Uberwall

0xDC626572 0x57616C6C

Syscall Proxying fun and applications

csk 'at' uberwall 'dot' org

UberWall
0xDC626572 0x57616C6C

SPEAKER

- csk from uberwall security
- Security auditor and researcher for Dreamlab Technologies AG
- Specialized in telco auditing but sometimes...

UberWall
0xDC626572 0x57616C6C

AGENDA

- Definition of syscall and OS cases
- Introduction to syscall proxying techniques
- Using syscall proxying in different applications
- Case of exploit writing – shellcodes
- Writing tools locally for remote fun – Uwsplib
- Playing further - UWinitfucker
- Future & Conclusion
- Q&A

UberWall
0xDC626572 0x57616C6C

AGENDA

- **Definition of a syscall and OS cases**
- Introduction to syscall proxying techniques
- Using syscall proxying in different applications
- Case of exploit writing – shellcodes
- Writing tools locally for remote fun – Uwsplib
- Playing further - UWinitfucker
- Future & Conclusion
- Q&A

Syscall Proxying - Introduction

- 2002 – Maximiliano Caceres (CORE SDI)
- Providing a direct interface into a target OS
- Used in automated pentest tools
- Used as “super” IPC in QNX

Syscall Proxying - Definition

- Definition of a syscall
  - Kernel trap calls used by userland programs to access wonderland (kernel) functions
- OS cases
  - All unices use syscalls
  - Win32 or IOS use non transparent syscalls. Syscall proxying is still possible but not in the same way, more like shellcodes for win32

UberWall
0xDC626572 0x57616C6C

AGENDA

- Definition of a syscall and OS cases
- **Introduction to syscall proxying techniques**
- Using syscall proxying in different applications
- Case of exploit writing – shellcodes
- Writing tools locally for remote fun – Uwsplib
- Playing further - UWinitfucker
- Future & Conclusion
- Q&A

UberWall
0xDC626572 0x57616C6C

Syscall Proxying - Basics

- Preparing locally your code
- Executing remotely syscalls
- Getting the result back
- Interpretation

Syscall Proxying - Interests

- Memory resident
- Real remote kernel interface
- Everything is possible!

Syscall Proxying - Uninteresting

- The only syscall which we can't deal with : fork()
- Can't be used as is with non syscall transparent OS like win32 plateforms or Cisco IOS

UberWall
0xDC626572 0x57616C6C

AGENDA

- Definition of a syscall and OS cases
- Introduction to syscall proxying techniques
- **Using syscall proxying in different applications**
- Case of exploit writing – shellcodes
- Writing tools locally for remote fun – Uwsplib
- Playing further - UWinitfucker
- Future & Conclusion
- Q&A

Syscall Proxying - Applications

- Legitimate syscall proxy servers
  - Remote patching for minor upgrades
  - Remote debugging
  - Transparent remote IPC
  - Etc... be creative

Syscall Proxying - Applications

- Illegitimate syscall proxy servers
  - Evolved exploits
  - Evolved backdoors & rootkits
  - Attack frameworks use syscall proxy agents
  - worms

UberWall
0xDC626572 0x57616C6C

AGENDA

- Definition of a syscall and OS cases
- Introduction to syscall proxying techniques
- Using syscall proxying in different applications
- **Case of exploit writing – shellcodes**
- Writing tools locally for remote fun – Uwsplib
- Playing further - UWinitfucker
- Future & Conclusion
- Q&A

Syscall Proxying - Exploits

- Why it can be useful for exploit writers
  - Multi-stage exploits
  - Exploit scalability / modularity
  - Priviledge escalation, exploitation at the same time ?
  - attacking, covering, backdooring at the same time ?
  - Used as transparent hop(e) station during the attack / discovery process

Syscall Proxying Shellcode principles

- Locally preparing stack
- Packing and sending it to shellcode
- Remote execution
- The shellcode sends the resultant stack
- Local interpretation
- Loop
- **Syscall proxy shellcodes are universal!**

AGENDA

- Definition of a syscall and OS cases
- Introduction to syscall proxying techniques
- Using syscall proxying in different applications
- Case of exploit writing – shellcodes
- **Writing tools locally for remote fun – Uwsplib**
- Playing further - UWinitfucker
- Future & Conclusion
- Q&A

Syscall Proxying
Locals tools for remote fun - UWsplib

- What was needed to make the work easier ?
  - Shellcodes
  - Writing tools locally to be used remotely
  - Easy to use API for these tools
- What we did
  - UWsplib: ultra light libc for syscall proxy usage

Syscall Proxying
Local tools for remote fun - UWsplib

- Initialization
  - sp_init(linux_x86);
- Get stack base addr and IDT
- Using normal standard functions
  - sp_open()
  - sp_read()
  - sp_exit()
  - sp_ptrace()
  - Etc...

Syscall Proxying
Locals tools for remote fun - shell

- Accessing a shell like an interpreter without using the target ones (monitored etc... ?)
- Implimenting special functions like importing and exporting files directly
- Always with the same things in mind: least possible resident code on the owned host
- Test code: UW_sp_minishell2

Syscall Proxying
Local tools for remote fun
Remote network applications

- Writing all your network tools to be executed remotely by the owned host
- Make attacks difficult to trace
- Using the trusted host relationship to access protected areas (hop stations)
- Test code: UW_sp_simplescan

UberWall
0xDC626572 0x57616C6C

Syscall Proxying
Localstools for remote fun
Remote process infection

- Exploiting a vulnerability to:
  - Remotely injecting a parasite into a process
  - Remotely backdooring a process
- Can be useful for:
  - Worm writing
  - Stealth backdoors like a patched sshd process

Syscall Proxying
Local tools for remote fun
Remote process infection

- 1st technique: .text infection
  - Injecting code into .text section
  - Hijacking GOT to redirect read() for example to execute our parasite and return the real read()
- Work only with dynamically linked binary
- Don't change the size of the process in memory
- You are limited by the .text size
- Test code: UW_sp_injectprocess

Syscall Proxying
Local tools for remote fun
Remote process infection

- 2nd technique: two stage injection with mmap code
  - Inject mmap shellcode into process
  - Execute it and return created memory zone
  - Inject parasite into this zone
  - Wake up the code with signal()/alarm() code
- Work with statically linked binary (init ?)
- Modify process size in memory
- You are not limited by the parasite size
- Test code: UW_sp_mmapinject

Syscall Proxying
Local tools for remote fun
I'm just too lazy sometimes

- Rewriting all my tools... NO WAY... i'm lazy
- How can I use my old tools: LDPRELOAD
- Just a syscall wrapper to use with “normal” tools
- Still in development but already usable
- Test code: UWskyzoexec

UberWall
0xDC626572 0x57616C6C

AGENDA

- Definition of a syscall and OS cases
- Introduction to syscall proxying techniques
- Using syscall proxying in different applications
- Case of exploit writing – shellcodes
- Writing tools locally for remote fun – Uwsplib
- **Playing further - UWinitfucker**
- Future & Conclusion
- Q&A

Syscall Proxying
Local tools for remote fun
Remote Kernel patching

- Exploiting a vulnerability
- Priviledge escalation if needed
- Patching the kernel remotely using well known IDT tricks and others for on the fly kernel patching througth /dev/kmem with sp_lseek(), sp_read(), sp_write() functions.

Syscall Proxying
Local tools for remote fun
Uwinitfucker - concept

- Using a vulnerability to directly rootkit the remote host during the exploitation process
- Modularity because all the code is on the client side e.g the attacker host
- Least code possible on the owned side
  - For antiforensic purposes
  - Scalability of your kit during the ownage
- Memory resident

Syscall Proxying
Local tools for remote fun
Uwinitfucker - how

- Using a vulnerability to inject syscall proxy code (here a findsock IDT patched one)
- Priviledge escalation if needed
- Remotely patching the kernel -> sp_ptrace() init
- One byte kernel patching (2.4.x and 2.6.x)
- Using mmap technique and then inject the kit into init
- Patch the kernel again to put everything back into place

Syscall Proxying
Local tools for remote fun
Uwinitfucker - parasite

- Mix between inline assembly and C code
- Independent of the compilation platform
  - Manual syscall trap code
  - Lot of manual defines
  - All made to have a parasite binary which fit into the target architecture / OS
- Integrate a syscall proxy server

Syscall Proxying
Local tools for remote fun
Uwinitfucker - parasite

- Executed for the first time by the injected mmap signal()/alarm() shellcode
- Loop in non blocking read() during 5sec expecting the “magic” ICMP packet to wake up
- If it receives the packet -> fork() and connects back to the source using as port the ICMP sequence number.
- signal()/alarm() code waking every 60 seconds

Syscall Proxying
Local tools for remote fun
Uwinitfucker - client

- Send “magic” ICMP packets while waiting for the parasite to connect back.
- Once connected it provides a local syscall proxy access and waits for your tools

Syscall Proxying
Local tools for remote fun
Uwinitfucker - demo

- Do we have time for a demo ? :p

UberWall
0xDC626572 0x57616C6C

AGENDA

- Definition of a syscall and OS cases
- Introduction to syscall proxying techniques
- Using syscall proxying in different applications
- Case of exploit writing – shellcodes
- Writing tools locally for remote fun – Uwsplib
- Playing further - UWinitfucker
- **Future & Conclusion**
- Q&A

Syscall Proxying – Future & Conclusion

- Investigate the legitimate server side
- “Good” or “bad” worms
- OS development
- What can we invent ? Be creative!

UberWall
0xDC626572 0x57616C6C

AGENDA

- Definition of a syscall and OS cases
- Introduction to syscall proxying techniques
- Using syscall proxying in different applications
- Case of exploit writing – shellcodes
- Writing tools locally for remote fun – Uwsplib
- Playing further - UWinitfucker
- Future & Conclusion
- **Q&A**

Syscall Proxying – Q&A

?

Uberwall

0xDC626572 0x57616C6C

Thank you for your attention

see you at the bar :)