

CHIRP

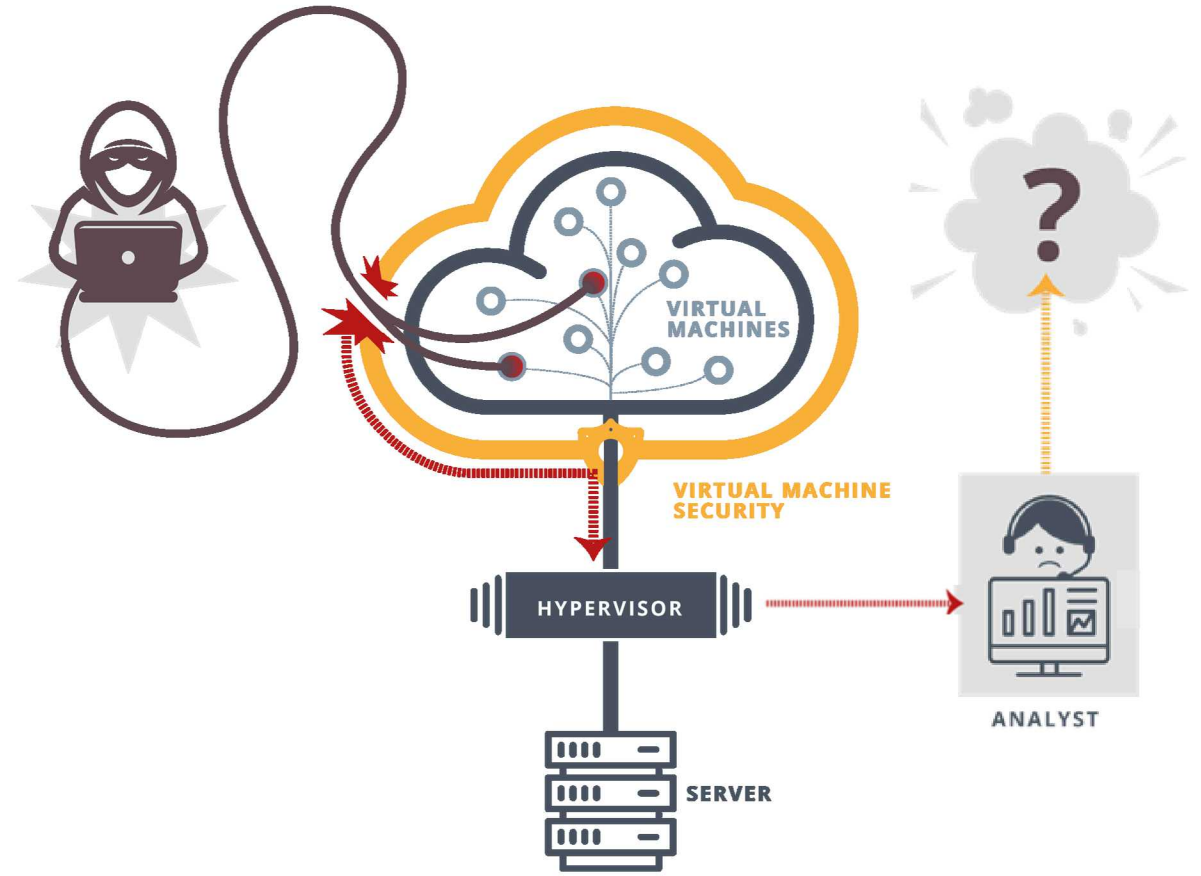
Cloud Hypervisor Forensics and Incident Response Platform

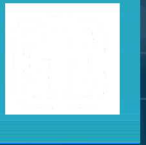
Vince Urias

SANDIA NATIONAL LABORATORIES
DOE PACT VIRTUAL SHOWCASE

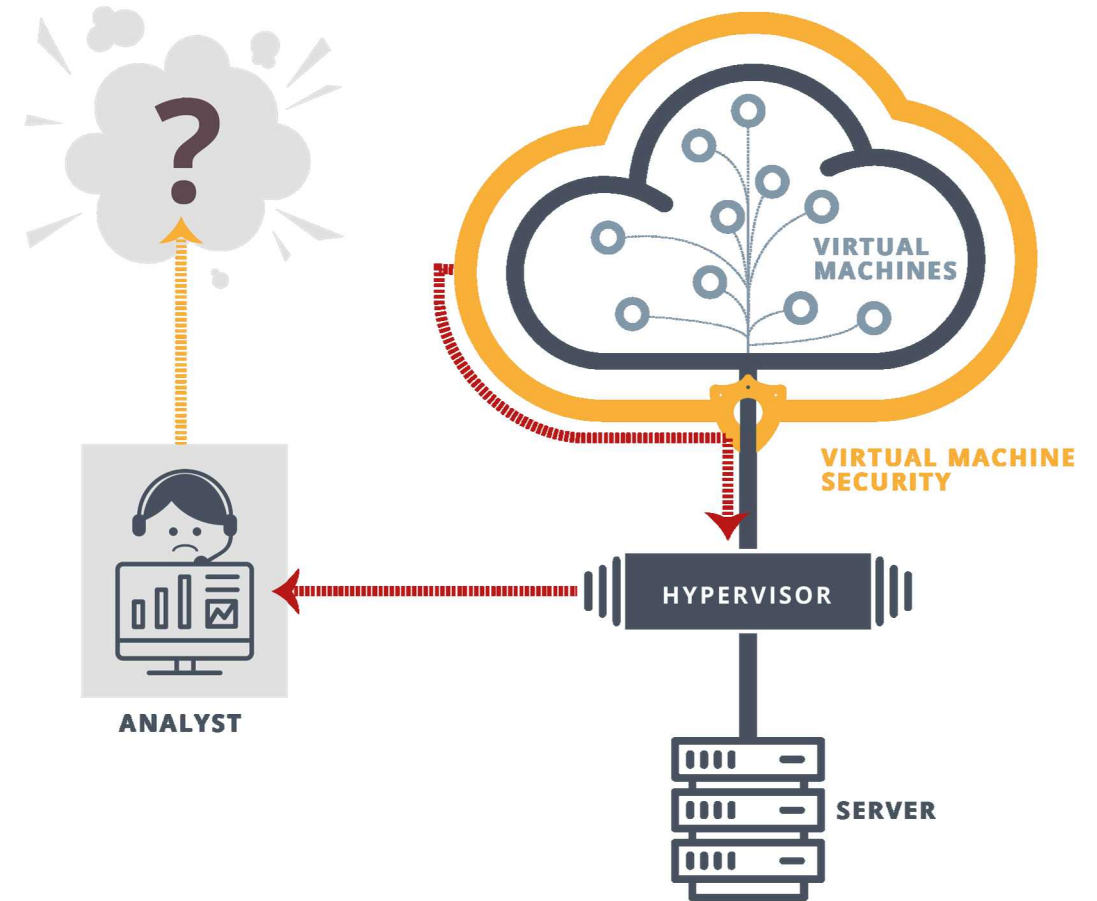


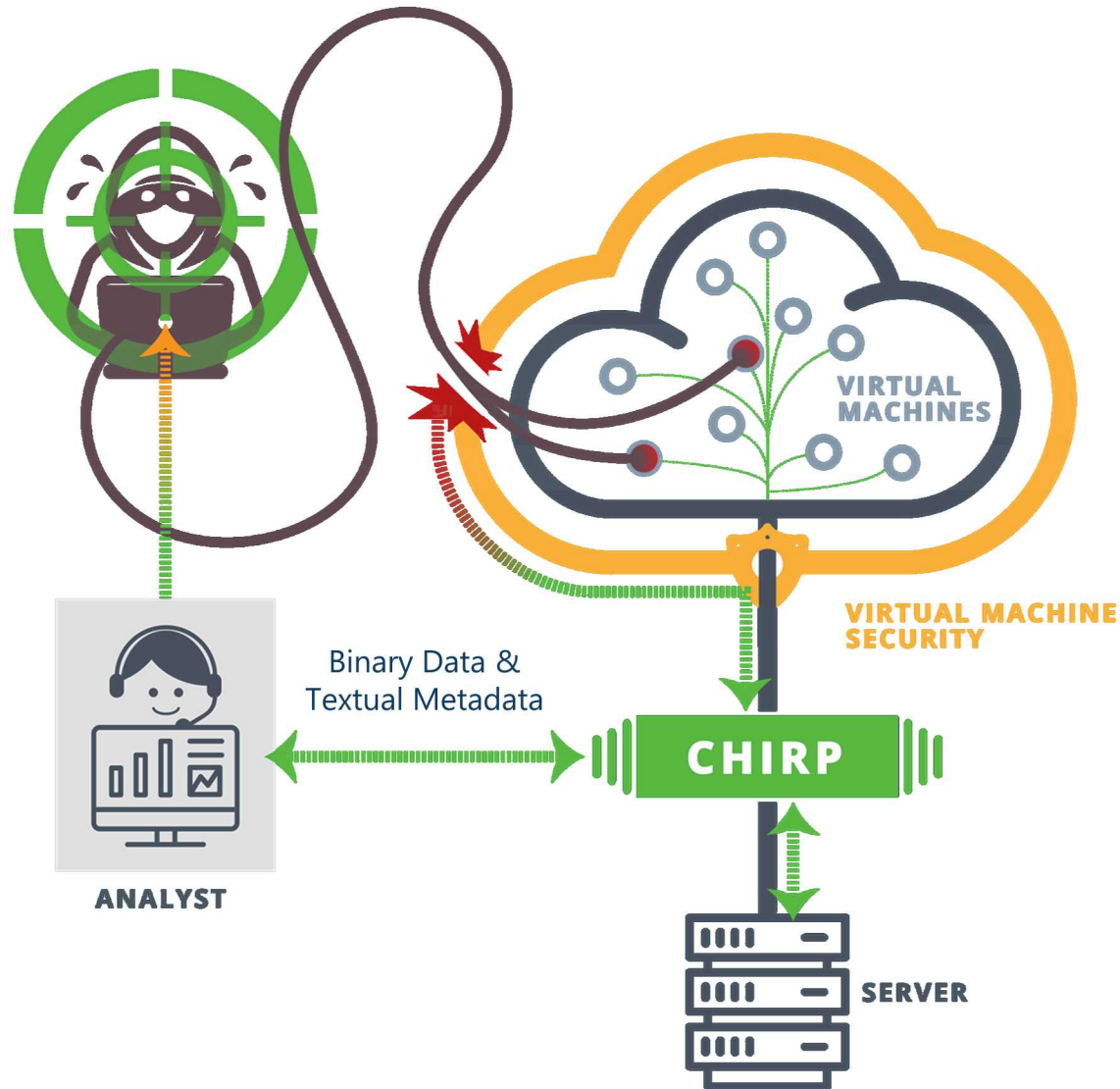
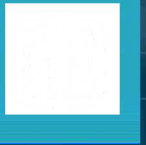
When cyber incidents occur in the cloud, the SOC Analyst has no visibility beyond the hypervisor.





- Ephemeral nature of cloud environment
- Hypervisor instrumentation with minimal overhead
- Hypervisor agnostic
- OS agnostic
- Common data model
- Broad data set collection





Virtual Machine Introspection (VMI)

- Sandia has developed a custom VMI implementation to address the issue of overhead.
- Platform agnostic
- Text and binary data collection
- Allows correlation with other sources

Development History & Results

CHIRP

Principal Investigator Vince Urias

- 20 Years of Cyber experience at Sandia Labs
- Numerous national awards

Developmental History

2016 Initial Development starts

2017 First disclosed (Dec)

2019 R&D 100 Winner (May)

Funding History
~\$1M LDRD



Research Team William Stout Caleb Loverro

Technology Readiness Level (TRL):
TRL 6 as of June 2020

IP Protection

- Patent Application # 16/051,005 filed July 31, 2018
- Copyright approved for commercial licensing – May 2018

Market Validation
2 government deployments



TECHNICAL REQUIREMENTS

Access to the Hypervisor

Designed for IaaS from the start

Off prem and on prem cloud-ready

BENEFITS

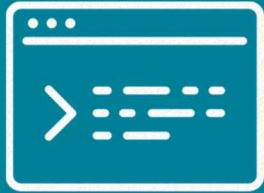
Lightweight

Real-time dynamic response

Configurable logging → incident response or forensics



Management
Dashboard



Expanded
Analytics



Improve
Deployment



Rigorous
Documentatio
n



Private sector
pilot for market
validation

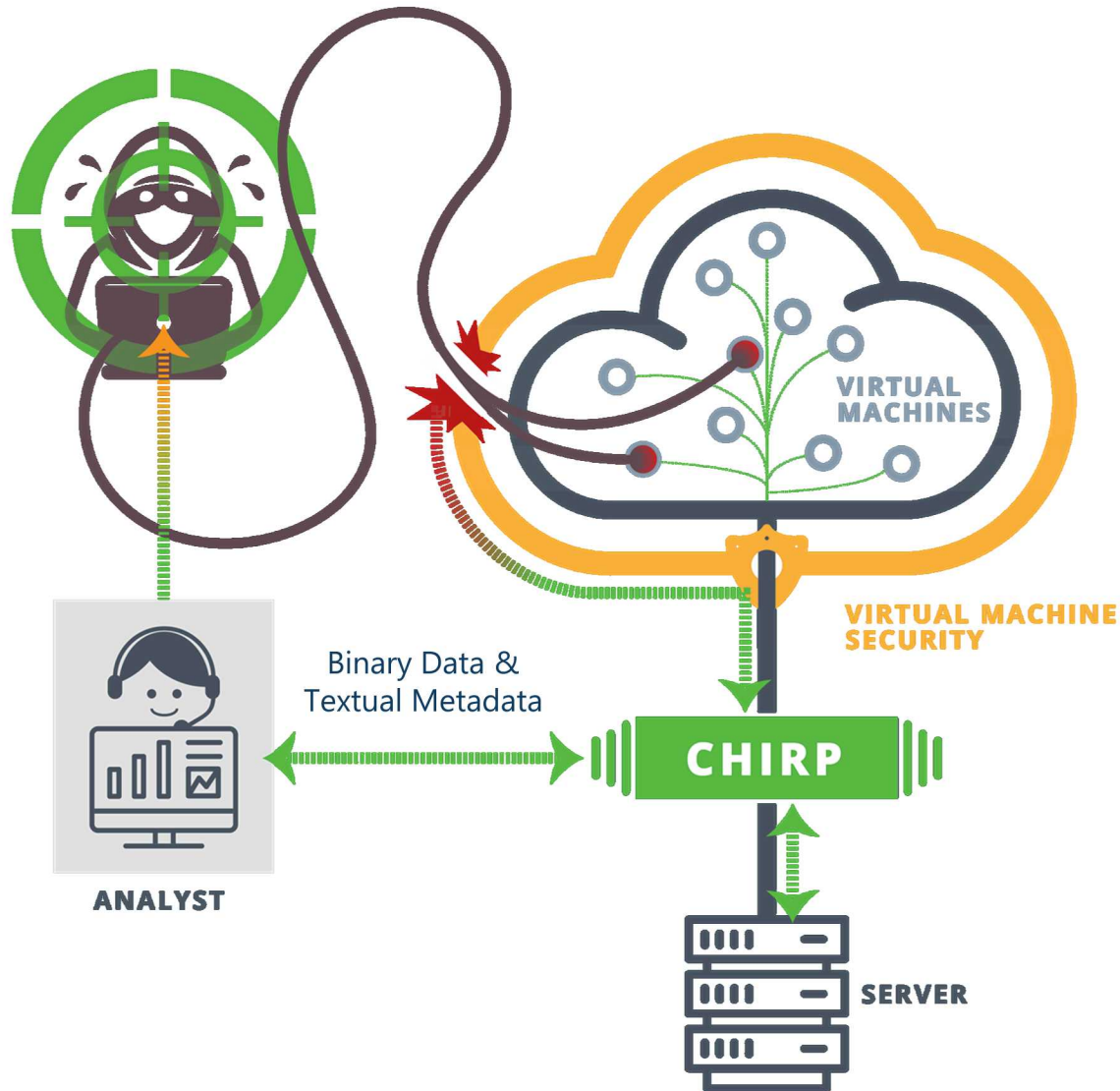
Bringing CHIRP to Market

DOE PACT VIRTUAL SHOWCASE
DEMONSTRATION

CHIRP

Cloud Hypervisor Forensics and Incident Response Platform

DOE PACT VIRTUAL SHOWCASE
DEMONSTRATION

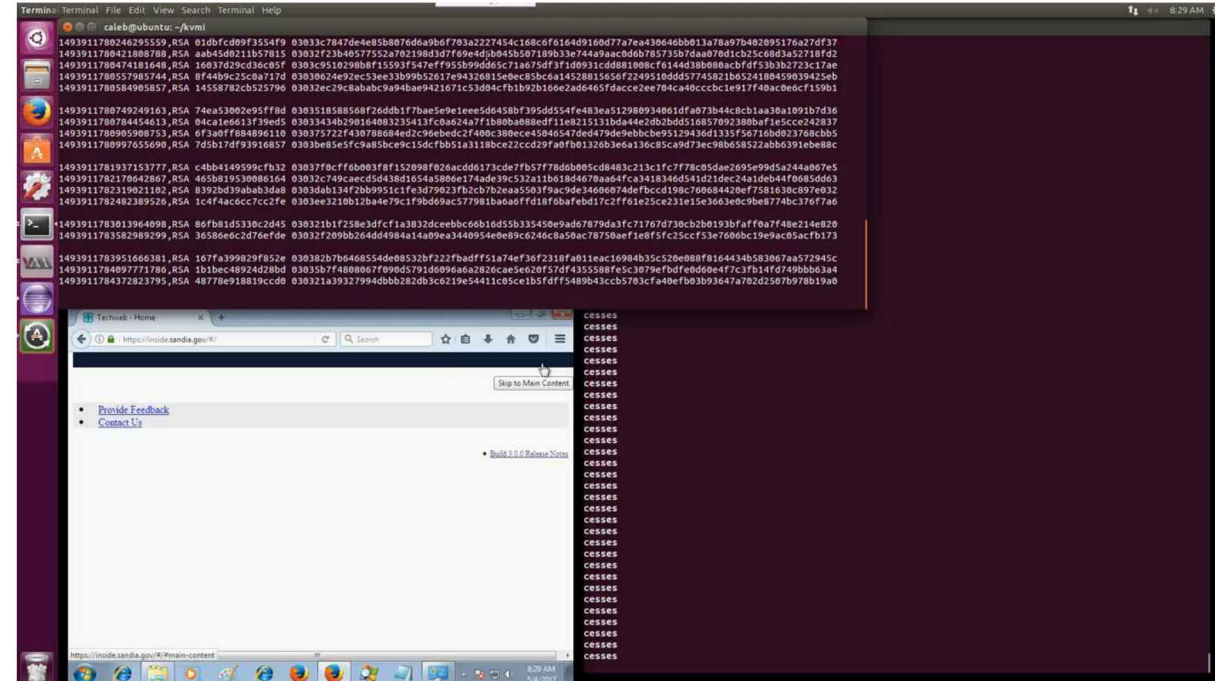
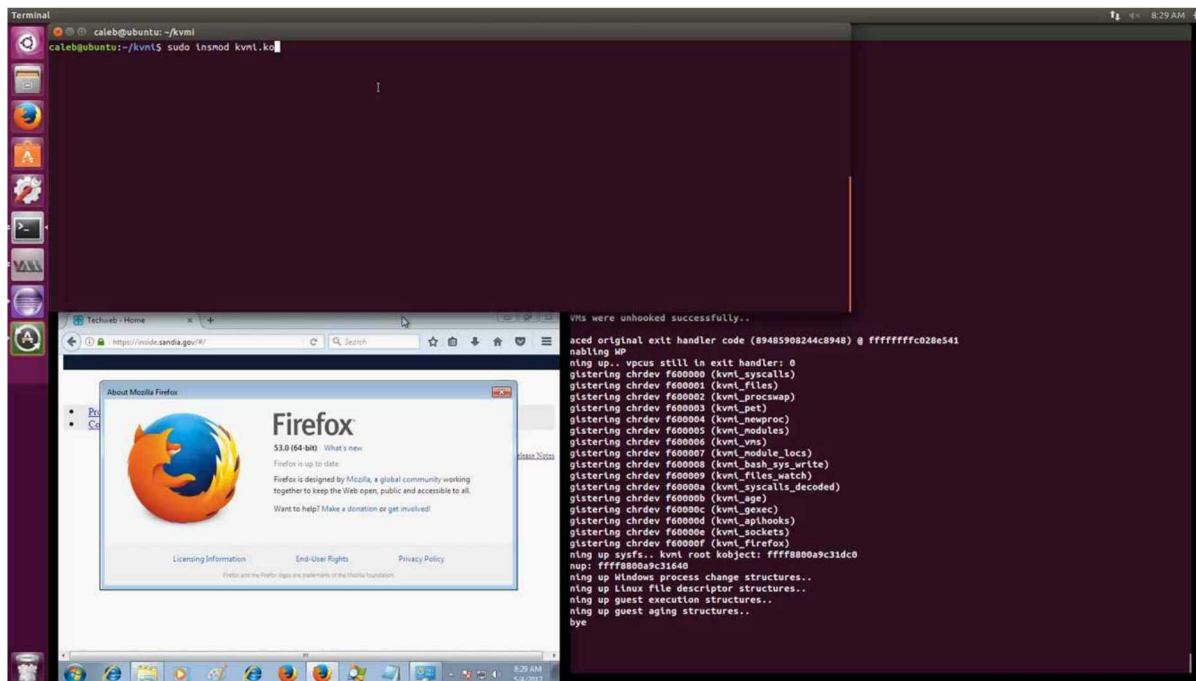


Vignette Demonstrations

- Module Dumping
- Key Extraction
- Shell
- Carving
- System Call Decoding
- Reverse Engineering Pipeline
- Big Data Extraction

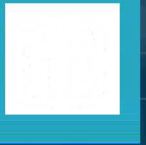
Key Extraction

CHIRP DEMONSTRATION



Keys used for encryption are extracted from processes (e.g., Firefox, Windows LSASS, etc.)
Here, Firefox browses to sandia.gov, RSA keys for the session are extracted.

Able to see within the session... eliminating man-in-the-middle attacks.



The screenshot displays two windows from the CHIRP demonstration. The left window, titled 'root@eadevnode: ~', shows a terminal session where the user has carved the 'vi' process from a VM. The terminal output includes a list of processes, a command to carve the 'vi' process, and the resulting memory dump of the 'vi' process. The right window, titled 'VNC: QEMU', shows the 'VIM - Vi IMproved' interface, version 7.4.1689, with a list of help commands.

```
root@eadevnode: ~
4081      0      0 kworker/0:2
4088      0      0 bash
4106      0      0 kworker/u2:2
4111      0      0 kworker/u2:0
4113      0      0 kworker/u2:1
4117      0      0 vi
28660     0      0 xfsalloc
28666     0      0 xfs_mru_cache
28673     0      0 jfsi0
28674     0      0 jfsCommit
28675     0      0 jfsSync
kvml[0]$ pscarve vi
kvml[0]$
Thank you for using KVM!!
root@eadevnode:~# ls
btrfs      iso          kvml.ko      linuxdumpprocs.py  shell.py  vms
dumpmodules.py  kernel  linuxdumpmodules.py  scripts    vi_4117
root@eadevnode:~# ls btrfs/
btrfs metadata
root@eadevnode:~# cat btrfs/metadata
module: btrfs
init: 0xffffffffc020698b
exit: 0xffffffffc01c6f21
num_syms: 0
guest virtual address: 0xffffffffc0113000
size: 0xf1000
text size: 0xb4000
read-only size: 0xc8000
root@eadevnode:~# ls kernel/
memory metadata
root@eadevnode:~# ls vi_4117/
0 1 2 3
root@eadevnode:~#
```

```
VIM - Vi IMproved
      version 7.4.1689
      by Bram Moolenaar et al.
Modified by pkg-vim-maintainers@lists.alioth.debian.org
Vim is open source and freely distributable

  Become a registered Vim user!

type  :help register<Enter>  for information

type  :q<Enter>              to exit
type  :help<Enter> or <F1>    for on-line help
type  :help version?<Enter>  for version info

0,0-1  All
```

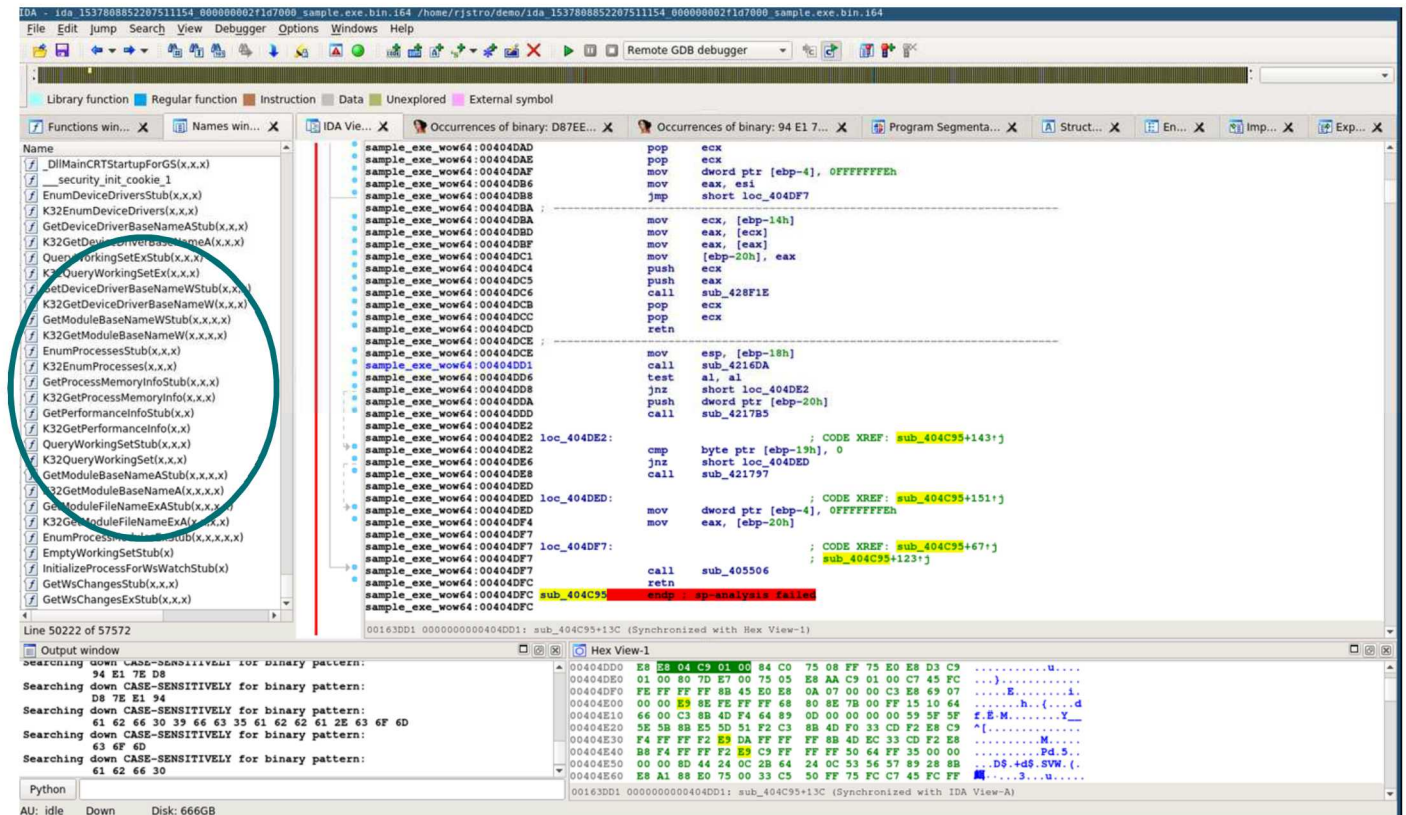
This example shows how the analyst can carve the VI text editor application (process) from the VM

Reverse Engineering Pipeline

CHIRP
DEMONSTRATION

- Track all memory for a process, including all libraries and OS code (not just the binary itself).
- Copy of all these mappings as binary that can be loaded into IDA, with multiple copies so program state can be analyzed at different times during execution.
- An idapython script is used to relocate all those mappings to correct locations, providing the reverse engineer deep insight into a given process.

Automatically gather symbols and library information, and correctly relocate (in memory)



The screenshot shows the IDA Pro interface with the following components:

- Functions window (left):** A list of functions including `_DlMainCRTStartupForGS(x,x,x)`, `_security_init_cookie_1`, `EnumDeviceDriversStub(x,x,x)`, `K32EnumDeviceDrivers(x,x,x)`, `GetDeviceDriverBaseNameAStub(x,x,x)`, `K32GetDeviceDriverBaseNameA(x,x,x)`, `QueryWorkingSetStub(x,x,x)`, `K32QueryWorkingSet(x,x,x)`, `GetDeviceDriverBaseNameWStub(x,x,x)`, `K32GetDeviceDriverBaseNameW(x,x,x)`, `GetModuleBaseNameWStub(x,x,x,x)`, `K32GetModuleBaseNameW(x,x,x,x)`, `EnumProcessesStub(x,x,x)`, `K32EnumProcesses(x,x,x)`, `GetProcessMemoryInfoStub(x,x,x)`, `K32GetProcessMemoryInfo(x,x,x)`, `GetPerformanceInfoStub(x,x,x)`, `K32GetPerformanceInfo(x,x,x)`, `QueryWorkingSetStub(x,x,x)`, `K32QueryWorkingSet(x,x,x)`, `GetModuleBaseNameAStub(x,x,x,x)`, `K32GetModuleBaseNameA(x,x,x,x)`, `GetModuleFileNameExStub(x,x,x,x)`, `K32GetModuleFileNameExA(x,x,x,x)`, `EnumProcessWorkingSetSet(x,x,x,x)`, `EmptyWorkingSetStub(x)`, `InitializeProcessForWsWatchStub(x)`, `GetWsChangesStub(x,x,x)`, and `GetWsChangesExStub(x,x,x)`.
- Assembly window (center):** Displays assembly code for `sample_exe_wow64:00404DAD` through `sample_exe_wow64:00404DFC`. The code includes instructions like `pop ecx`, `mov dword ptr [ebp-4], 0FFFFFFFh`, `mov eax, esi`, `jmp short loc_404DF7`, `mov ecx, [ebp-14h]`, `mov eax, [ecx]`, `mov [ebp-20h], eax`, `push ecx`, `push eax`, `call sub_428F1E`, `pop ecx`, `pop ecx`, `ret`, `mov esp, [ebp-18h]`, `call sub_4216DA`, `test al, al`, `jnz short loc_404DE2`, `push dword ptr [ebp-20h]`, `call sub_4217B5`, `cmp byte ptr [ebp-19h], 0`, `jnz short loc_404DED`, `call sub_421797`, `mov dword ptr [ebp-4], 0FFFFFFFh`, `mov eax, [ebp-20h]`, `call sub_405506`, and `ret`. There are also comments like `; CODE XREF: sub_404C95+143;j` and `; CODE XREF: sub_404C95+151;j`.
- Hex view window (bottom):** Displays the hex dump of the assembly code, showing the raw bytes and their corresponding ASCII representation.

Big Data Extraction



Feature	Linux	Windows	Mac
Raw syscalls	Y	Y	Y
Decoded syscalls	Y	Y	Y
PID/proc name extraction	Y	Y	Y
Guest execution	Y	Y	N
Kernel carving	Y	Y	N
Module carving	Y	Y	N
Process carving	Y	Y	N
Process tracking (start & exit)	Y	Y	N
File extraction	Y	Y	N
Biometrics	Y	N	N
Socket chardev	N	Y	N

Registry Key Access			
src_process	handle	registry_key	ret
sample.exe	184	Extensible Cache	STATUS_SUCCESS:0x0
sample.exe	428	Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache	STATUS_SUCCESS:0x0
sample.exe	428	Extensible Cache	STATUS_SUCCESS:0x0
sample.exe	184	Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache	STATUS_SUCCESS:0x0
sample.exe	184	Extensible Cache	STATUS_SUCCESS:0x0
sample.exe	428	Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache	STATUS_SUCCESS:0x0
sample.exe	428	Extensible Cache	STATUS_SUCCESS:0x0
sample.exe	184	Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache	STATUS_SUCCESS:0x0
sample.exe	184	Software\Microsoft\Windows NT\CurrentVersion\PeerDist\Service	STATUS_SUCCESS:0x0
sample.exe	184	\REGISTRY\MACHINE\SOFTWARE\Wow6432Node\Piriform\Agomo	STATUS_SUCCESS:0x0
sample.exe	184	{91150000-0011-0000-1000-0000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-012B-0409-1000-0000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-0117-0409-1000-0000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-0115-0409-1000-0000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00E2-0409-1000-0000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00E1-0409-1000-0000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00C1-0409-1000-0000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00C1-0000-1000-0000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00BA-0409-1000-0000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00A1-0409-1000-0000000FFICE}	STATUS_SUCCESS:0x0

Page Execution Coverage (Including Libraries)		
pages_of_code	pages_touched	percent_coverage
9621	1662	17.27471156844403
Running Processes		
pid	wow64	name
504	1	sample.exe
70	0	conhost.exe
31c	0	cmd.exe
294	0	rundll32.exe
7b0	0	ipconfig.exe

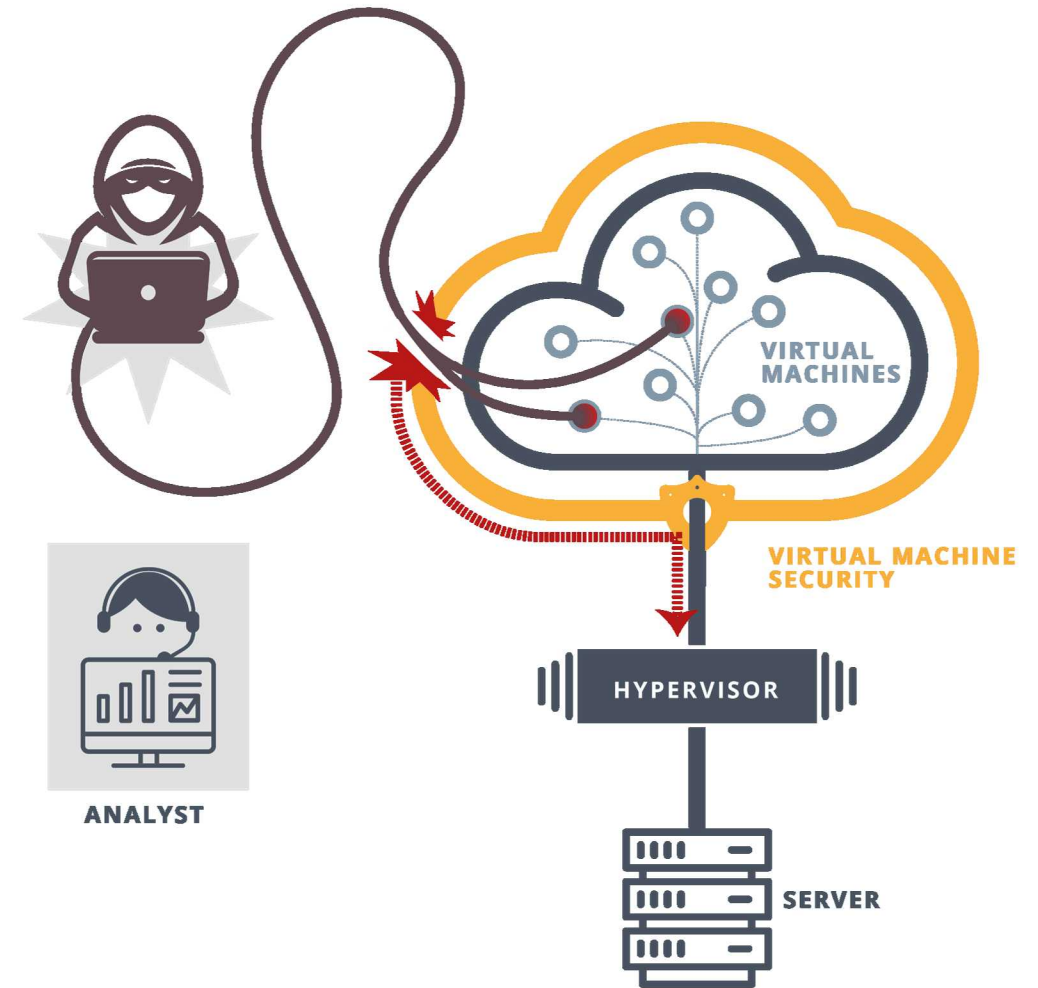
Page Execution Coverage (binary only)		
total_pages	pages_touched	percent_coverage
821	393	47.868453105968335
Loaded Modules		
fullname	size	loadcount
c:\windows\system32\aelupsvc.dll	15000	1
C:\Windows\system32\srvc11.dll	19000	2
C:\Windows\system32\netutils.dll	9000	1
C:\Windows\system32\netapi32.dll	11000	1
C:\Windows\system32\uxtheme.dll	80000	4

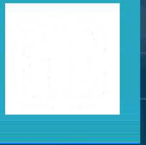
NtOpenProcess			
src_process	dst_process	access_mask	ret
sample.exe	conhost.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	cmd.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	GoogleUpdate.e	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	explorer.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	taskeng.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	taskhost.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	svchost.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	svchost.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	svchost.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	spoolsv.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0

Reserve Slides



Real-time forensics and **incident response** for Cloud Service Providers providing deeper analytics of security incidents—leading to better protection and **improved visibility**.





Virtual Machine Introspection (VMI)

- Sandia has developed a custom VMI implementation to address the issue of overhead.
- Platform agnostics
- Text and binary data collection
- Allows correlation with other sources

Cloud user accesses the cloud system to instantiate a VM

The CHIRP VMI module is loaded on the Cloud host

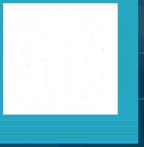
The CHIRP intercepts communication between guest and host

Binary data is extracted from the VMI and output to the host

Textual metadata is extracted from the VM and output to the host

Analyst accesses CHIRP to perform duties

CHIRP Process Flow



Cloud user accesses the Cloud system to instantiate a VM

VM specification is sent to a hypervisor on the host.

Hypervisor allocates resources to the guest VM and starts it.

The CHIRP VMI module is loaded on the Cloud host

CHIRP is injected into kernel-space of the host Cloud server

CHIRP detects hypervisor and places itself in the same administrative domain

The CHIRP intercepts communication between guest and host

CHIRP hooks VM-exit handler, assuming hypervisor functions

Dynamically detects all VM Operating Systems and state on the host

Binary data is extracted from the VM and output to the host

Binary data (memory, files, documents, malware, etc.) stored to the host

Binary data are analyzed

Textual metadata is extracted from the VM and output to the host

Textual content (e.g., logs) are ingested by a

Security Information and Event Management (SIEM) system

Textual data is correlated against other data (network, system) in the SIEM

Analyst accesses CHIRP to perform duties

Inspects binary data (executes files,
reverse engineers malware, etc.)

Inspects textual data to identify
indicators of compromise or attack

Modifies VM state as needed to
aid/prolong an investigation

Demonstration Outline



Vignette Demonstrations

- Module Dumping
- Key Extraction
- Shell
- Carving
- System Call Decoding
- Reverse Engineering Pipeline
- Big Data Extraction

Interactive Shell

The image displays three screenshots of a KVM interactive shell interface, showing the host's perspective and the guest's perspective.

Left Screenshot (Host View): The terminal shows the host's shell prompt `root@esdevnode:~`. The user has entered `./shell.py`, which has started an interactive shell. The prompt is now `kvml$`. The user has entered `first_time`, which displays the general workflow for the shell. The user has entered `list`, which displays a list of running VMs. The user has entered `select 0`, which has selected the first VM.

Middle Screenshot (Guest View): The terminal shows the guest's shell prompt `ubuntu@ubuntu1604tm:~$`. The user has entered `ls`, which displays the contents of the current directory.

Right Screenshot (Guest View): The terminal shows the guest's shell prompt `ubuntu@ubuntu1604tm:~$`. The user has entered `ps`, which displays the running processes on the VM.

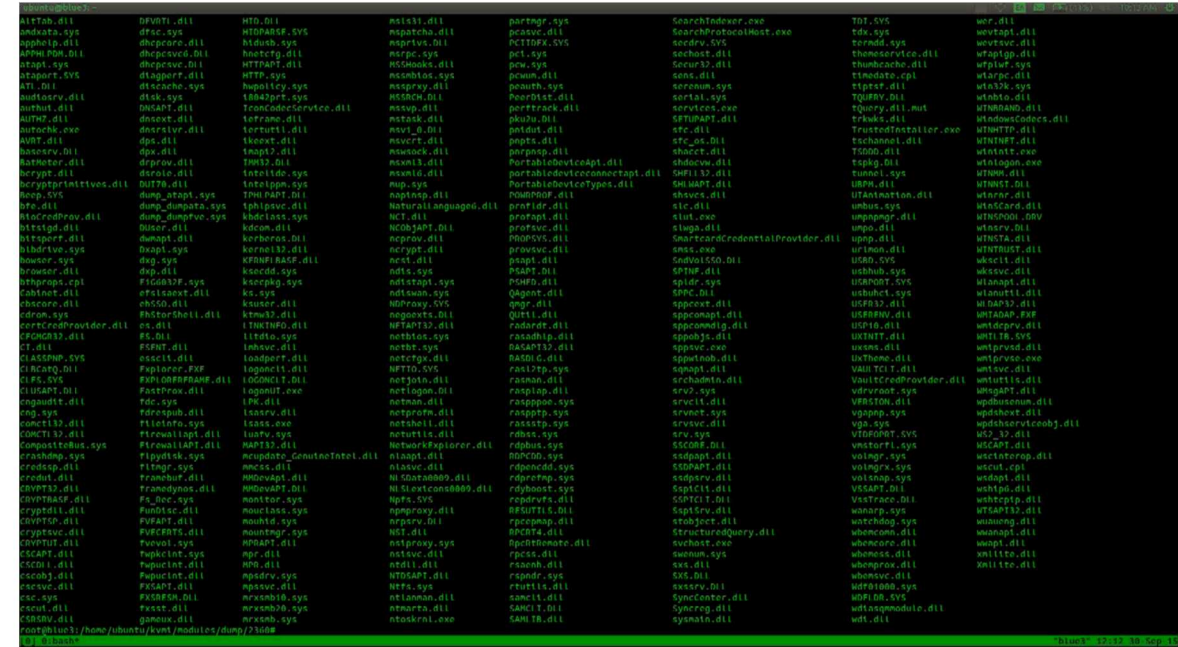
An interactive shell can be used by the analyst to gather information non-obtrusively from the VM in real-time.

- List VMs
- List libraries in use
- List modules executed
- List file descriptors
- List running processes
- Write into memory/process
- Start processes on the VM
- Carve objects from memory
 - Files
 - Operating system kernel

the 1990s, the number of people in the world who are illiterate has increased from 1.2 billion to 1.5 billion. The number of illiterate people in the world is projected to reach 1.7 billion by the year 2015. The number of illiterate people in the world is projected to reach 1.7 billion by the year 2015. The number of illiterate people in the world is projected to reach 1.7 billion by the year 2015.

```
root@eadevnode:~  
:mh=00;pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su  
=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.*tar=01;31:*.*tgz  
=01;31:*.*arc=01;31:*.*arj=01;31:*.*taz=01;31:*.*lha=01;31:*.*lzma=01;31:*.  
*.lzm=01;31:*.*tlz=01;31:*.*txz=01;31:*.*tzx=01;31:*.*tzo=01;31:*.*zip=01;31:  
*:Z=01;31:*.*dz=01;31:*.*gz=01;31:*.*lrz=01;31:*.*lz=01;31:*.*lzo=01;31:*.*xz=  
01;31:*.*bz2=01;31:*.*bz=01;31:*.*tbz=01;31:*.*tbz2=01;31:*.*tz=01;31:*.*deb=01;  
.rpm=01;31:*.*jar=01;31:*.*war=01;31:*.*ear=01;31:*.*sar=01;31:*.*rar=01;31:*.*alz=0  
1;31:*.*ace=01;31:*.*zoo=01;3...  
1507066489871185216,os=linux,sysret,vcpuid=0x1,process="ls",pid=1499,current_c  
r3=0x232941000,syscall_cr3=0x232941000,syscall_num=12,brk(0x0) = 16924672  
1507066489871276703,os=linux,sysret,vcpuid=0x1,process="ls",pid=1499,current_c  
r3=0x232941000,syscall_cr3=0x232941000,syscall_num=21,access("/etc/d.l.so.nohwc  
ap",-2)=-2  
1507066489871311517,os=linux,sysret,vcpuid=0x1,process="ls",pid=1499,current_c  
r3=0x232941000,syscall_cr3=0x232941000,syscall_num=9,mmap(0x0, 8192, PROT_READ  
|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,-1, 0)=0x7f59761a0000  
1507066489871352172,os=linux,sysret,vcpuid=0x1,process="ls",pid=1499,current_c  
r3=0x232941000,syscall_cr3=0x232941000,syscall_num=21,access("/etc/d.l.so.prelo  
ad", 0x4 /* ?_OK */)=-2  
1507066489871390046,os=linux,sysret,vcpuid=0x1,process="ls",pid=1499,current_c  
r3=0x232941000,syscall_cr3=0x232941000,syscall_num=2,open("/etc/d.l.so.cache"  
,"O_CLOEXEC, 0x1")=3  
1507066489871425128,os=linux,sysret,vcpuid=0x1,process="ls",pid=1499,current_c  
r3=0x232941000,syscall_cr3=0x232941000,syscall_num=5,fstat(3,{fst_dev=makedev(  
252, 0), st_ino=132987,S_IFREG|0644,st_nlink=1,st_uid=0,st_gid=0,st_blkis  
ize=4096,st_blocks=48,st_size=20501,st_atime="1507064516",st_mtime_nsec="92  
4000000",st_mtime="1409582889",st_mtime_nsec="21200000",st_ctime="14095828  
89",st_ctime_nsec="21200000"))=0  
1507066489871474389,os=linux,sysret,vcpuid=0x1,process="ls",pid=1499,current_c  
r3=0x232941000,syscall_cr3=0x232941000,syscall_num=9,mmap(0x0, 20501, PROT_REA  
D, MAP_PRIVATE, 3, 0)=0x7f597619a000  
1507066489871505029,os=linux,sysret,vcpuid=0x1,process="ls",pid=1499,current_c
```

This example shows how an analyst can further watch actions on the VM through system call analysis. Decoded system calls with parameters are shown in the shell, with an strace of the 'ls' program in the right side (VM).



From boot up onward, CHIRP captures valuable information.