Introduction to Information Security

Miscellaneous
To implement a buffer overflow, you need to know:

- The overflow size (from the buffer start to the return address)
- The stack address*

In the example, we have 120 bytes until RET, which can be 1000

Assume our shellcode is 60 bytes

- We can write the shellcode from 1000 to 1060, then 60 garbage bytes, then 1000

If the stack moves even a little, we skip some of our shellcode / execute garbage bytes!

- Assume the stack can move ±30 bytes.

- We can write 60 NOPs from 1000 to 1060, then the shellcode from 1060 to 1120, then...

  - Jump to 1000?
  - Jump to 970?
  - Jump to 1030? ✓
ROP Arguments

• In interleaved ROP codes, how does each function know what is arguments are?
  • It doesn't!
  • Each function assumes (in CDECL...) that:
    • The caller pushes the necessary arguments in reverse order (explicitly)
    • The caller pushes the return address (by using CALL)
    • The callee has the first argument at ESP+4, the second at ESP+8, etc.
      • If it has a standard prolog (PUSH EBP), then at ESP/EBP+8, ESP/EBP+12, etc.
    • So system just assumes that at its ESP+4 is the first (and only) argument
    • And exit just assumes that at its ESP+4 is its argument
    • This happens to interleave smoothly, but if we wanted more calls we'd have a problem
      • Or if system had 2 argument, it'd have to share its seconds argument with exit

1
"/bin/sh"
exit
system
ROP - Continued

- So what *do* we do?
  - Gadgets that POP can "traject" us up the stack, into a less cluttered area
  - Gadgets the load ESP, change it a little, and then write to it (polymorphic ROP)
  - Be creative!
    - For example, the RET is not really the *first* thing you get control over
    - Almost all functions have an epilogue with POP EBP, so whatever you put before RET is loaded into EBP for free

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>RET</th>
<th>EBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>&quot;/bin/sh&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>exit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Symmetric Encryption

• Alice wants to talk to Bob
  • Eve eavesdrops
  • Alice and Bob encrypt the content with symmetric encryption with key $k$
    • Alice sends $SE_k(x)$ and Bob decrypts it with $SD_k(SE_k(x)) = x$
      • It should be hard to decrypt $SE_k(x)$ without $k$!
  • But how do Alice and Bob synchronize $k$?
    • If they just send it, and Eve is already eavesdropping, it was all for naught
• Asymmetric Encryption / Key Exchange!
Asymmetric Encryption / Key Exchange

- Bob has two keys – the public key, $P$, and the private key, $q$ (RSA, El-Gamal, Elliptic Curves)
  - Bob sends $P$ to Alice.
  - Alice uses $P$ to encrypt $k$ and sends $AE_P(k)$
    - It should be hard to decrypt $AE_P(k)$ without $q$ (not $P$!)
      - That's the magic of Asymmetric encryption
  - Bob decrypts it with $AD_q(AE_P(k)) = k$, et voilà! A symmetric key is exchanged
- Simpler Key Exchange – Diffie Hellman
  - Alice and Bob agree publicly on $g$ (a generator of a cyclic group...)
  - Alice generates $a$ and sends $g^a$
  - Bob generates $b$ and sends $g^b$
  - Alice and bob use $(g^b)^a = (g^a)^b = g^{ab}$, which should be otherwise hard to obtain (not $g^{a+b}$!)
SSL

• Ciphersuites
  • Most common uses RSA and AES

• MITM
  • Bob sends $P$ to Alice, but Eve (which generated $P'$ and $q'$) receives it and sends $P'$ instead
  • Alice sends over $AE_{p'}(k)$, Eve decrypts it with $AD_{q'}(AE_{p'}(k)) = k$ and sends $AE_p(k)$
  • Bob decrypts it with $AD_q(AE_p(k)) = k$ and starts talking with Alice using $k$ – but Even knows it!

• Certificates
  • Bob sends **his name and $P$, signed**
Signatures

- Asymmetric Cryptography can be used for Digital Signatures as well as Encryption
  - For simplicity, I assume the encryption/decryption functions are commutative
    - ($\star$) $AD(AE(x)) = AE(AD(x))$
    - Charlie generates $P$ and $q$, and to sign $x$ generates $x \circ AD_q(x)$
    - Anyone with $P$ (which is public) can validate the $x = AE_P(AD_q(x)) = AD_q(AE_P(x)) = x$
  - A browser has a hardcoded list of CA (certificate authorities) whose public keys it knows
SSL - Continued

• Bob contacts a CA, proves he's Bob, and purchases a certificate (Bob, P, S).
  • Where \( S = AD_{q_{CA}}(Bob \circ P) \) is a signature on both the name Bob and the key P.
• Eve intercepts (Bob, P, S).
  • She could send (Bob, P, S) – but then she won't be able to decrypt \( AE_P(k) \)
  • She could send (Bob, \( P' \), S) – but then the signature is invalidated
    • Only the CA should be able to "regenerate" the signature
    • The CA won't agree to do that for Eve, since she can't prove that she's Bob (she's not...)
      • Tricking the CA, or hacking into its system and getting \( q_{CA} \), is an option
  • She could send (Eve, \( P' \), \( S' \)) – but then Alice might notice it's not Bob she's speaking to
    • Remember UI redressing? Imageshack, etc?
  • She could send (Even, P, S) – but that's just stupid 😐
SSL – What Can Be Done?

• Version Rollback Attacks – sabotage the handshake so an older ciphersuite is attempted
• Compression Attacks – assuming the content is compressed
  • Measure compression ratio of various inputs to learn more on "neighboring" data (cookies)
• HeartBleed – an echo request used as a heartbeat
  • The length was determined by the sender, so a small echo with a large length leaked information
• The browser's known CAs list can be changed – that's not actually a vulnerability...
  • Enterprises do that to proxy SSL
TPM

• Does nothing **active** on its own!
  • "Documents" the boot process into PCR registers
  • The BIOS/Bootloader/OS can access these registers to verify the state wasn't compromised
• Sounds weak – if I compromised the OS, I can compromised its verification processes
  • The verification can be done against external resources
    • Dedicated hardware (built-in "learning" chips, or USB dongles)
    • Cloud applications