Lecture #5: On Safes, Sandboxes, and Spies

Now that we have some concepts...
Its time for a more deep dive

- Use this as a review of what we've learned so far and putting it into action
- Deep Dive #1: Safes
  - Requirements
  - Cost/benefit tradeoffs
  - Detection & response countermeasures
- Deep Dive #2: Sandboxes
  - Detailed concept and objectives
  - How to Sandbox on Linux, Old School
  - How to Sandbox on Linux, New School
- Deep Dive #3: Spies

So You Want to Buy a Safe...

- What are the actual requirements?
- Protect against:
  - Fire damage?
  - Low Grade Threats?
  - Legal Liability?
  - Determined Theft?

Fire Damage

- Most "safes" you buy at Office Depot are not actual safes...
  - They are not rated nor even well designed to keep out a burglar
- Rather, they are "fire safes": designed to prevent damage in case of a file
  - Often rated by Underwriters Laboratories (UL): means tested to a given standard
- Two big categories
  - Documents/guns/etc: Keeps temperatures below 350F
    - Will keep that passport from burning
  - Data safes: Keeps temperatures below 125F and humidity below 80%
    - Computer media much more delicate
  - Testing also indicates duration
- Security lesson: Know what you are protecting and what your threat is
  - Don't expect a document-rated fire safe to keep a hard drive safe from damage in a fire
  - Don't expect either to meaningfully stop a teenager with a crowbar
- And do your threat modeling ***before*** you commit to a security procedure!

Low-Threat Entry & Legal Liability

- Some safes are concerned with rather low-threat attackers
  - Toddlers and the like
- Classic example are CA state mandated "gun" locks
  - A long list of "approved" devices
  - That often can't even keep a toddler out!
- Security Lesson: Checkbox security and real security are often two different things

High Threat: Ratings

- UL Listed ratings for various level of attackers
- Residential Security Container:
  - 5 minutes by a professional with hammer etc
- Ratings up from there
  - TL-15
    - 15 minutes with power tools
  - TRTL-30
    - 30 minutes with power tools or cutting torch
  - TXTL-60X6
    - 60 minutes, working on all 6 sides, and the attacker even gets to use 8 ounces of nitroglycerine!
- These are conservative ratings:
  - They assume an attacker with the proper set of tools ***and knowledge of the safe's construction*** oe
  - Shannon's Maxim: "The enemy knows the system"
  - Kerckhoffs's Principle: "A cryptosystem should be secure even if everything about the system, except the key, is public knowledge."

Detection and Response

- A safe doesn't have to be a passive device!!
  - In addition to the burglar alarms
- Relockers: Fail closed when under attack
  - EG, a piece of glass which holds spring-loaded bolts open
  - If the glass ever breaks, the additional bolts close and stay closed
- Very expensive false positives!

The Sandbox...

- You have a lot of ***crappy*** code that take input from ***hostile*** users
  - Often written in C/C++
- In an ideal world, you'd extinguish the dumpster fire...
  - But if it is only burning within the dumpster, is it really harmful?
- The sandbox generally covers an ***entire process***
  - That way one can take advantage of operating-specific features that allow a process to restrict what it is allowed to do

Why Sandbox At All?

- The sandbox is mostly good at making C/C++ memory exploits no longer exploitable
  - Now the attacker needs to both exploit the C code AND exploit a weakness in the sandbox
- Defense in depth...
  - But why bother with defense in depth?
- Because its cheaper!
  - Cheaper to keep using the same crappy C and C++ code and put it in a letterbox than it is actually rewriting the code in a secure language!

Sandbox Objectives

- From the Chromium Project: https://www.chromium.org/developers/design-documents/sandbox
- Don't Reinvent the Wheel
- Principle of Least Privilege
- Assume Sandboxed Code is Malicious Code
- Be Nimble
- Emulation is Not Security

Don't Reinvent The Wheel

- Modern operating systems offer different mechanisms for containment
  - Modifying the OS to ad a new containment feature is going to be a loser: you ***will*** get it wrong
  - When security systems require modifying the OS its often a big danger sign

The Principle of Least Privilege

- In an ideal world, running the code in a sandbox should not require any more privileges than a normal user
  - This is ***not*** the case on old-school Linux
  - This is the case on new-school Linux and Windows
- This is critical: You don't want your sandboxing to make things worse!
  - If your sandbox does require root you must design it to give up those privileges
- Also, whatever mechanism ***must*** be inheritable:
  - Any process launched by a sandbox process ***must operate*** under the same strict restrictions
  - OR the sandboxed process MUST NOT be able to launch another process!

Assume Sandboxed Code is Malicious Code

- The sandbox ***must*** work if the code within it is compromised by an attacker
  - So simply assume for the purposes of the sandbox that the code you are running is already compromised
- Must ensure ***complete mediation***:
  - After the sandbox itself hands control over to the running code, that code ***must not*** be able to access any resource beyond that necessary to perform its operation
  - And that which it can access ***must*** be checked and treated as potentially hostile input

Be Nimble

- Sandboxing adds overhead...
  - But its often important to not add ***too much*** overhead, otherwise it gets unused
- So make an assumption:
  - For correctness, you must assume malicious code
  - For performance, you can assume ***only*** correct code
- Allows you to optimize your performance for the "good" case

Emulation is Not Security

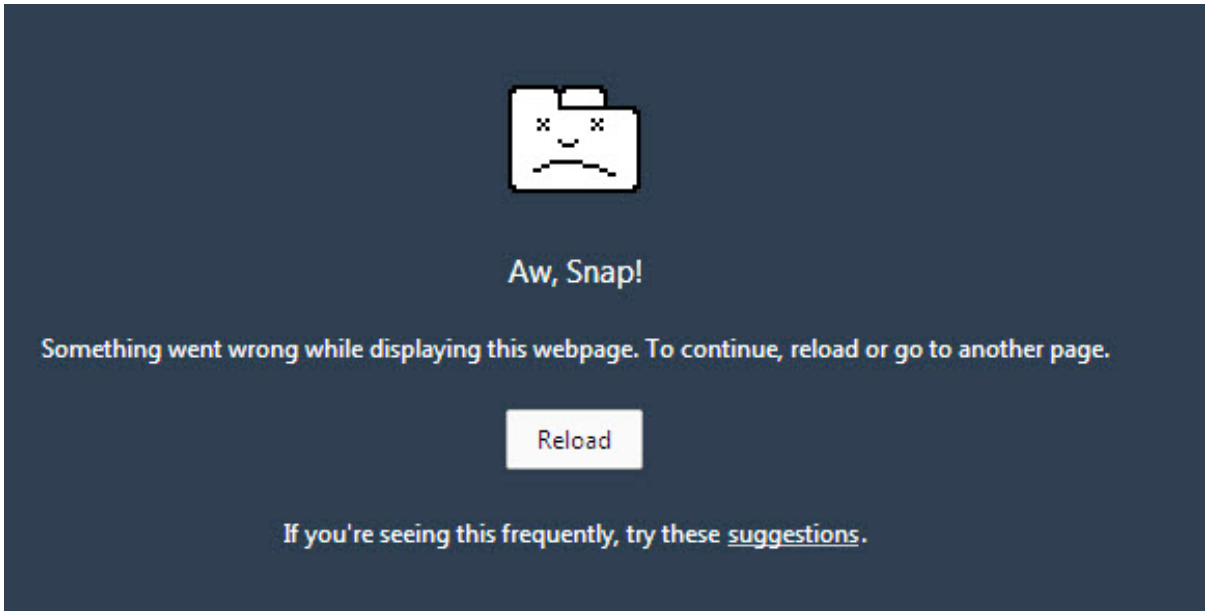
- Emulation primitives (Virutal Machines etc) are often not designed as security sandboxes!
  - Relying on something misdesigned for sandboxing can be a problem!

The Broker and the Target

- Most sandboxes separate out the problem into separate components
  - That run as separate processes
- The Broker is the reference monitor/trusted computing base
  - Its job is to start up the targets
  - ***ALL*** requests for anything sensitive in the target must be passed to the broker
- The target is the sandboxed code
  - Establish communication with the broker
  - Provide an API for talking to the broker
  - And then yield all other privileges

Robustness...

- The sandboxed process also can now fail gracefully
  - And not take the rest of the program down with it
  - So you design around the notions of sandboxed programs failing



And Don't Reinvent the Wheel #2: Just Download Someone Else's!

- Mozilla is finally adding sandboxing to Firefox...
  - Thanks to Mitar for the note
- For Windows:
  - Wrapper around Chrome's sandbox
- For Linux:
  - Uses seccomp as the building block

Old School Unix Sandboxing: the `chroot` jail

- People have wanted sandboxes for a long time...
  - Far longer than the OSs have provided fine grained mediation necessary to create sandboxes
- The gen-1 Unix Sandbox:
  - The `chroot` system call changes the definition **/** for the invoking process
- Thus it enforces a property:
  - The process (and all processes it invokes, directly or indirectly) can not read or write to any new file outside the new directory
    - But can still access existing files

Limitations of `chroot`

- It is a privileged operation!
  - Because you can do things that would compromise the system otherwise:
  - Create a directory with a file name etc/sudoers with the appropriate context
  - Now chroot to that directory
  - Now invoke sudo
  - Voila, you have root!
    - So any program using `chroot` must then drop it privileges to run as "nobody" or an otherwise unknown user
- It does ***not*** affect system call operation
  - So a "jailed" process can still access the network, call the kernel (and therefore perhaps kernel bugs), etc etc etc...
  - The "nobody" account actually still has privileges! Like the ability to interfere with other processes also owned by the "nobody" user

New School Linux Sandbox Building: `seccomp`

- Desired property:
  - An application can read and write to specified "file descriptors", but otherwise invoke no other system calls
    - File descriptors are more than just files but can be general communication pipes between processes
- `seccomp` enables a process to lock itself out of the system call interface:
  - Enables read() and write() on ***open*** file descriptors
  - Enables exit()
  - ***Everything*** else is blocked
    - An alternative form allows specifying a filter to only block some system calls, but this loses the elegance of

Using `seccomp` to create a sandbox

- Split things conceptually into a ***broker*** and one or more ***targets***
  - The targets are the sandboxed elements
  - The broker controls the targets and is the trusted base
- The broker starts up the targets
  - Using fork or clone
  - Establishes communication channels to the targets
  - Starts the target processes running
- The target process then invokes `seccomp` to give up any privileges

Communication:
Lots to chose from...

- Standard Unix pipes/file descriptors
  - Basically shove bits to/from the sandboxed process
  - Just like a network program
- Shared Memory
  - A common pool of memory that both can read and write to
  - Need to also use a semaphore to ensure coherency

Drawing Time...

Man Pages!

- "man x" is often the most important starting point
  - Then you start googling for stack overflow examples
- Required reading man pages:
  - chroot
  - seccomp
  - shm_overview

Bugging and Physical Security: The Stuff of Spies

- Going to be mostly sans-slides:
  - Notion of a SCIF/secured facility
  - History of physical bugs
  - How to bug someone today
- Inspired by this great essay:
  - https://tisiphone.net/2016/09/08/why-do-smartphones-make-great-spy-devices/

SCIF

The Great Seal Bug

TOP SECRET//COMINT//REL TO USA, FVEY

LOUDAUTO

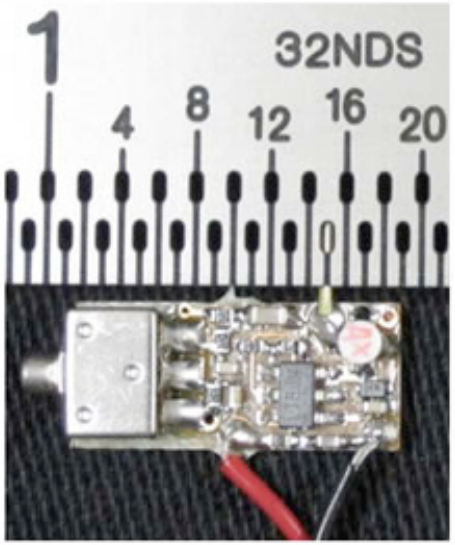
ANT Product Data

07 Apr 2009

(TS//SI//REL TO USA,FVEY) Audio-based RF retro-reflector. Provides room audio from targeted space using radar and basic post-processing.

(U) Capabilities

(TS//SI//REL TO USA,FVEY) LOUDAUTO's current design maximizes the gain of the microphone. This makes it extremely useful for picking up room audio. It can pick up speech at a standard, office volume from over 20' away. (NOTE: Concealments may reduce this distance.) It uses very little power (~15 uA at 3.0 VDC), so little, in fact, that battery self-discharge is more of an issue for serviceable lifetime than the power draw from this unit. The simplicity of the design allows the form factor to be tailored for specific operational requirements. All components at COTS and so are non-attributable to NSA.



(U) Concept of Operation

TOP SECRET//COMINT//REL TO USA, FVEY

TAWDRYYARD

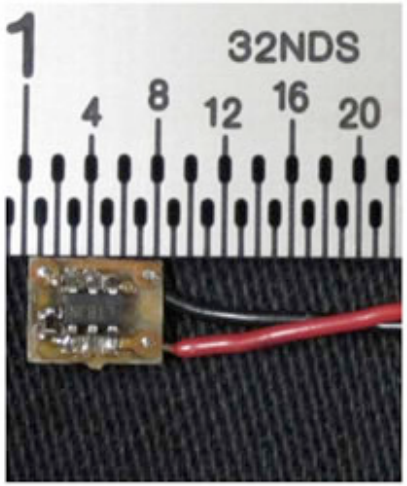
ANT Product Data

(TS//SI//REL TO USA,FVEY) Beacon RF retro-reflector. Provides return when illuminated with radar to provide rough positional location.

07 Apr 2009

(U) Capabilities

(TS//SI//REL TO USA,FVEY) TAWDRYYARD is used as a beacon, typically to assist in locating and identifying deployed RAGEMASTER units. Current design allows it to be detected and located quite easily within a 50' radius of the radar system being used to illuminate it. TAWDRYYARD draws as 8 µA at 2.5V (20µW) allowing a standard lithium coin cell to power it for months or years. The simplicity of the design allows the form factor to be tailored for specific operational requirements. Future capabilities being considered are return of GPS coordinates and a unique target identifier and automatic processing to scan a target area for presence of TAWDRYYARDs. All components are COTS and so are non-attributable to NSA.



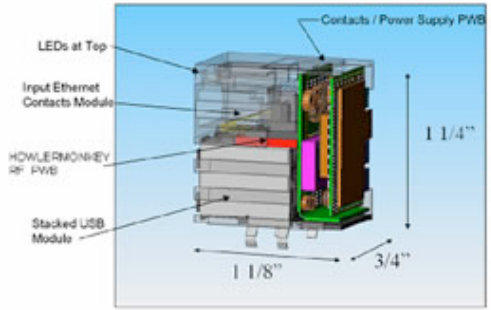
TOP SECRET//COMINT//REL TO USA, FVEY

COTTONMOUTH-III

ANT Product Data

08/05/08

(TS//SI//REL) COTTONMOUTH-I (CM-I) is a Universal Serial Bus (USB) hardware implant, which will provide a wireless bridge into a target network as well as the ability to load exploit software onto target PCs.



(TS//SI//REL) CM-III will provide air-gap bridging, software persistence capability, "in-field" re-programmability, and covert communications with a host software implant over the USB. The RF link will enable command and data infiltration and exfiltration. CM-III will also communicate with Data Network Technologies (DNT) software (STRAITBIZARRE) through a covert channel implemented on the USB, using this communication channel to pass commands and data between hardware and software implants. CM-III will be a GENIE-compliant implant based on CHIMNEYPOOL.

(TS//SI//REL) CM-III conceals digital components (TRINITY), a USB 2.0 HS hub, switches, and HOWLERMONKEY (HM) RF Transceiver within a RJ45 Dual Stacked USB connector. CM-I has the ability to communicate to other CM devices over the RF link using an over-the-air protocol called SPECULATION. CM-III can provide a short range inter-chassis link to

The GUNMAN Bug

And that brings us to the HORROR that is a cellphone!