEGY

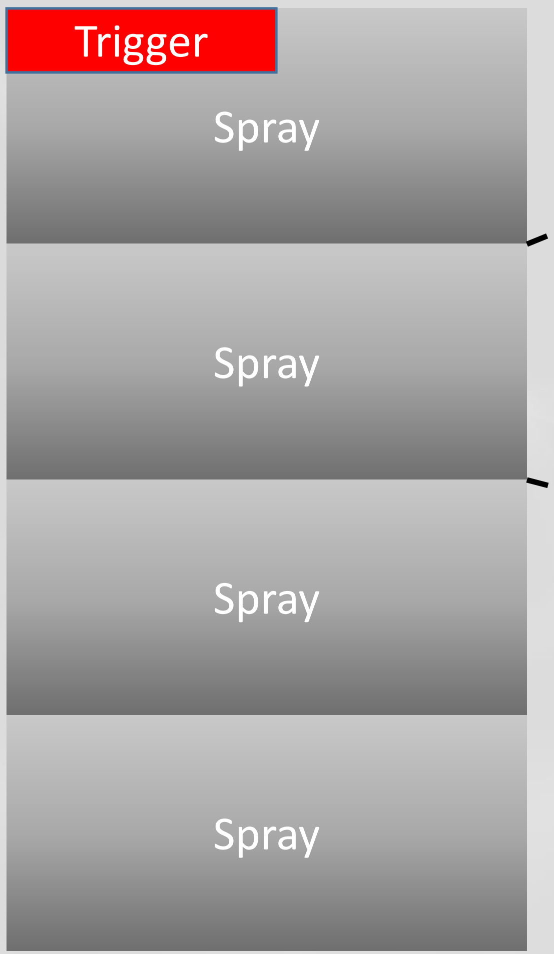
- Allocate 256M memory in our process
- Use the ALPC layer directly to send RPC message
  - PythonForWindows has example code
  - Share the 256M mapping
- Brute-force base address in avp.exe
  - Read access violations are handled :D
  - 2-3 tries in practice

HEAP SPRAY



Trigger
Spray
Spray
Spray
Spray
...
256M
SELFPTR
0xdeadbeef
Landing zone

ROP CHAIN



ESI ->

SELFPTR	SELFPTR	SELFPTR	SELFPTR
SELFPTR	SELFPTR	SELFPTR	SELFPTR
SELFPTR	SELFPTR	SELFPTR	SELFPTR
SELFPTR	SELFPTR	SELFPTR	SELFPTR
ROP1	ROP1	ROP1	ROP1
ROP1	ROP1	ROP1	ROP1
ROP1	ROP1	ROP1	ROP1
ROP1	ROP1	ROP1	ROP1

ROP1: ESP<-ESI (+POPs); RET 10

ROP CHAIN

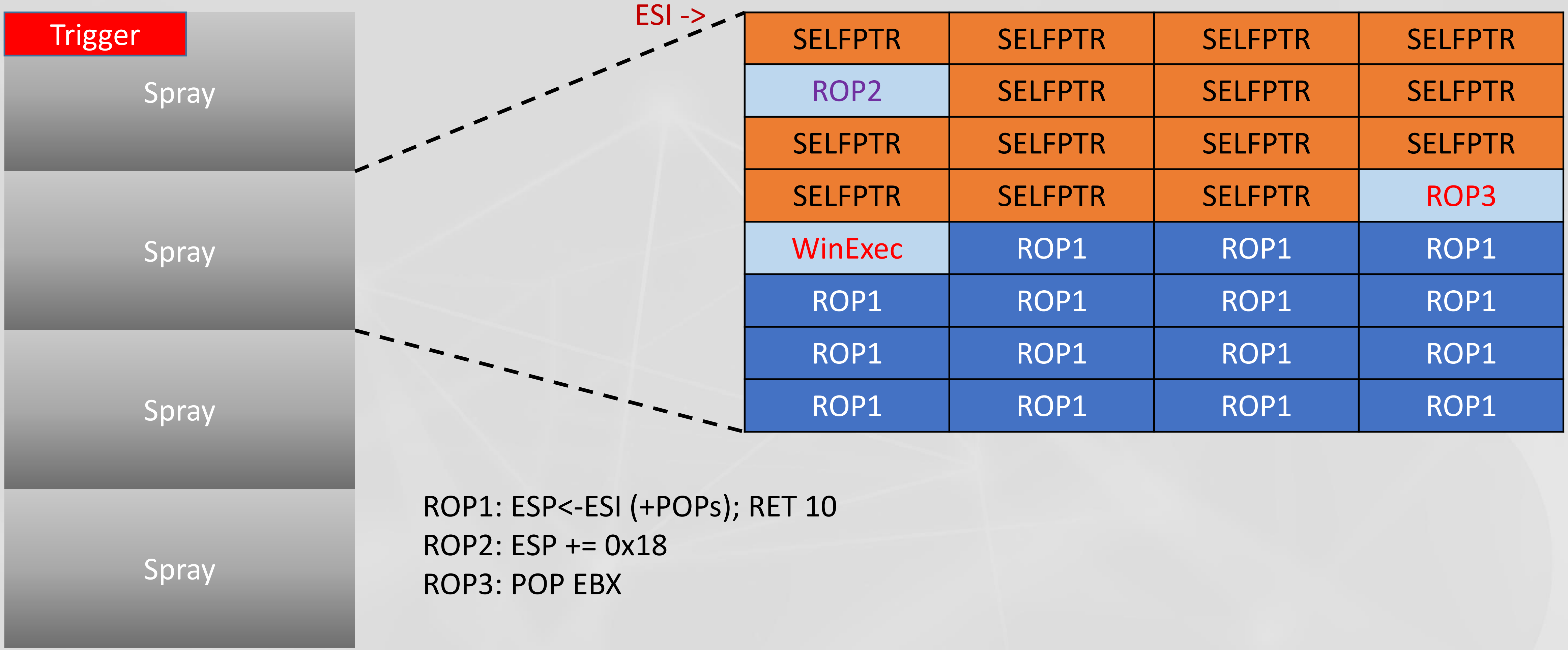
ESI ->

SELFPTR	SELFPTR	SELFPTR	SELFPTR
ROP2	SELFPTR	SELFPTR	SELFPTR
SELFPTR	SELFPTR	SELFPTR	SELFPTR
SELFPTR	SELFPTR	SELFPTR	SELFPTR
ROP1	ROP1	ROP1	ROP1
ROP1	ROP1	ROP1	ROP1
ROP1	ROP1	ROP1	ROP1
ROP1	ROP1	ROP1	ROP1

ROP1: ESP<-ESI (+POPs); RET 10
ROP2: ESP += 0x18

ROP CHAIN

silent signal



ROP1: ESP<-ESI (+POPs); RET 10
ROP2: ESP += 0x18
ROP3: POP EBX

ROP CHAIN

silent signal

ESI ->

SELFPTR	SELFPTR	SELFPTR	SELFPTR
ROP2	SELFPTR	SELFPTR	SELFPTR
SELFPTR	SELFPTR	SELFPTR	SELFPTR
SELFPTR	SELFPTR	SELFPTR	ROP3
WinExec	ROP4	Command	ROP1
ROP1	ROP1	ROP1	ROP1
ROP1	ROP1	ROP1	ROP1
ROP1	ROP1	ROP1	ROP1

Command is sprayed at every 0x10000

ROP1: ESP<-ESI (+POPs); RET 10
ROP2: ESP += 0x18
ROP3: POP EBX
ROP4: POP EDI

ROP CHAIN

silent signal

ESI ->

SELFPTR	SELFPTR	SELFPTR	SELFPTR
ROP2	SELFPTR	SELFPTR	SELFPTR
SELFPTR	SELFPTR	SELFPTR	SELFPTR
SELFPTR	SELFPTR	SELFPTR	ROP3
WinExec	ROP4	Command	ROP5
ROP1	ROP1	ROP1	ROP1
ROP1	ROP1	ROP1	ROP1
ROP1	ROP1	ROP1	ROP1

Command is sprayed at every 0x10000

ROP1: ESP<-ESI (+POPs); RET 10
ROP2: ESP += 0x18
ROP3: POP EBX
ROP4: POP EDI
ROP5: PUSH EBX; CALL EDI

DEMO

OUTRO

COORDINATED DISCLOSURE?

If these are your priorities…

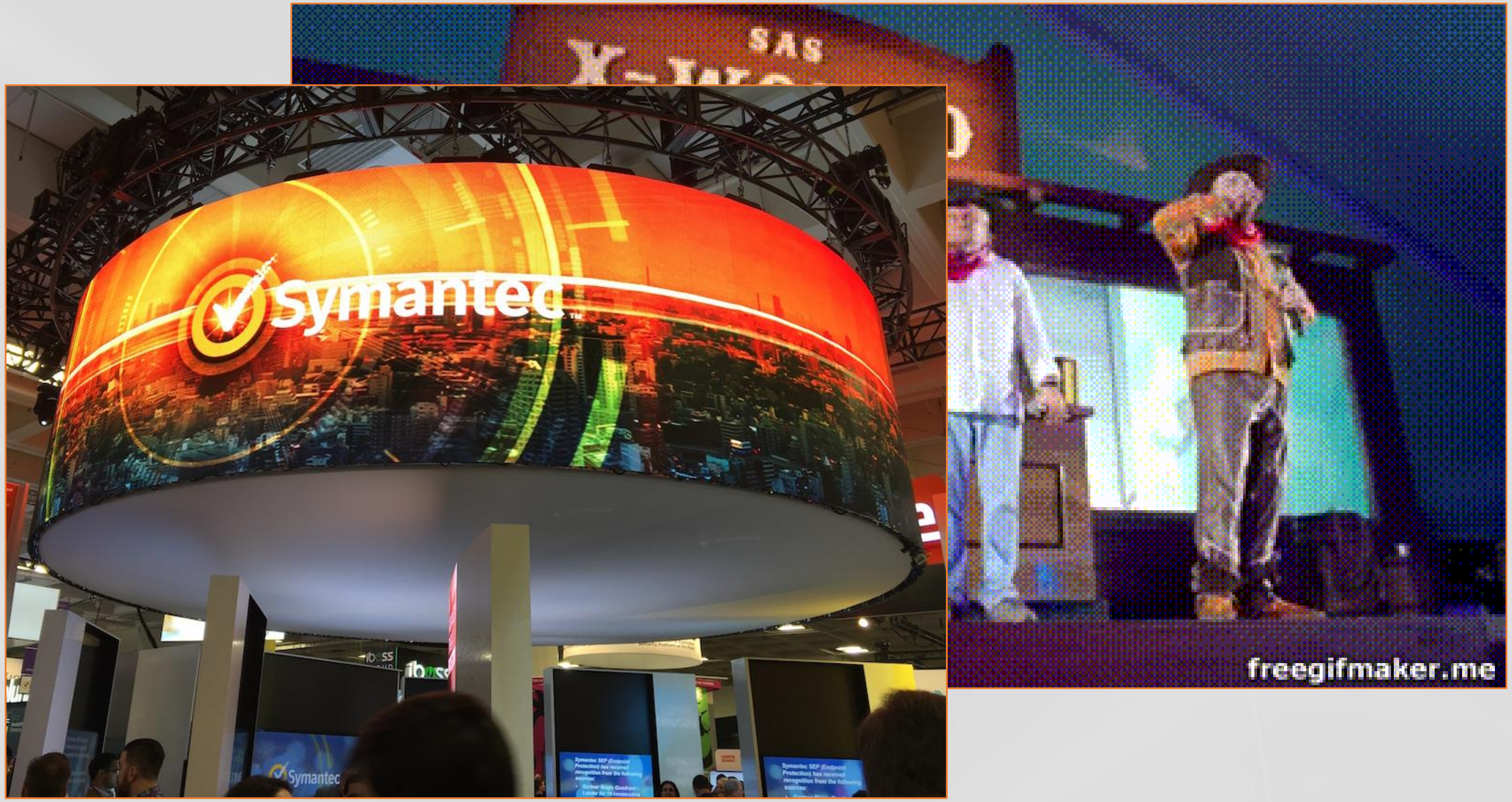
COORDINATED DISCLOSURE?

If these are your priorities…

COORDINATED DISCLOSURE?

silent signal

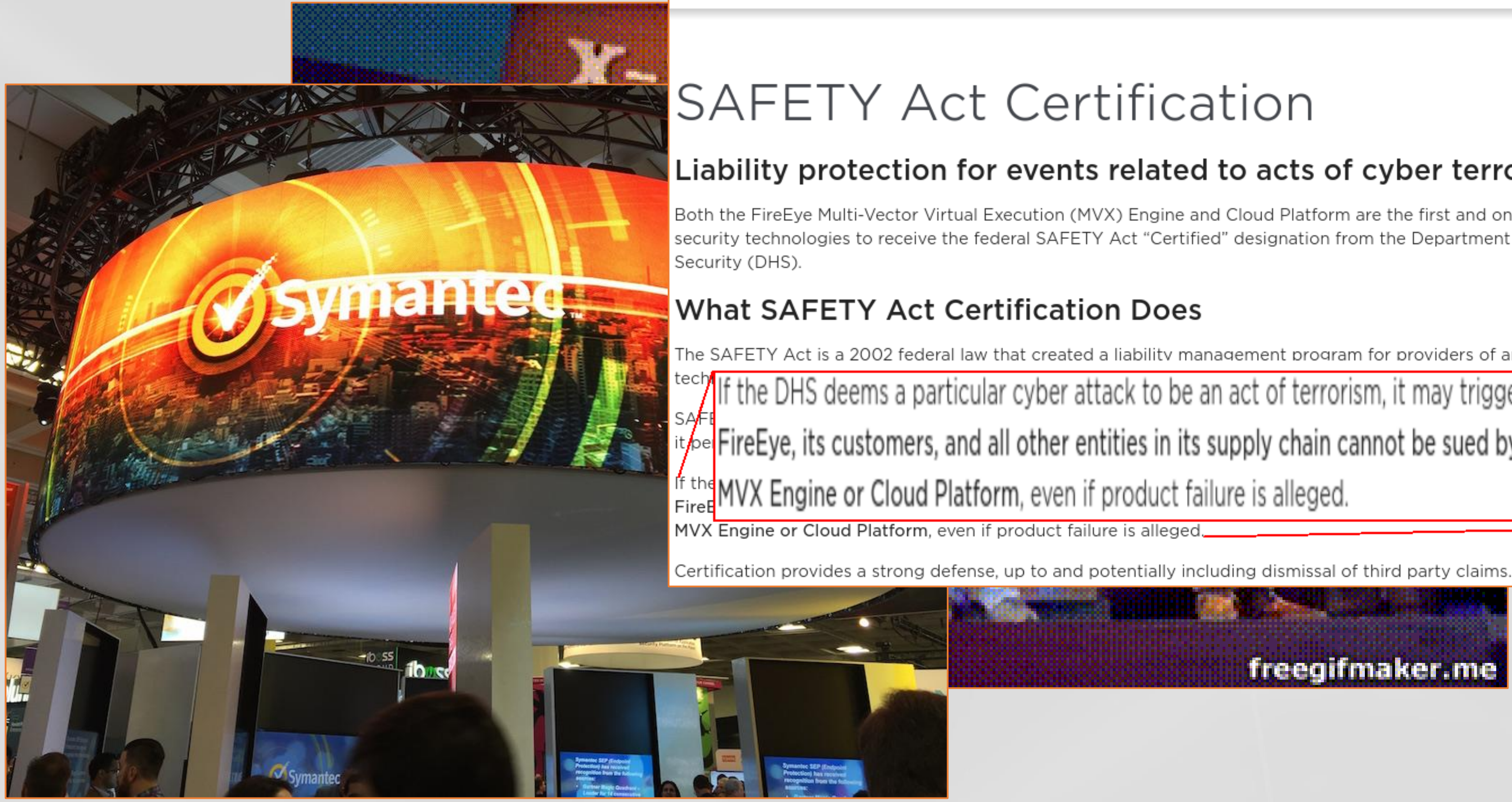
If these are your priorities…



COORDINATED DISCLOSURE?

silent signal

If these are your priorities…



FIREEYE™ Solutions Services Partners Support Resources Comp

Home > Company > FireEye Awards > **SAFETY Act Certification - Cyber Attack Liability ...**

SAFETY Act Certification

Liability protection for events related to acts of cyber terrorism

Both the FireEye Multi-Vector Virtual Execution (MVX) Engine and Cloud Platform are the first and only true cyber security technologies to receive the federal SAFETY Act "Certified" designation from the Department of Homeland Security (DHS).

CERTIFIED SAFETY ACT www.safetyact.gov

What SAFETY Act Certification Does

The SAFETY Act is a 2002 federal law that created a liability management program for providers of anti-terrorism tech…

SAFE…

it pe…

If the…

Fire…

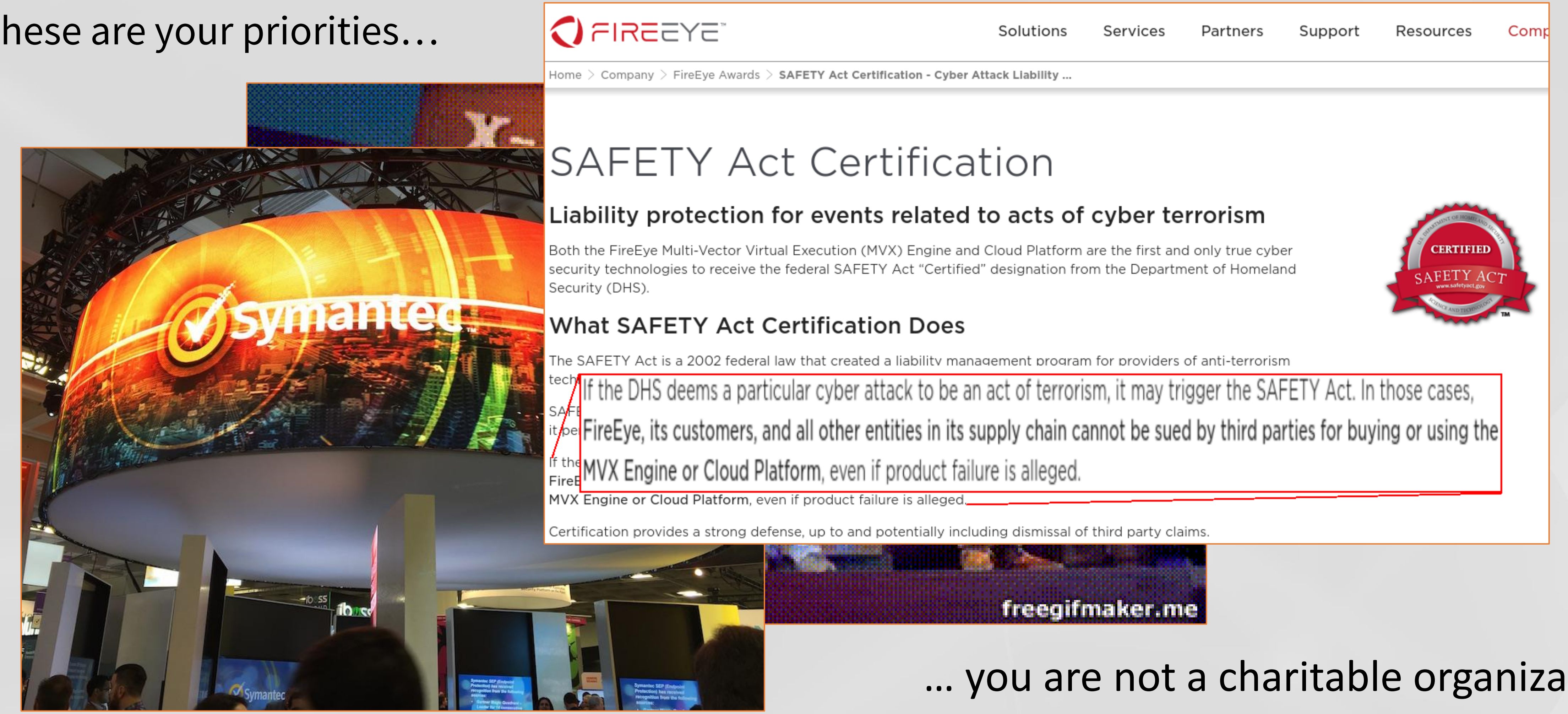
MVX Engine or Cloud Platform, even if product failure is alleged.

If the DHS deems a particular cyber attack to be an act of terrorism, it may trigger the SAFETY Act. In those cases, **FireEye, its customers, and all other entities in its supply chain cannot be sued by third parties for buying or using the MVX Engine or Cloud Platform**, even if product failure is alleged.

Certification provides a strong defense, up to and potentially including dismissal of third party claims.

COORDINATED DISCLOSURE?

If these are your priorities…



… you are not a charitable organization.

BUG BOUNTY?

silent signal

- Research value > Bounty value
- Unrealistic scoping doesn't encourage researchers
  - Client-side exploits?
  - Dependencies?
- Limited impact
  - Local
  - Needs self-defense bypass
  - PoC to be released a bit later

Scope of program:

	remote (no direct access to host, i.e. behind nat)	**LAN (network access to host in the same broadcast domain)**	**local vector (direct access to host operating system with user privileges)**
RCE in product high privilege process	$5 000[1] – $20 000[2]	$5 000[1] – $10 000[2]	-
Other RCE in product	$2 000[1] – $10 000[2]	$2 000[1] – $5 000[2]	-
Local Privilege Escalation	-	-	$1 000[1] – $5 000[2]
Sensitive[3] user data disclosure	$2 000[1] – $10 000[2]	$2 000[1] – $5 000[2]	$500[1] – $2 000[2]

Based on our product's threat model, attacks on the communication channel within remote management services (configuration, update, etc.) can be implemented on any target system regardless of user activity. Thus, by using a man in the middle attack, arbitrary code can be remotely executed in high privilege AV processes. As a result, malware code will work as part of AV product and bypass detection technologies. We take this possibility very seriously.

A special bounty of $100,000 will be awarded for high-quality report with PoC that implements this attack vector.

CONCLUSIONS

silent signal

RESULTS

- Self-defence does hide exploitable attack surface
- Self-defense bypasses are useful
  - Attack from two ends
  - Look into persistence, code injection techniques
- Kaspersky IPC parsers are fragile
- Local exploits are easy, despite mitigations

TIPS

- This is just the tip of the iceberg
  - Other parses
  - Other vendors!
- Neat ideas in other IPC research (browsers)
  - Gamozolabs, Ned Williamson+NiklasB, etc.
- Fuzzing is a metal detector
  - Interesting code > Unexploitable bugs

THANK YOU!

BÁLINT VARGA-PERKE

BUHERATOR@SILENTSIGNAL.HU

@buherator

@SilentSignalHU