Context-keyed Payload Encoding

I)ruid <druid@caughq.org>
Computer Academic Underground

whoami

- I)ruid
  - http://druid.caughq.org/
- Founder, Computer Academic Underground
  - http://www.caughq.org/
- Co-Founder, AHA! (Austin Hackers Association)
  - http://www.austinhackers.org/
- Employed by BreakingPoint Systems, Inc.
  - http://www.bpointsys.com

Payload Encoders

- Encoder:
  - Encodes the payload prior to exploit packaging
  - Prepends a decoder stub to the original payload
- Decoder stub:
  - Prepended to original payload
  - Executes first on the target
  - Responsible for decoding the original payload
  - Executes the original payload once decoded

Why are they used?

- Evade detection of common payloads
  - Filtering of traffic containing the likes of:
    - exec of /bin/sh or other shells
    - adduser commands
    - interaction with /etc/passwd
    - etc...
- Restricted payload byte values:
  - Input filtered for non-alphanumeric
  - Payload must pass through modification functions:
    - tolower() / toupper()
    - Character set conversions

Encoder Examples

- Metasploit (x86):
  - Alpha2 Alphanumeric Mixed-cased
  - Alpha2 Unicode Mixed-cased
  - Avoid UTF-8 and tolower()
  - Call+4 Dword XOR
  - Polymorphic XOR Additive Feedback (Shikata Ga Nai)

The Problem

- Inherent expected functionality:
  - The decoder stub must be able to decode the payload
- Existing payload encoders either:
  - Don't use a key at all
  - Use a key that is statically included in the decoder stub
- Observer can capture the payload and easily decode it for analysis
- All encoding methods I've found suffer from this problem

How can this improve?

- Always use a keyed encoder
- Don’t include the key in the decoder stub!
  - But then how does the decoder get the key?

Context-keyed Payload Encoding

Definitions

- <u>Contextual Keying</u> - The process of key selection from context information that is either known or predictable about the target.
- <u>Context-key</u> - The key value resulting from the contextual keying process.
- <u>Context-address</u> - The address at which the context-key will be found on the target.
- <u>Memory Map</u> - A file containing chunks of static data and their location addresses as will be found within an application upon execution.

Context-keyed Encoder

- Encoder
  - Encodes the payload prior to exploit packaging using the context-key
  - Prepends the decoder stub to the original payload
- Decoder stub:
  - Prepended to original payload
  - Executes first on the target
  - Responsible for:
    - Locating or generating the context-key
    - Decoding the original payload
  - Executes the original payload once decoded

Usable Context

"There are known knowns; there are things we know that we know. We also know there are known unknowns; that is to say, we know there are some things we do not know. But there are also unknown unknowns; the ones we don't know we don't know." -- Donald Rumsfeld

Context: Static Application Data

- Easy to profile if an attacker can reproduce:
  - Application's operating environment
  - Execution of the target application
- Also easy if the attacker has access to the application executable or linked libraries
- Context-key can be chosen from static values found in the process's memory
- Can use known locations of static values such as:
  - Environment variables
  - Static strings
  - The application's executable instructions (.text)

Profiling an Application

- Create an application memory map from one or more of the following methods:
  - Repeatedly poll a running process’s memory, eliminating the locations of changing data
  - Parse an application executable or dynamically-linked library’s .text data and locations where it will be mapped in memory

smem-map

- Linux application
- Relies on /proc/<pid>/maps for memory locations
- Will also do an exhaustive search of all memory
- Relies on /proc/<pid>/mem for access to memory
- Repeatedly polls the memory locations
- Eliminates data that changes
- smem-map <pid> <output.map>
- Results in a memory map of a process's static data in memory
- http://sourceforge.net/projects/smem-map/

msfpescan

- Metasploit Framework tool
- Targets Portable Executable formatted files
- Parses files for sections with data which will be loaded into memory such as .text
- msfpescan --context-map <outdir> <files>
- Results in a memory map of an executable or library's static data in memory
- http://www.metasploit.com

Memory Map

- File contains data structures for each chunk of data:
  - 8-bit: Data Type
  - 32-bit: Chunk base address
  - 32-bit: Chunk size (in octets)
  - Size: Chunk Data
- 010Editor Template Available:
  - With smem-map package from SourceForge
  - http://druid.caughq.org/src/

Context: Event Data

- Transient data may also be used as long as it persists long enough for the decoder stub to access it
- Applications that you are exploiting generally accept input somehow
- Data sent prior to or with the exploit may end up in a known location

Context: Temporal Data

- skape introduced the concept of temporal addresses
- Location in memory that holds timer data:
  - System time
  - Uptime
  - Other types of counters
- Contents originally used as viable return instructions for exploitation
- Suffers from some restrictions:
  - Window during which you can actually send the exploit
  - Data is called directly as instructions, may be marked non-executable

Context: Temporal Data

- When used as a context-key there are fewer constraints:
  - Data must not change during use of it as a context-key
  - Data remains viable provided:
    - It's used within it's update time window
    - When used as an encoding key it doesn't produce bad payload byte values
- Must be able to predict the value of the temporal data
- Frequently changing data is not useful as a context-key
- Some timers are large enough that parts of them change infrequently

Temporal Data Case Study

- Windows NT+ SystemTime is:
  - An 8 (12) byte timer
  - 100 nanosecond resolution
  - Epoch of January 1st, 1961
  - Mapped into every process at a known location as part of the SharedUserData region of memory

Windows SystemTime

- Byte Indices update frequency:
  - 0 = < 1 second
  - 1 = < 1 second
  - 2 = < 1 second
  - 3 = 1 second
  - 4 = 429 secs (7 mins 9 secs)
  - 5 = 109951 secs (1 day 6 hours 32 mins 31 secs)
  - 6 = 28147497 secs (325 days 18 hours 44 mins 57 secs)
  - 7 = 7205759403 secs (228 years 179 days 23 hours 50 mins 3 secs)
- Given the desired length of the key, the window of opportunity can be quite large
- The smaller the desired length of the key, the less exact the prediction of the target’s system time needs to be

Context-key Selection

- Using memory map static chunks as data source:
  - Select sequential data at any address that is large enough to use as a context-key
  - Check that the result of encoding the payload using that key does not violate any byte value restrictions
  - Check that the context-address does not violate any byte value restrictions
  - If everything is good, note the context-key's value and context-address

Encoding/Decoding with Context

- Encoder gets the context-key value and produces an encoded payload as usual
- Decoder stub gets the context-address and is prepended to the encoded payload
- When the decoder stub executes, it:
  - Retrieves the context-key from the context-address
  - Decodes as usual.

Proof of Concept

- Metasploit's Shikata Ga Nai
- Updated to optionally use context-keys instead of randomly generated
- From MSF Console:
  - (regular exploit & payload commands)
  - set ENCODER x86/shikata_ga_nai
  - set EnableContextEncoding 1
  - set ContextInformationFile application.map
  - exploit

ms04-007 vs. XP-SP0

- Create Memory Map
  - msfpescan --context-map context lsass-dlls/*
  - cat context/* >> lsass.exe.map
- Metasploit:
  - use exploit/windows/smb/ms04-007-killbill
  - set PAYLOAD windows/shell_bind_tcp
  - set ENCODER x86/shikata_ga_nai
  - set EnableContextEncoding 1
  - set ContextInformationFile lsass.exe.map
  - exploit

Conclusions

Q&A